RESEARCH INNOVATION DISCOVERY

Reg.No: 048884
Foundation Day
05-09-2024

RID संस्था समस्या का समाधान

RUN BY TWKSAA WELFARE FOUNDATION

Django E-Book

(BACKEND E-Book

Er. RP Sir (B.E, M.E)
Founder: TWF & RID Org.

- **RID BHARAT** यानि **Research, Innovation and Discovery** संस्था जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले **NEW (RID, PMS & TLR)** की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान **NEW (RID, PMS & TLR)** के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद **इस RID संस्था** की स्थपना किया हूँ |
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी **इस RID संस्था** से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई **(RID, PMS & TLR)** की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

RID BHARAT

"A Place for Coders, Developers & Researchers"

Online & Offline Class:

Python Full Stack, Web Development, Java, UI/UX, Data Science, AI Training, Internships & Placements

करने के लिए Message/Call करें. 9202707903 E-Mail_id: ridorg.in@gmail.com

Website: www.ridtech.in

RID हमें क्यों करना चाहिए ?

(Research)

अनुसंधान हमें क्यों करना चाहिए ?

Why should we do research?

1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge)
2. समस्याओं का समाधान(To Solving problems)
3. सामाजिक प्रगति (To Social progress)
4. विकास को बढ़ावा देने(To promote development)
5. तकनीकी और व्यापार में उन्नति(To advances in technology & business)
6. देश विज्ञान और प्रौद्योगिकी के विकास(To develop the country's science & technology)

(Innovation)

नवीनीकरण हमें क्यों करना चाहिए ?

Why should we do Innovation?

1. प्रगति के लिए(To progress)
2. परिवर्तन के लिए(For change)
3. उत्पादन में सुधार(To Improvement in production)
4. समाज को लाभ(To Benefit to society)
5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition)
6. देश विज्ञान और प्रौद्योगिकी के विकास।
(To develop the country's science & technology)

(Discovery)

खोज हमें क्यों करना चाहिए?

Why should we do Discovery?

1. नए ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. अविष्कारों की खोज(To Discovery of inventions)
3. समस्याओं का समाधान(To Solving problems)
4. ज्ञान के विकास में योगदान(Contribution to development of knowledge)
5. समाज के उन्नति के लिए (for progress of society)
6. देश विज्ञान और तकनीक के विकास
(To develop the country's science & technology)

New Technology पर **Research** करने के लिए सम्पर्क करें: **info@ridbharat.com Mob.No: 9202707903**

❖ Research(अनुसंधान):

- अनुसंधान एक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोधोगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उचित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ Innovation(नवीनीकरण): -

- Innovation एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ Discovery (आविष्कार):

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

नोट : अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिस्टमैटिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

सुविचार:

1.	**समस्याओं का समाधान करने का उतम मार्ग हैं |**	→ शिक्षा ,RID, प्रतिभा, सहयोग, एकता एवं समाजिक-कार्य
2.	**एक इंसान के लिए जरूरी हैं |**	→ रोटी, कपड़ा, मकान, शिक्षा, रोजगार, इज्जत और सम्मान
3.	**एक देश के लिए जरूरी हैं |**	→ संस्कृति-सभ्यता, भाषा, एकता, आजादी, संविधान एवं अखंडता
4.	**सफलता पाने के लिए होना चाहिए |**	→ लक्ष्य, त्याग, इच्छा-शक्ति, प्रतिबद्धता, प्रतिभा, एवं सतता
5.	**मरने के बाद इंसान छोडकर जाता हैं |**	→ शरीर, अन-धन, घर-परिवार, नाम, कर्म एवं विचार
6.	**मरने के बाद इंसान को इस धरती पर याद किया जाता हैं उनके**	

➔ नाम, काम, दान, विचार, सेवा-समपर्ण एवं कर्मो से...

आशीर्वाद (बड़े भैया जी)

Mr. RAMASHANKAR KUMAR

मार्गदर्शन एवं सहयोग

Mr. GAUTAM KUMAR

.....सोच है जिनकी नये कुछ कर दिखाने की, खोज हैं मुझे आप जैसे इंसान की.....

"अगर आप भी **Research, Innovation and Discovery** के क्षेत्र में रूचि रखतें हैं एवं अपनी प्रतिभा से दुनियां को कुछ नया देना चाहतें एवं अपनी समस्या का समाधान **RID** के माध्यम से करना चाहतें हैं तो **RID ORGANIZATION (रीड संस्था)** से जरुर जुड़ें" || धन्यवाद || **Er. Rajesh Prasad (B.E, M.E)**

Index

Introduction to Django

- Django is a high-level Python web framework used for building secure, scalable, and maintainable web applications.
- It follows the Model-View-Template (MVT) architecture and provides built-in features like authentication, ORM (Object-Relational Mapping), admin panel, and security mechanisms.
- Django is widely used for developing websites, APIs, and web applications quickly with minimal code.

❖ Advantage of Django

- Fast Development – Uses DRY (Don't Repeat Yourself) principle for quick coding.
- **Scalable** – Handles high traffic efficiently.
- **Secure** – Protects against SQL injection, XSS, CSRF, etc.
- **Built-in Admin Panel** – Easy data management.
- **ORM Support –** Simplifies database operations.
- **Rich Libraries** – Offers reusable apps and plugins.
- **SEO-Friendly** – Supports clean URLs.
- **Cross-Platform** – Works on various OS and databases.

❖ Applications Developed Using Django

- **Social Media Platforms –** Example: Instagram
- **E-commerce Websites –** Example: Shopify
- **News & Blogging Sites –** Example: The Washington Post
- **Healthcare & Medical Apps –** Example: OpenMRS
- **Finance & Banking Apps –** Example: Robinhood
- **E-learning Platforms –** Example: edX

❖ PIP (Preferred Installer Program)

- PIP (Preferred Installer Program) is the package manager for Python used to install, update, and manage Python libraries and dependencies from the Python Package Index (PyPI).
- It allows developers to easily install external packages using a simple command like:

```
>> pip install package_name
```

For example, to install Django:

```
>> pip install django
```

Process to Learn Django

1. Learn Python Basics

- Variables, Data Types, Functions
- Loops & Conditional Statements
- Object-Oriented Programming (OOP)
- Working with Modules & Packages

2. Install Django & Set Up Environment

- Install Python & PIP
- **Install Django using** >> pip install django
- **Create a new Django project** >> django-admin startproject project_name
- **Run the development server** >> python manage.py runserver

3. Understand Django Project Structure

- **settings.py –** Project Configuration
- **urls.py –** URL Routing
- **views.py –** Handles Requests & Responses
- **models.py –** Database Management

4. Learn MVT (Model-View-Template) Architecture

- **Model –** Handles database operations
- **View –** Processes requests & returns responses
- **Template –** Manages HTML rendering

5. Work with Django Models & ORM

- Define models in models.py
- Migrate changes using >> python manage.py makemigrations

  >> python manage.py migrate
- Use Django ORM to interact with the database

6. Learn URL Routing & Views

- Define URLs in urls.py
- Create views in views.py

7. Use Django Templates for Frontend

- Template Tags & Filters
- Static Files (CSS, JS, Images)

8. Handle Forms & Validations

- Django Forms (forms.py)
- Form Validation & Input Handling

9. Implement Authentication & User Management

- User Registration & Login
- Session & Authentication System

10. Work with Django Admin Panel

- Create a Superuser
- python manage.py createsuperuser

- Customize Admin Dashboard

11. Build a REST API with Django REST Framework (DRF)

Install DRF

- pip install djangorestframework
- Create APIs using serializers & views

12. Learn Middleware & Security Features

- CSRF Protection, Authentication Middleware

13. Deploy Django Project

- Deploy on Heroku, AWS, or DigitalOcean
- Use Gunicorn & Nginx for production

14. Work on Real Projects

- Build blogs, e-commerce sites, social media apps

❖ Important Topics to Learn in Django

1. Django Installation & Setup
2. Project & App Structure
3. MVT (Model-View-Template) Architecture
4. URL Routing & Views
5. Models & ORM (Object-Relational Mapping)
6. Django Forms & Validations
7. Templates & Template Tags
8. Static Files & Media Handling
9. Authentication & User Management
10. Django Admin Panel
11. Middleware & Request Handling
12. Django REST Framework (DRF) for APIs
13. Database Migrations
14. Security Features (CSRF, XSS Protection, etc.)
15. Deployment & Hosting (Gunicorn, Nginx, Heroku, etc.)

❖ Django Basic Command

1. How to check the python is install or not

Syntax: python --version

Example: C:\Users\rid>python --version

Python 3.13.2

2. How to check the pip is install or not

Syntax: pip --version

Example: C:\Users\rid>pip --version

pip 24.3.1

3. How to Install Django (if not already installed)

Syntax: pip install Django

Example: C:\Users\rid>pip install django

Requirement already satisfied: django in c:

4. How to check the Django is install or not

Syntax: django-admin --version (for cmd and for bash)

Syntax: python -m django --version

Example: C:\Users\rid>django-admin version

5.1.6

5. How to check all library that are install in python

Syntax: pip list or

Example: C:\Users\rid>pip list

Syntax: pip freeze

C:\Users\rid>pip freeze

asgiref==3.8.1

Django==5.1.6

sqlparse==0.5.3

tzdata==2025.1

6. How to check the all Django management commands

If you are using `django-admin` (without a project setup):

Syntax: django-admin

Example: C:\Users\rid>django-admin

Available subcommands:

[django]

- check
- dbshell
- diffsettings
- dumpdata
- flush
- inspectdb
- loaddata
- makemessages
- makemigrations
- migrate
- optimizemigration
- runserver
- sendtestemail
- shell
- showmigrations
- sqlflush
- sqlmigrate
- sqlsequencereset
- squashmigrations
- startapp
- startproject
- test
- testserver

Note that only Django core commands are listed as settings are not properly configured (error: Requested setting INSTALLED_APPS, but settings are not configured. You must either

7. How to view the specific command details

Syntax: python manage.py <command> --help
Example: django-admin startapp –help
after Django project create
Example-2: python manage.py migrate --help

8. How to create the Django Project

Syntax: django-admin startproject projectname
Example: C:\Users\rid>django-admin startproject myproject
It will create a folder named `myproject` in the current directory (`C:\Users\rid\`)
Step-1 Navigate to your project directory
Syntax: cd projectname
Example: C:\Users\rid>cd myproject
After change your project name created directory you can start the server

9. How to Run the development server (to see if everything works)

Syntax: python manage.py runserver
Example: C:\Users\rid\myproject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
Django version 5.1.6, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Note: open this http://127.0.0.1:8000/ in your browser the see Django project

10. How to change the port number

Syntax: python manage.py runserver <IP address>:<port number>
Example: python manage.py runserver 127.0.0.1:8080

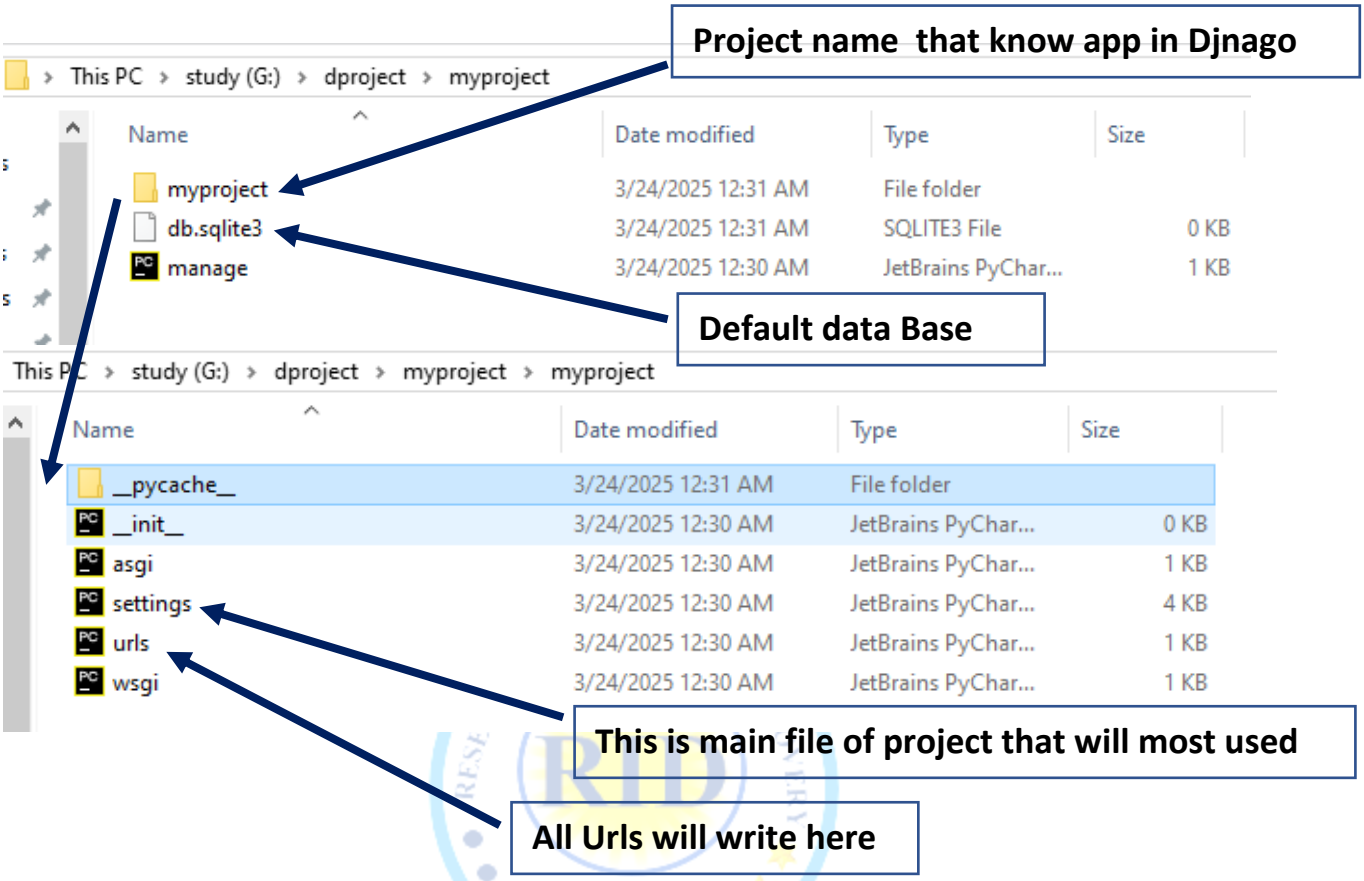
Or

Example: C:\Users\rid\myproject>python manage.py runserver 8080
Watching for file changes with StatReloader
Performing system checks...
Django version 5.1.6, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8080/
Note Now you can open this Urls http://127.0.0.1:8080/ in browser

11. How we can stop the server

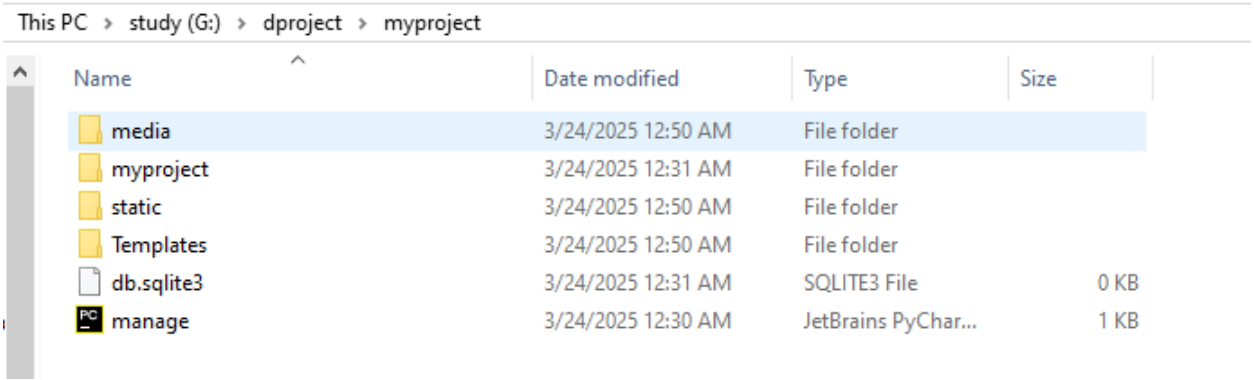
Using Keyboard Shortcut (For Local Server)
>> Press **CTRL + C**

Folder & File Structure in Djnago



❖ **Now we need to create three different folders**

1. **Media**
2. **Static**
3. **templates**



1. media/:

- **Purpose:** This folder is where user-uploaded files (like images, documents, etc.) are stored.
- **Example:** If you allow users to upload profile pictures, the images would be stored here.
- **Config:** The MEDIA_ROOT and MEDIA_URL settings in Django's settings file control the location and access URL for media files.

2. db.sqlite3:

pose: This is the default SQLite database file created by Django when you run migrate for the first time.

mple: Stores all your app data like users, posts, comments, etc.

fig: You can use a different database backend (e.g., PostgreSQL, MySQL) by configuring the DATABASES setting in your ngo settings file.

3. manage.py:

- **Purpose:** This is the command-line utility that lets you interact with your Django project.
- **Example:** You use this file to run commands like runserver, migrate, makemigrations, etc.
- **Usage:** You would run python manage.py <command> to execute a management command.

4. templates/:

- **Purpose:** This folder is where your HTML files (templates) are stored.
- **Example:** It might contain pages like index.html, about.html, etc., which are rendered by Django when you return views to users.
- **Config:** The TEMPLATES setting in Django's settings file points to this directory.

5. static/:

- **Purpose:** This folder stores static files like CSS, JavaScript, images, and fonts that are used for styling and interactivity in your frontend.
- **Example:** Your styles.css or app.js files would be placed here.
- **Config:** The STATIC_URL and STATICFILES_DIRS settings in Django control how static files are handled.

Folder Structure

```
myproject/
  manage.py
  db.sqlite3
  media/
  templates/
    index.html
  static/
    css/
      styles.css
    js/
      app.js
```

Main Project Structure:

```
myproject/
│
├── manage.py              # Command-line utility to interact with the project
├── myproject/             # Main project folder (also named after your
project)
│   ├── __init__.py        # Marks this directory as a Python package
│   ├── settings.py        # Django settings for the project (configurations)
│   ├── urls.py            # URL routing for the project
│   ├── asgi.py            # ASGI configuration for asynchronous communication
│   └── wsgi.py            # WSGI configuration for deploying to web servers
│
├── app1/                  # Example of a Django app (you can have multiple
apps)
│   ├── __init__.py
│   ├── admin.py           # Django admin configurations for this app
│   ├── apps.py            # App configurations
│   ├── models.py          # Database models for this app
│   ├── views.py           # Views to handle requests and return responses
│   ├── urls.py            # App-specific URL routing
│   ├── migrations/        # Folder to store migration files
│   └── templates/         # Template folder for this app
│       └── app1/          # Sub-folder (app-specific) for HTML templates
│           └── home.html
│
├── db.sqlite3             # Default SQLite database file (if using SQLite)
├── media/                 # Directory for user-uploaded files (e.g., images,
videos)
├── static/                # Folder for static files (CSS, JS, images)
│   ├── app1/              # App-specific static files (optional)
│   └── css/
│       └── styles.css     # Example of a static CSS file
│
└── templates/             # Global templates used by the entire project
(optional)
    └── base.html          # Example of a global base template
```

Detailed Explanation:

1. **`manage.py`:**
   - A command-line tool that helps interact with your Django project. You can use it to run the server, make migrations, apply migrations, etc.
2. **`myproject/` (Main project directory):**
   - This is the project configuration folder. It is named after your project and contains important files for settings, URLs, and server configuration:
   - `settings.py`: Contains all the configuration for your project, like database settings, installed apps, and middleware.
   - `urls.py`: Defines URL routes and maps them to views for the whole project.
   - `asgi.py` & `wsgi.py`: Server gateway interface (WSGI) and Asynchronous Server Gateway Interface (ASGI) configurations for deploying Django.

3. **App Directories (e.g., `app1/`)**:
    - A Django project consists of one or more "apps" that provide functionality to your project. Each app is self-contained, with its own models, views, templates, etc.
    - `admin.py`: Register models to appear in Django's admin interface.
    - `models.py`: Define the data models for the app.
    - `views.py`: Handles logic for responding to requests and rendering templates.
    - `urls.py`: App-specific URL routing (can be included in the main `urls.py`).
    - `migrations/`: Stores database migration files for the app.
    - `templates/`: Contains HTML files specific to this app.
4. **`db.sqlite3`**:
    - This is the default SQLite database used by Django for storing app data. If you're using a different database, this file may not exist (e.g., if using PostgreSQL or MySQL).
5. **`media/`**:
    - This folder stores files that users upload (e.g., profile images, documents). You need to configure `MEDIA_URL` and `MEDIA_ROOT` in `settings.py`.
6. **`static/`**:
    - Stores static files like CSS, JavaScript, and image files for the frontend.
    - The `static/` folder can contain app-specific folders (e.g., `app1/`) for each app's static files.
7. **`templates/`** (Optional, project-level):
    - A global templates folder that contains base templates or layout templates shared across multiple apps.
    - For example, you could have a `base.html` template with common HTML structure (header, footer) shared by all other templates.

setting.py

1. BASE_DIR:- The base directory of your Django project (used for path references).
2. SECRET_KEY:- A secret string used for encryption and security — must be kept confidential.
3. DEBUG:- Enables detailed error messages during development (True); should be False in production.
4. ALLOWED_HOSTS:- List of domains/IPs allowed to access the app.
5. INSTALLED_APPS:- List of Django apps and components used in the project.
6. MIDDLEWARE:- A list of classes that process requests and responses (security, sessions, etc.).
7. ROOT_URLCONF:- The path to the main URL configuration module.
8. TEMPLATES:- Settings that control how HTML templates are loaded and rendered.
9. WSGI_APPLICATION:- The WSGI entry point for deploying the Django app.
10. DATABASES:- Defines which database your project uses (default is SQLite).
11. AUTH_PASSWORD_VALIDATORS:- Rules to validate the strength and security of user passwords.
12. LANGUAGE_CODE:- Default language used in the project ('en-us').
13. TIME_ZONE:- Sets the default time zone ('UTC').
14. USE_I18N:- Enables internationalization (multiple language support).
15. USE_TZ:- Enables timezone-aware datetime support.
16. STATIC_URL:- URL prefix for accessing static files like CSS and JS.
17. DEFAULT_AUTO_FIELD:- Default type for primary key fields in models (BigAutoField).

❖ What is MIDDLEWARE in Django?

Middleware in Django is a **layer between the request and response**. It's a series of **hooks or functions** that are processed **before and after** Django views.

❖ Why do we use Middleware?

Middleware is used to **process requests and responses globally** across the project. It helps in:

1. **Security** – e.g., SecurityMiddleware adds security headers.
2. **Session Handling** – e.g., SessionMiddleware manages session data.
3. **Authentication** – e.g., AuthenticationMiddleware checks logged-in users.
4. **CSRF Protection** – e.g., CsrfViewMiddleware helps prevent cross-site request forgery.
5. **Message Handling** – e.g., MessageMiddleware supports flash messages.
6. **Request/Response Modification** – e.g., adding or editing headers.

❖ How it works?

- **Before View**: Middleware processes the incoming HTTP request.
- **After View**: Middleware processes the outgoing HTTP response.

Example Middleware List:

```
MIDDLEWARE = [
   'django.middleware.security.SecurityMiddleware',
   'django.contrib.sessions.middleware.SessionMiddleware',
   'django.middleware.common.CommonMiddleware',
   'django.middleware.csrf.CsrfViewMiddleware',
   'django.contrib.auth.middleware.AuthenticationMiddleware',
   'django.contrib.messages.middleware.MessageMiddleware',
]
```

Each middleware is executed **in order** (top to bottom on request, bottom to top on response).

❖ What are templates in Django?

TEMPLATES tells Django how to find and render HTML templates.

Why use it?

It configures:

- Which template engine to use (BACKEND)
- Where to find templates (DIRS)
- Whether to look inside apps automatically (APP_DIRS)
- Extra data to pass to templates (context_processors)

Key parts:

- 'BACKEND': Django's template engine
- 'DIRS': List of folders for templates outside apps
- 'APP_DIRS': True: Search app folders for templates
- 'context_processors': Add useful data (like user info) to templates

Example:

With 'APP_DIRS': True and an app in INSTALLED_APPS,
render(request, 'home.html') finds home.html inside myapp/templates/.

How To add your own custom templates in Django

- **1. Create a folder for templates**

You can create a folder named templates in your project root (same level as manage.py), like this:

```
mypro/
|
├── myapp/
├── templates/      ← your custom template folder
|   └── home.html
├── manage.py
```

- **2. Update 'DIRS' in settings.py** Find the TEMPLATES section and update 'DIRS':

```
from pathlib import Path
BASE_DIR = Path(__file__).resolve().parent.parent

TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR / 'templates'],  # 👈 Add this line
    'APP_DIRS': True,
    'OPTIONS': {
      'context_processors': [
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
      ],
    },
  },
]
```

Database connection in Django

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'db.sqlite3',
  }
}
```

In Django, DATABASES sets your database config.

- ENGINE: Type of database (e.g., 'sqlite3' for SQLite).
- NAME: Database name or file path (db.sqlite3 for SQLite).

It tells Django where and how to store data. By default, it uses SQLite for easy setup.

To use your own **MySQL database** in Django, update the DATABASES setting like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',      # Use MySQL engine
        'NAME': 'your_db_name',                    # Your MySQL database name
        'USER': 'your_username',                   # MySQL username
        'PASSWORD': 'your_password',               # MySQL password
        'HOST': 'localhost',                       # Database server address
        'PORT': '3306',                            # MySQL default port
    }
}
```

Steps:

1. Install MySQL database server and create a database.
2. Install MySQL Python connector:

pip install mysqlclient

or

pip install pymysql

3. Update DATABASES in settings.py as shown above.
4. Run migrations to create tables:

python manage.py migrate

Now Django will use your MySQL database instead of SQLite.

Example:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'ecommerce_db',                    # Your real database name
        'USER': 'admin',                           # Your MySQL username
        'PASSWORD': 'admin123',                    # Your MySQL password
        'HOST': 'localhost',                       # Usually localhost if running locally
        'PORT': '3306',                            # Default MySQL port
    }
}
```

Note: This config connects Django to a MySQL database named ecommerce_db with user admin on your local machine. Use this for real projects needing MySQL instead of SQLite.

Migrations

- **Migrations in Django** are like version control for your database schema.
- They **track changes** you make to your models (like adding or modifying fields).
- When you run migrations, Django **applies those changes** to your actual database (creating or altering tables or perform curd operation in database).
- This helps keep your database structure **in sync** with your Django models easily.

Note: - Migrations = automatic updates to your database schema based on model changes.

Migrate

- In Django, **migrate** is a command that **applies migrations** to your database.
- It used for **creates or updates database tables** based on your models and migration files.
- Keeps your database structure **in sync with your Django app's models**.
- You run it after making model changes to update the actual database.

Note: - migrate = execute changes in the database according to migration files.

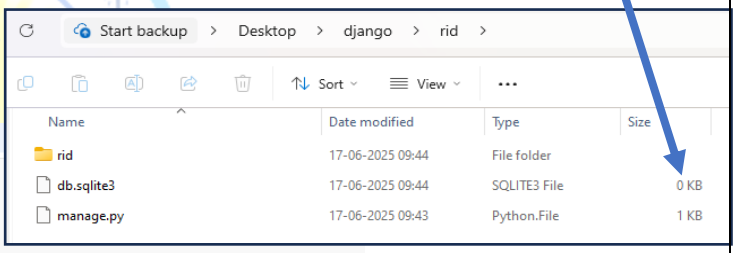
How to Use?

1. Change your models (e.g., add a new field).
2. Run:
   - ➢ python manage.py makemigrations ***# Save changes***
   - ➢ python manage.py migrate ***# Update database***

That's it! Keeps your database updated automatically.

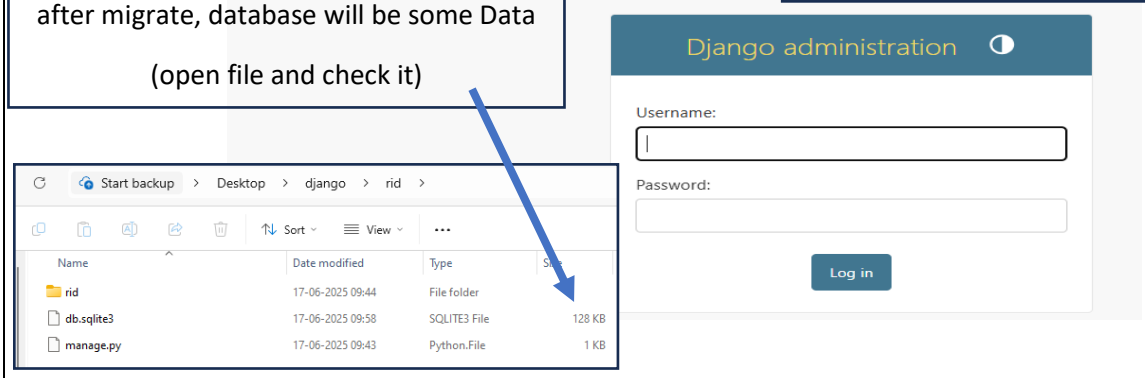
Before migrate, database have 0 KB Data

(open file and check it)



after migrate, database will be some Data

(open file and check it)

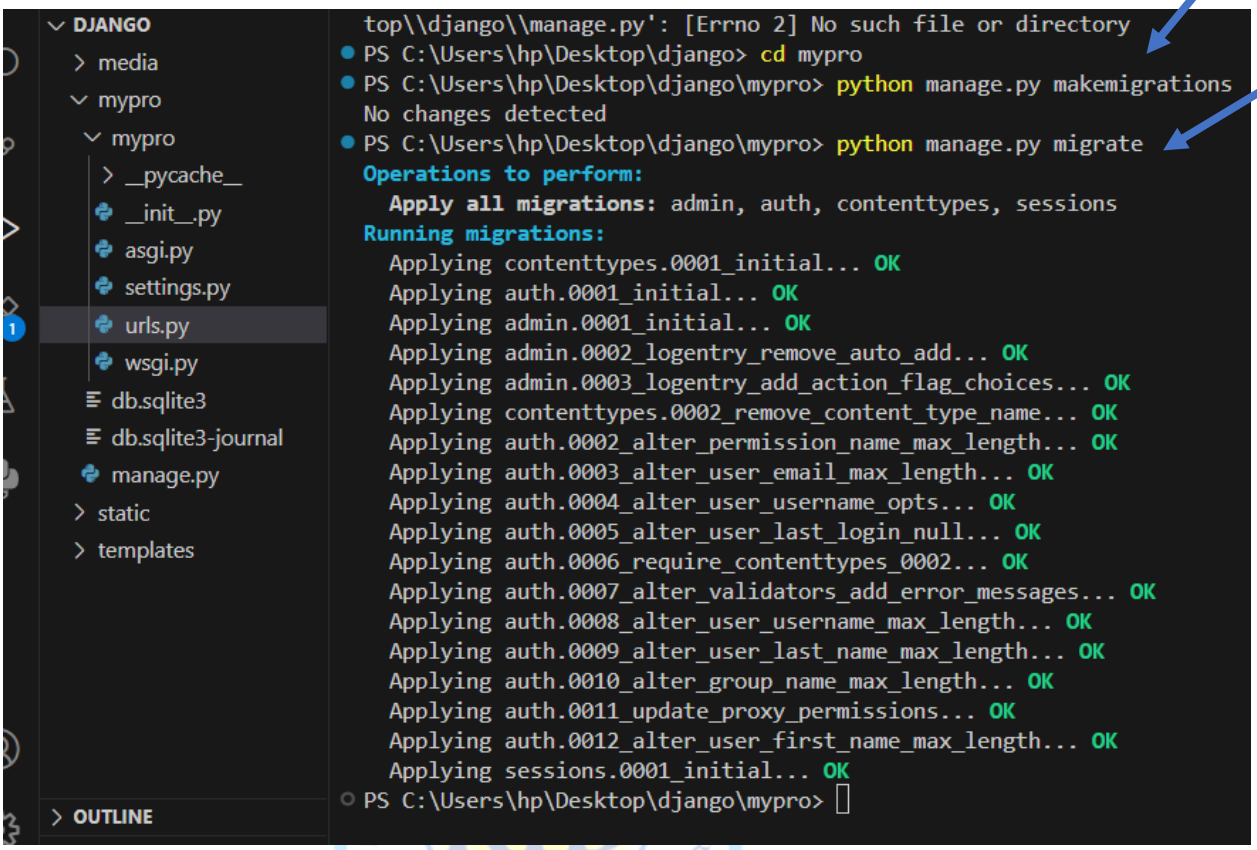


How to use the migrate command

```
PS C:\Users\hp\Desktop\django> cd mypro
PS C:\Users\hp\Desktop\django\mypro> python manage.py makemigrations
No changes detected
PS C:\Users\hp\Desktop\django\mypro> python manage.py migrate
Operations to perform:
```

Note: We use **migrate** in Django to **apply changes to the database** based on our models, ensuring the database structure matches the code.

Example:-

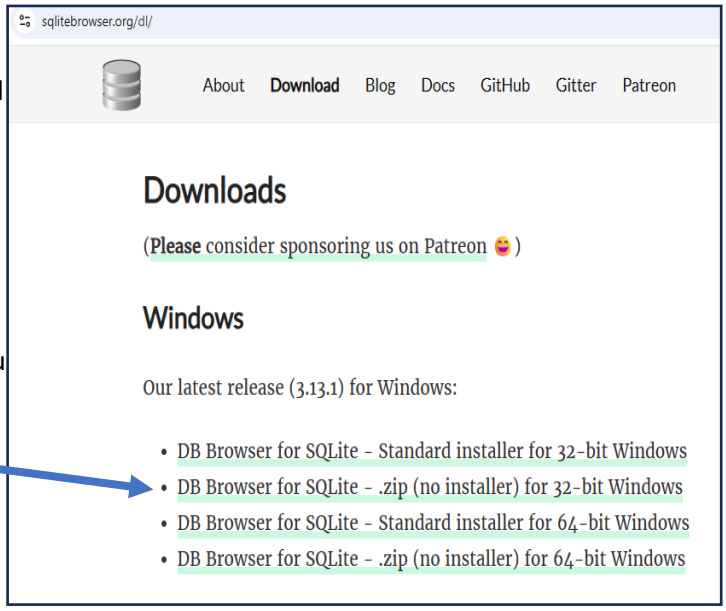


How to see SQLite database table and data

How to Download & Install DB Browser for SQLite on Window

Step-by-Step:

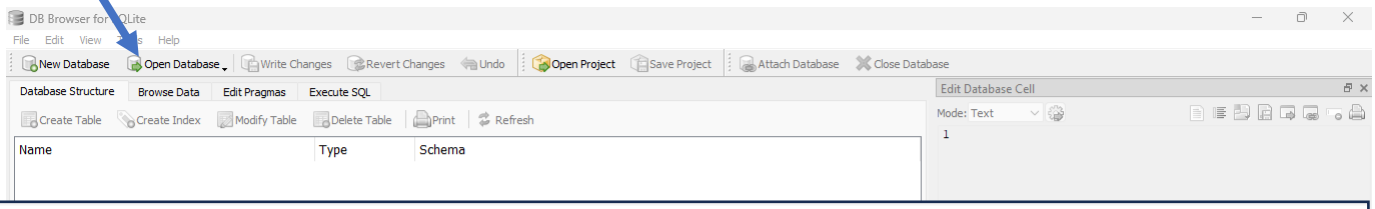
1. **Go to the official website:** https://sqlitebrowser.org/dl/
2. **Find the Windows section** Click on: **"Windows 64-bit Standard installer"**
3. **Download the .exe file** The installer file will download (e.g., DB.Browser.for.SQLite-3.x.x-win64.exe)
4. **Install the program**
   - Double-click the .exe file
   - Click **Next** and follow the instructions
   - Finish the installation
5. **Launch DB Browser**
   - Search for **"DB Browser for SQLite"** in Start Menu
   - Open the program

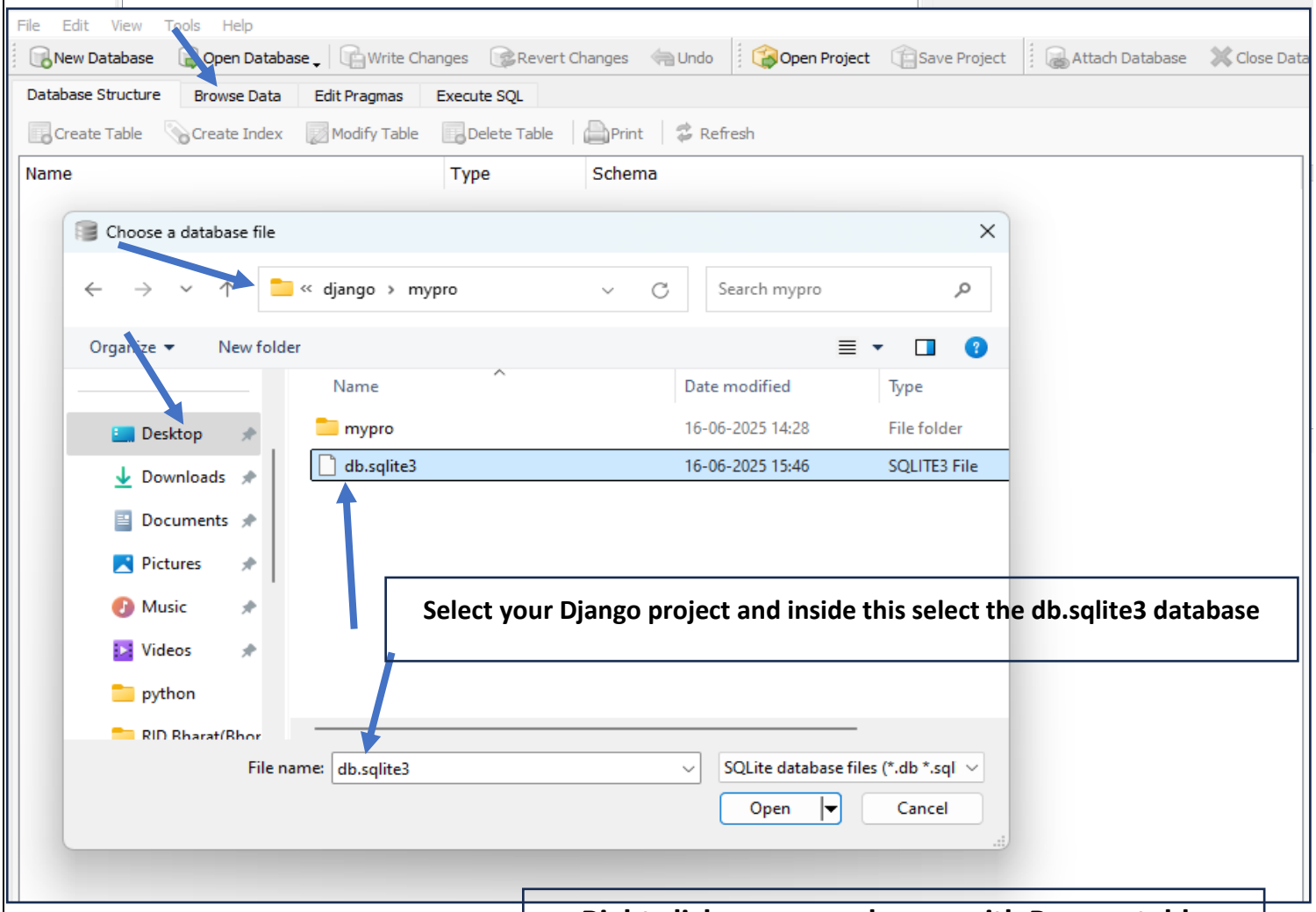


Open Your Django Database

- Click **"Open Database"**
- Select your Django db.sqlite3 file
- Click on the **"Browse Data"** tab to see your tables and

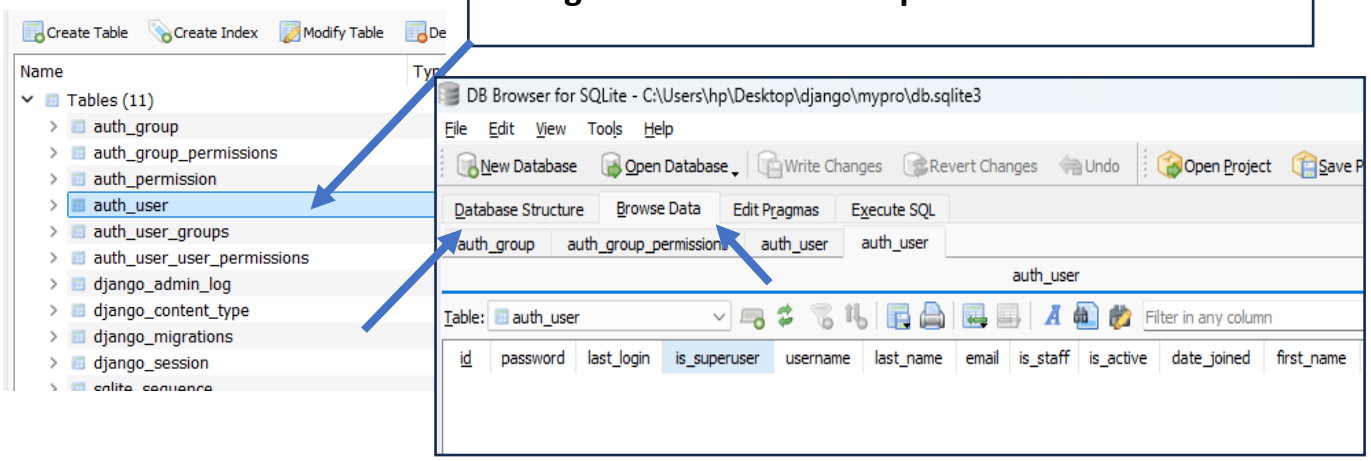
After install double click to open the DB browser database





Select your Django project and inside this select the db.sqlite3 database

Right click mouse and open with Brower table



Superuser

What is a Superuser in Django

- A **superuser** in Django is a **user with full admin access** to the Django project. They can **view, add, edit, and delete** any data in the Django Admin Panel.

Why Superuser is Used:

- To **log in** to the Django admin dashboard (/admin)
- To **manage models and data** easily through a web interface
- To **control user permissions** and site settings

How to Create a Superuser and inter in admin panel:

- Run this command in your terminal:

➢ python manage.py createsuperuser (Press Enter)

It will ask you to enter:

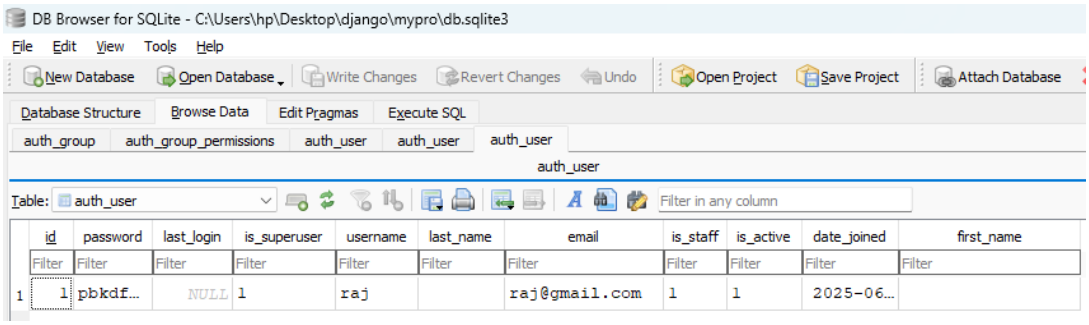
- **Username**
- **Email**
- **Password (type twice)**

After that, your superuser is ready.

```
PS C:\Users\hp\Desktop\django\mypro> python manage.py createsuperuser
Username (leave blank to use 'hp'): raj
Email address: raj@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: n
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: n
Password:
Password (again):
Superuser created successfully.
PS C:\Users\hp\Desktop\django\mypro>
```

Note: - Password example like this: ridbharat2550

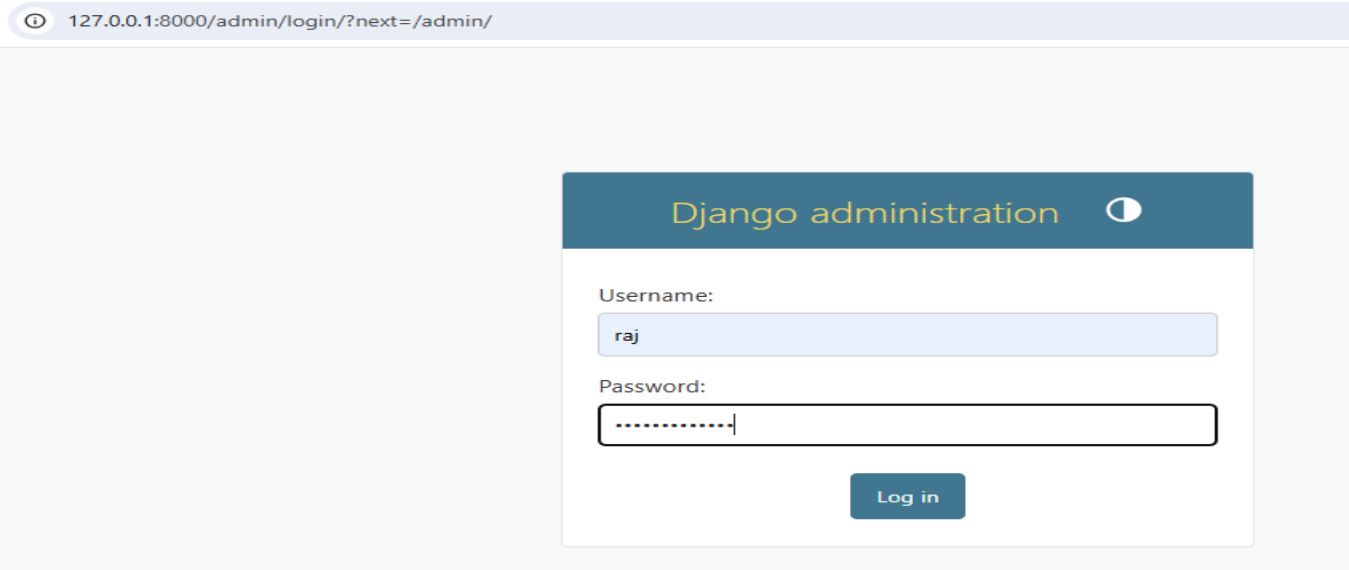
How to check the user id and password

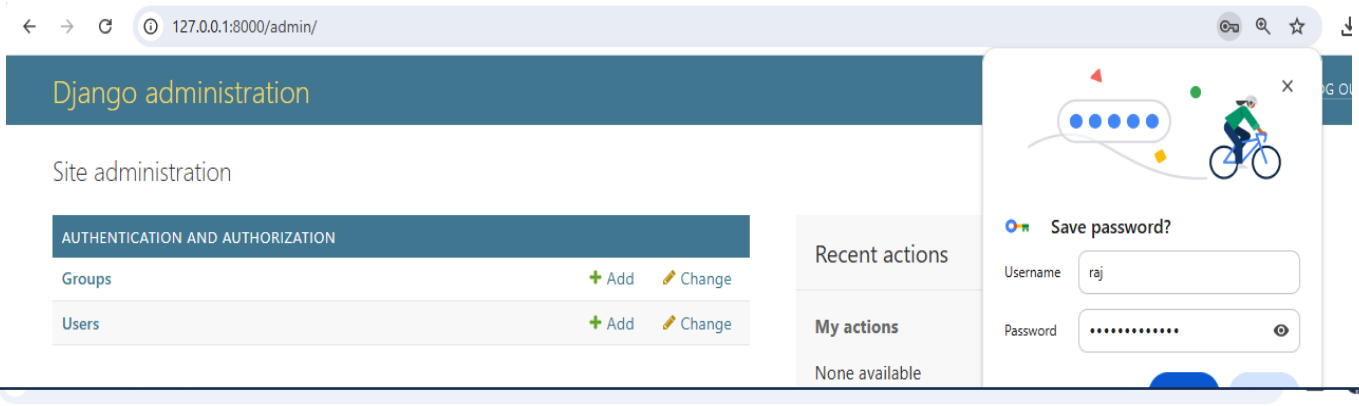


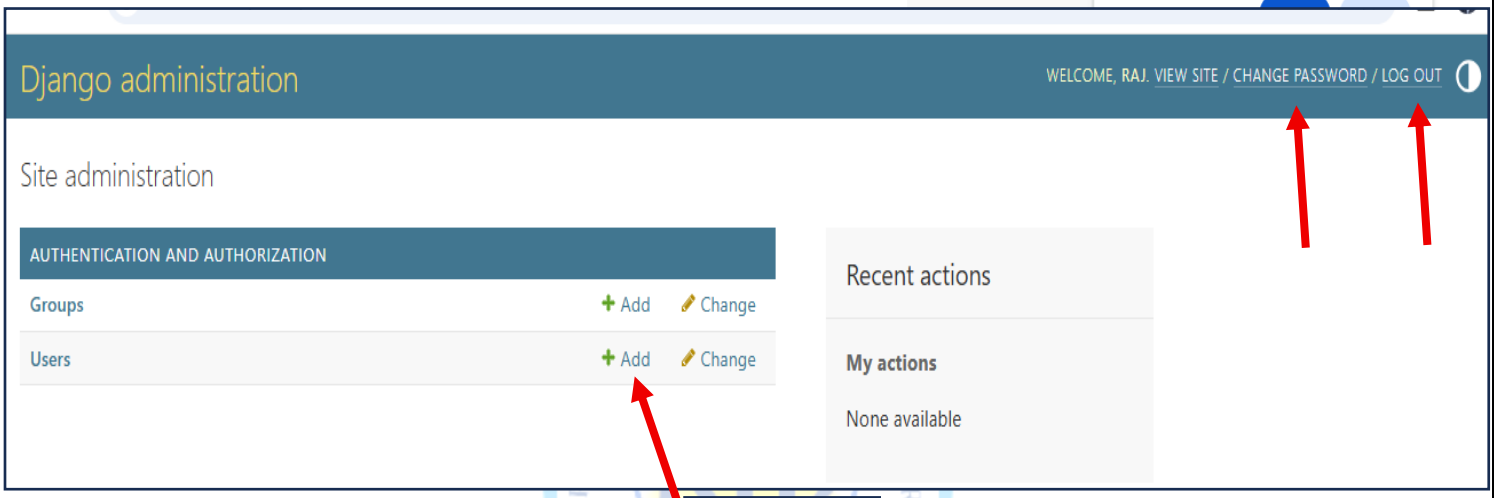
How to login in admin panel through super user

Access Admin Panel:

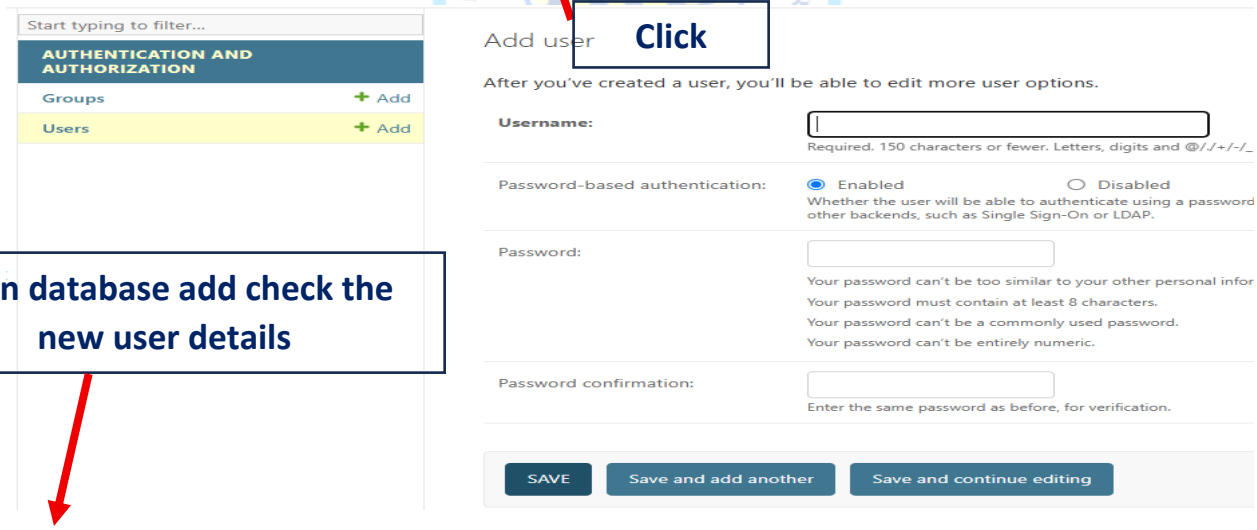
1. Start the server open the terminal and run this command
2. python manage.py runserver
3. Open browser and go to: http://127.0.0.1:8000/admin
4. Login using the superuser credentials.



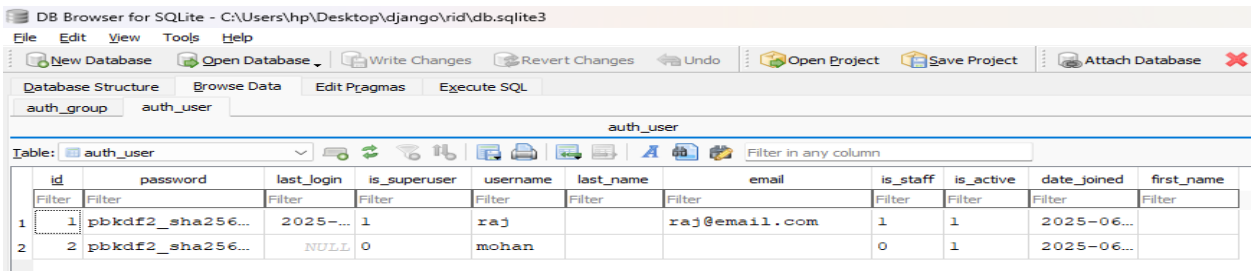




Click



Open database add check the new user details



What is View and URLS(Route)

- **URLs** are the addresses (where to go)(कहाँ जाना है) URLs Means Route
- **Views** are the workers (what to do when you get there)

  Views means (**काम करने वाले होते हैं** (वहाँ पहुँचने पर क्या करना है)

❖ **Views: -** Views are like the chefs in a restaurant. They prepare the "meal" (response) that gets served to your users.

What they do:

- Handle requests from users
- Process data (like getting info from databases)
- Decide what response to send back (usually HTML pages, but could be JSON, files, etc.)

Example: A simple view that says "Hello"

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("Hello World!")
```

❖ **URLs (Routes):-** URLs are like the restaurant's menu - they tell Django which view to use for which web address.

What they do:

- Map web addresses (URLs) to views
- Define the paths users can visit in your application
- Can capture parts of the URL as variables

Example:

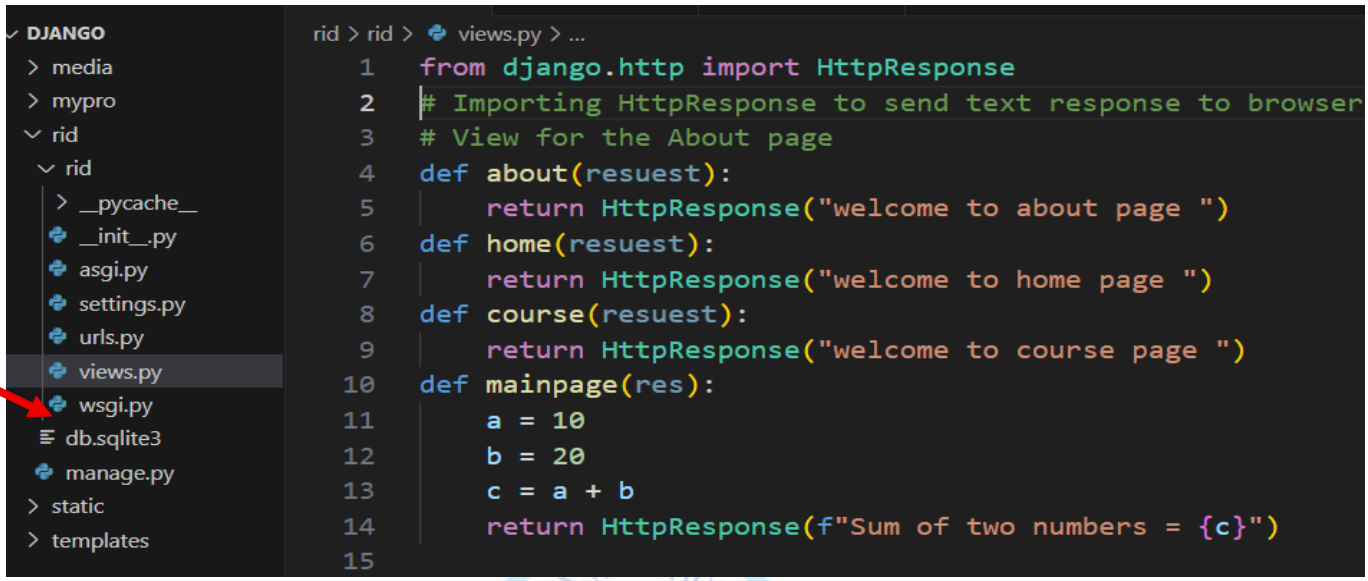
```
from django.urls import path
from . import views
urlpatterns = [
    path('hello/', views.hello),      # When someone visits /hello/, use the hello view
    path('about/', views.about),    # /about/ goes to the about view
]
```

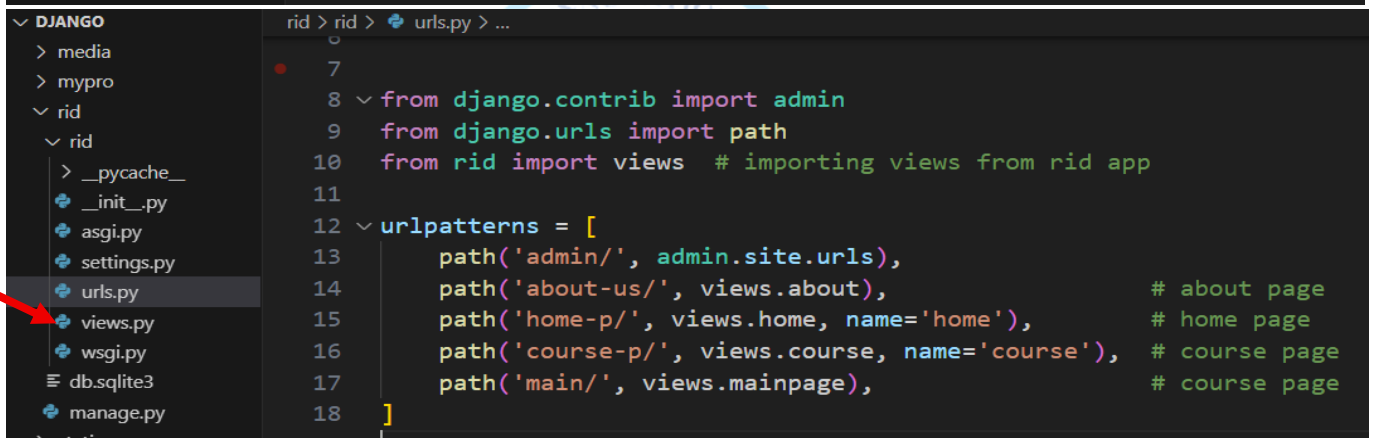
How They Work Together

1. A user visits `yoursite.com/hello/`
2. Django checks the URL patterns to find a match
3. It finds the `path('hello/', views.hello)` match
4. Django calls the `hello()` view function
5. The view returns "Hello World!" which gets sent to the user's browser

How to create URLs (Route) and views

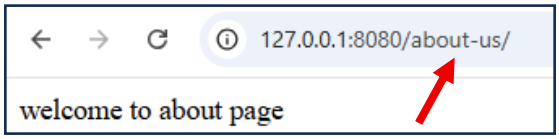
Create views file inside the main project file and write all functionality of project.

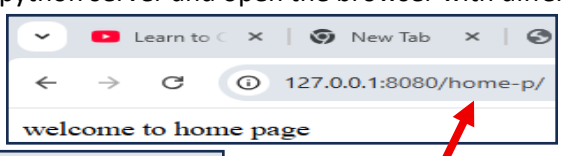


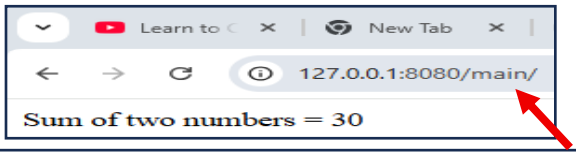


- **Wrong:**
  - return HttpResponse("sum of two number=", c)
- **Reason:** HttpResponse() takes only one string, not multiple values.
  - Correct ways (combine into one string):
- **Method 1:** f-string (Recommended) :-return HttpResponse(f"Sum of two numbers = {c}")
- **Method 2:** String concatenation :-return HttpResponse("Sum of two numbers = " + str(c))
- **Method 3:** .format() method :-return HttpResponse("Sum of two numbers = {}".format(c))

How to navigate the route the see the result(run the python server and open the browser with different file







Views.py

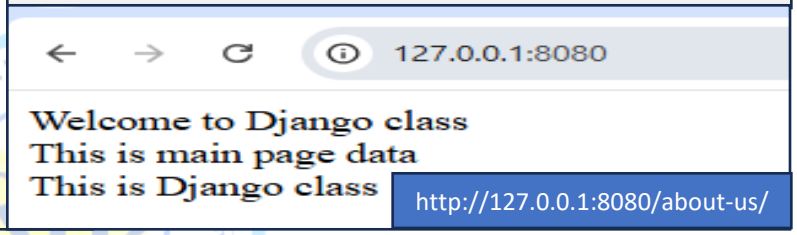
```
# Importing HttpResponse to send text response to browser
from django.http import HttpResponse
# View for the main/root page
def allone(res):
    s = "Welcome to Django class"
    s1 = "This is main page data"
    s2 = "This is Django class"
    return HttpResponse(f"{s}<br>{s1}<br>{s2}")
# using <br> for line breaks in browser
# Other views
def about(resuest):
    return HttpResponse("Welcome to about page")
def home(resuest):
    return HttpResponse("Welcome to home page")
def course(resuest):
    return HttpResponse("Welcome to course page")
def mainpage(res):
    a = 10
    b = 20
    c = a + b
    return HttpResponse(f"Sum of two numbers = {c}")
```

Urls.py

```
from django.contrib import admin
from django.urls import path
from rid import views
# Importing views from the 'rid' app
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.allone),                    # Root URL → allone view
    path('about-us/', views.about),     # About page
    path('home-p/', views.home, name='home'),   # Home page
    path('course-p/', views.course),                # Course page
    path('main/', views.mainpage),
]
```

Note:

- Each path() maps a URL to its corresponding view.
- name='...' is optional but helpful for reverse URL lookups in templates.



What is a Dynamic Route in Django?

- A **dynamic route** in Django means a URL that can change based on user input or data. Instead of writing fixed URLs, we can use **variables** in the URL path.

Example: Static URL (Fixed):

path('course/', views.course)

This only matches:

/course/

Each time, it passes the value (e.g., 1, 100, 999) to the view as a parameter.

Dynamic URL (Changes based on value):

path('course/<int:id>/', views.showCourse)

This matches:

/course/1/

/course/100/

/course/999/

How to Create a Dynamic Route

Step 1: Add route in urls.py

path('course/<int:id>/', views.showCourse)

- int → type of the variable (you can use str, slug, etc.)
- id → variable name sent to the view function

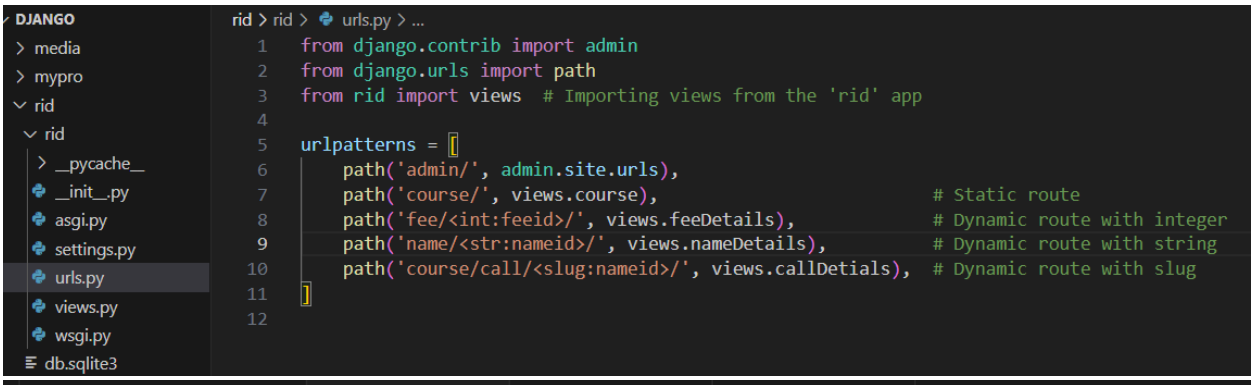
Step 2: Create view in views.py

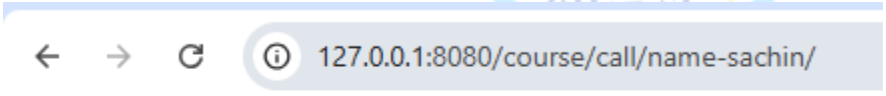
```
def showCourse(request, id):
        return HttpResponse(f"Course ID is {id}")
```

Common Dynamic Types

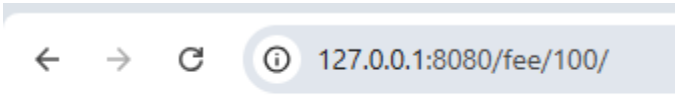
Type	Description	Example URL
<int:id>	Integer only	/course/10/
<str:name>	Any string (no slashes)	/user/Tanmay/
<slug:slug>	Letters, numbers, hyphens	/post/python-basics/

Example: -How to Creating the dynamic route

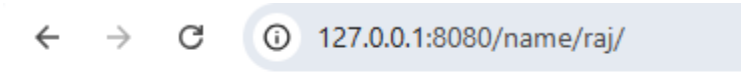




Call Details Slug: name-sachin



Fee ID: 100



Name ID: raj

Final Exampleurls.py path('student/<int:roll>/', views.showStudent)
views.py

```
def showStudent(request, roll):
    return HttpResponse(f"Student Roll Number is {roll}")
```

URL:
http://127.0.0.1:8000/student/55/
Output: Student Roll Number is 55

How to create class-based views in Django

Step 1: Create or open a Django app

- Make sure your app is already created and added to INSTALLED_APPS in settings.py.
- python manage.py startapp myapp

Step 2: Import required classes in views.py

```
from django.http import HttpResponse
from django.views import View
```

Step 3: Create a class-based view

```
class HomeView(View):
    def get(self, request):
        return HttpResponse("Welcome to Class-Based View")
```

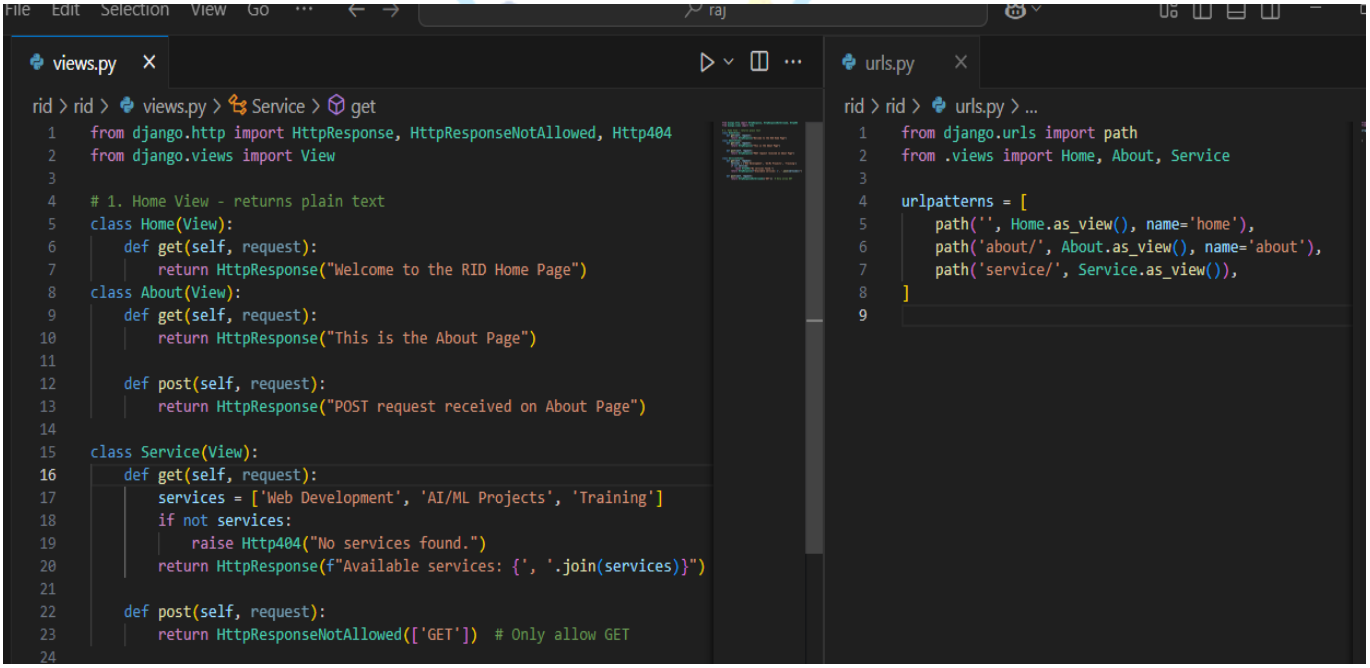
Step 4: Add the view to urls.py

- In your **project's or app's urls.py**, connect the view:

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

urlpatterns = [
    path('', HomeView.as_view(), name='home'),  # Use .as_view()
]
```

Step 5: Run the server and visit the URL

- python manage.py runserver
- Open in browser:
  http://127.0.0.1:8000/
  Output: Welcome to Class-Based View



views.py

```
from django.http import HttpResponse, HttpResponseNotAllowed, Http404
from django.views import View

# 1. Home View - returns plain text
class Home(View):
    def get(self, request):
        return HttpResponse("Welcome to the RID Home Page")
class About(View):
    def get(self, request):
        return HttpResponse("This is the About Page")

    def post(self, request):
        return HttpResponse("POST request received on About Page")

class Service(View):
    def get(self, request):
        services = ['Web Development', 'AI/ML Projects', 'Training']
        if not services:
            raise Http404("No services found.")
        return HttpResponse(f"Available services: {', '.join(services)}")

    def post(self, request):
        return HttpResponseNotAllowed(['GET'])  # Only allow GET
```

urls.py

```
from django.urls import path
from .views import Home, About, Service

urlpatterns = [
    path('', Home.as_view(), name='home'),
    path('about/', About.as_view(), name='about'),
    path('service/', Service.as_view()),
]
```

How to render the html file in Django

Step 1: Create a Template Folder

- In your Django app folder (e.g., rid/), create a folder named:
- templates
- Inside templates/, create another folder with the **same name as your app** (rid/), like this:
- rid/

```
├── views.py
├── templates/
│   └── rid/
│       └── index.html
```

Step 2: Write an HTML File

- Create index.html inside templates/rid/:

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Page</title>
</head>
<body>
  <h1>Hello, this is a rendered HTML page!</h1>
</body>
</html>
```

Step 3: Update views.py to Render the HTML

- Import render and create a view:

```
from django.shortcuts import render
def showPage(request):
return render(request, 'rid/example.html')
```

Step 4: Add Route in urls.py

```
from django.urls import path
from rid import views
urlpatterns = [
  path('page/', views.showPage),  # Renders the example.html page
]
```

Step 5: Make Sure Template Directory is Configured

- In settings.py, check that this line is present in TEMPLATES:
- 'APP_DIRS': True,
- It's usually already enabled by default in Django projects.

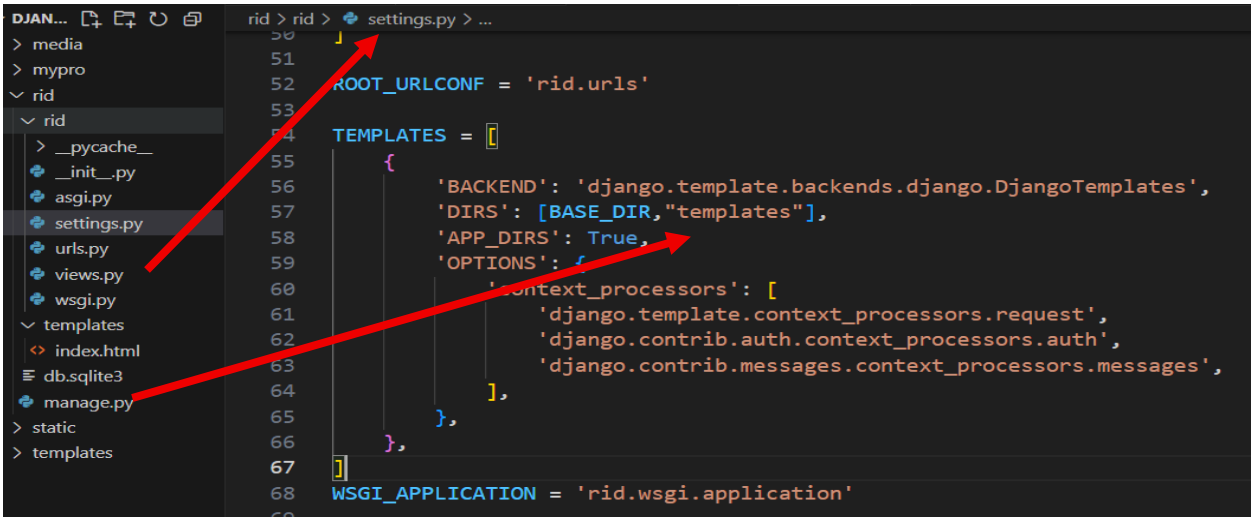
Now Open This in Browser:

http://127.0.0.1:8000/page/

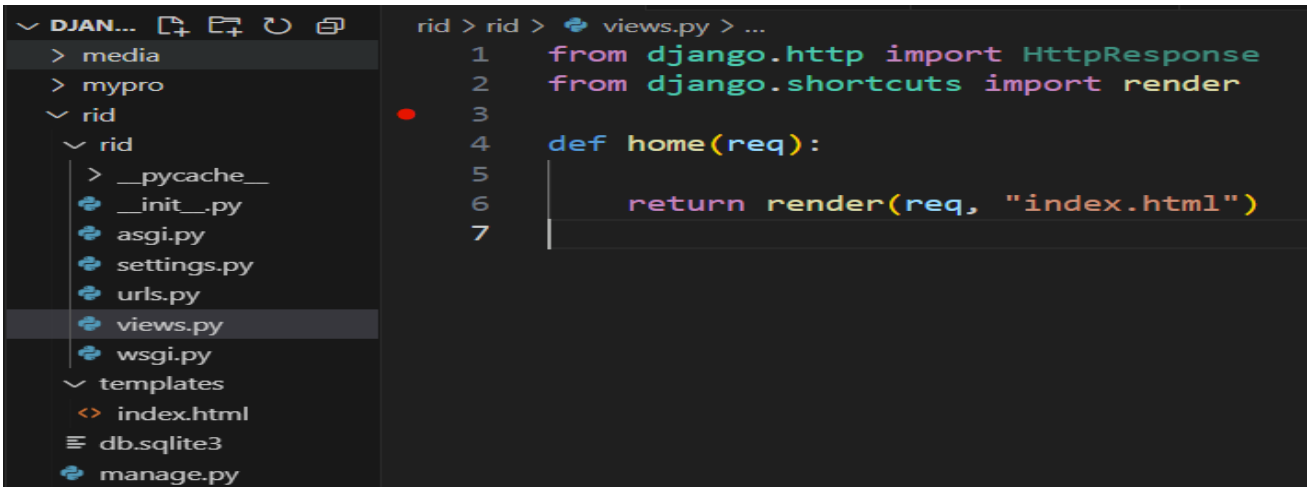
You will see:

Hello, this is a rendered HTML page!

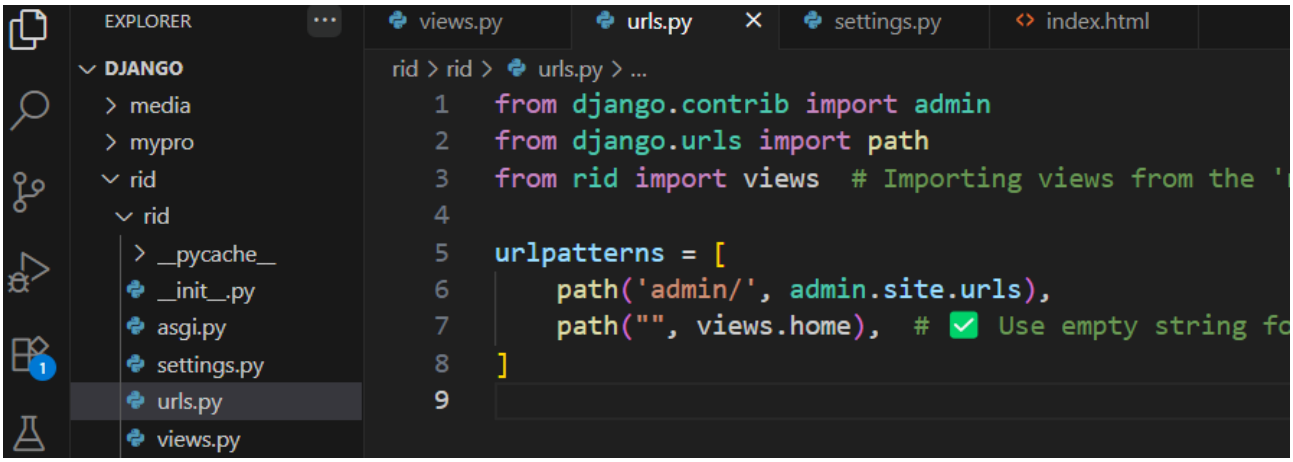
Example: - setting.py



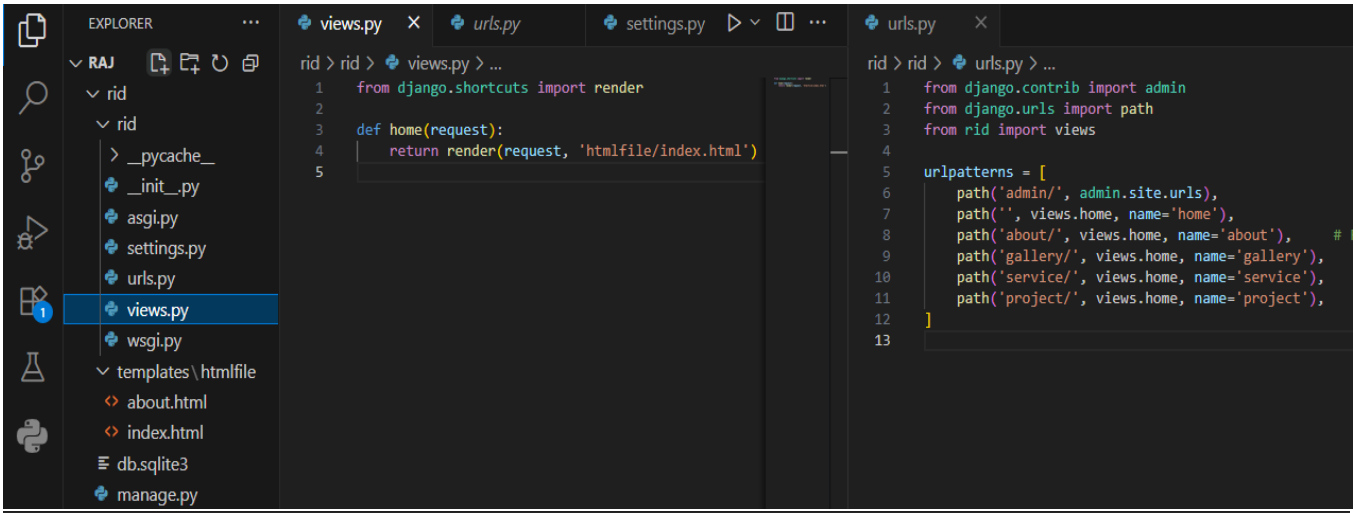
views.py

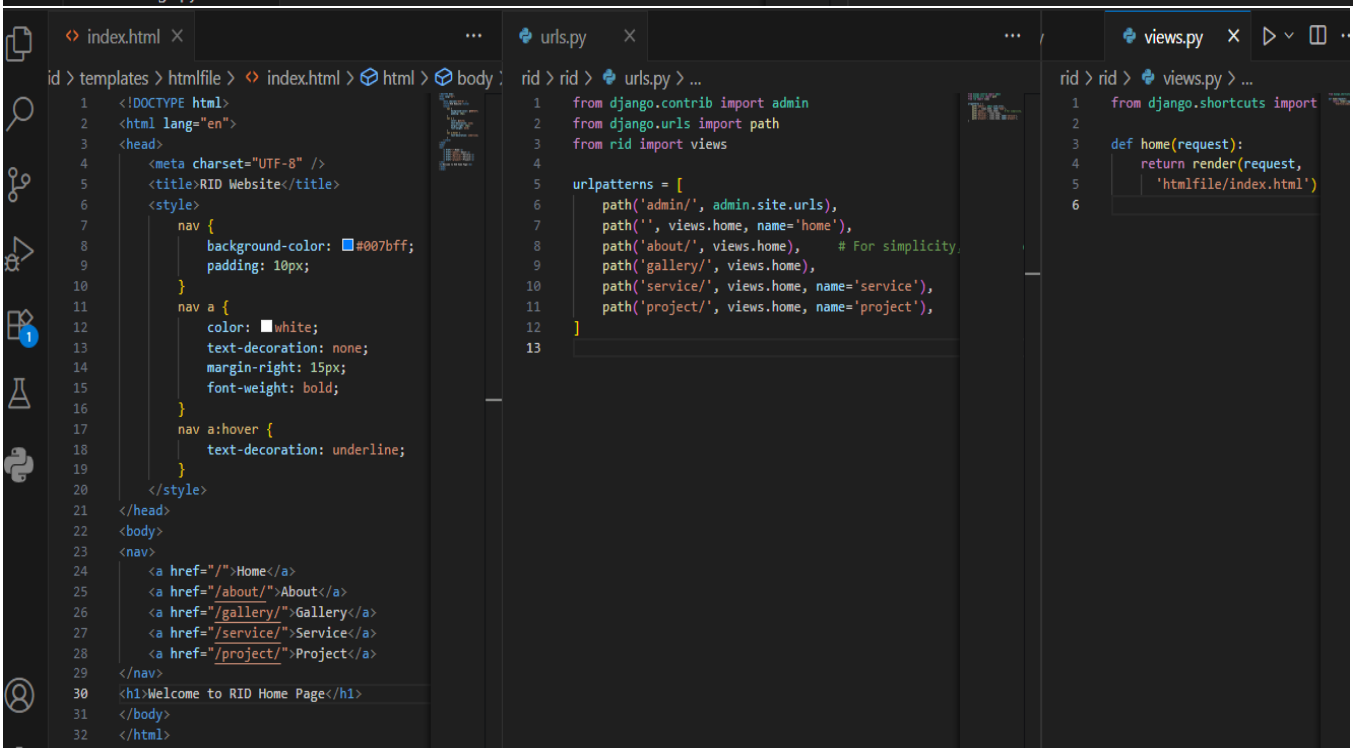


Urls.py

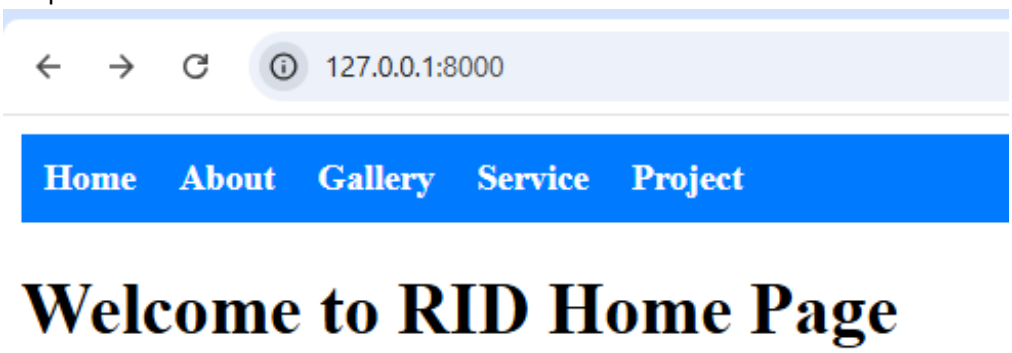


Example-2:

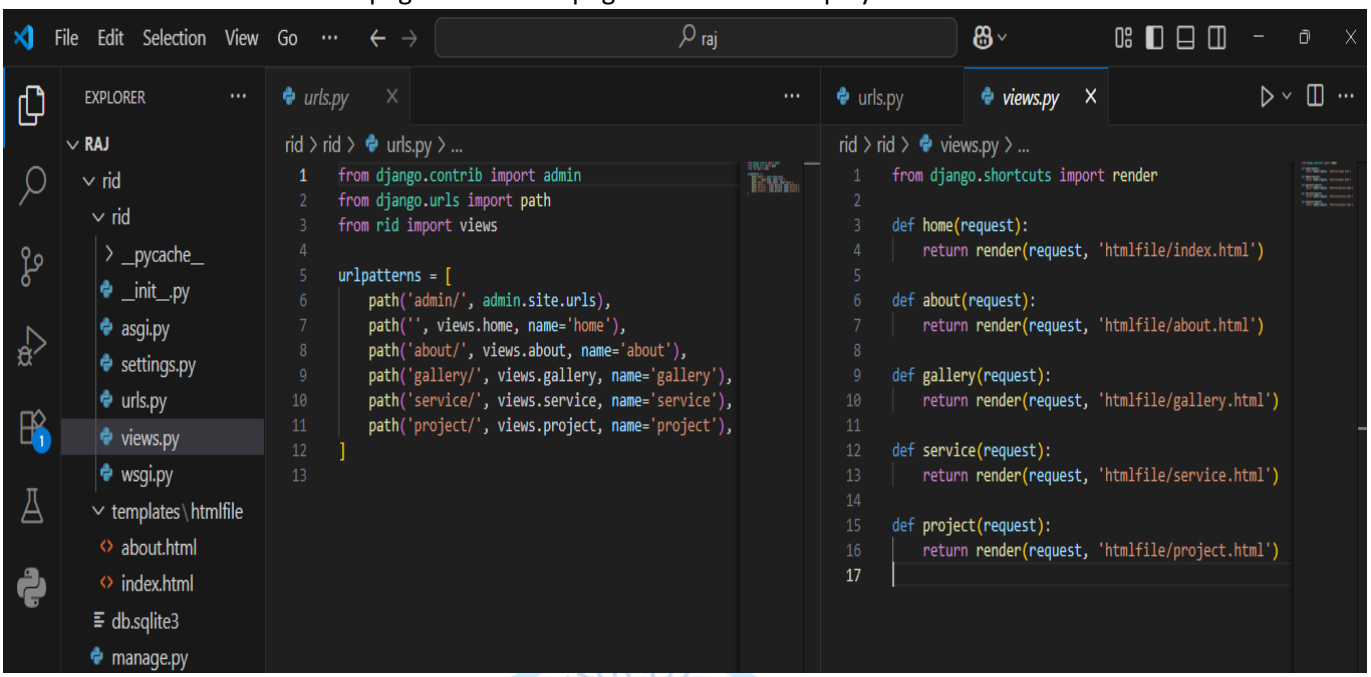




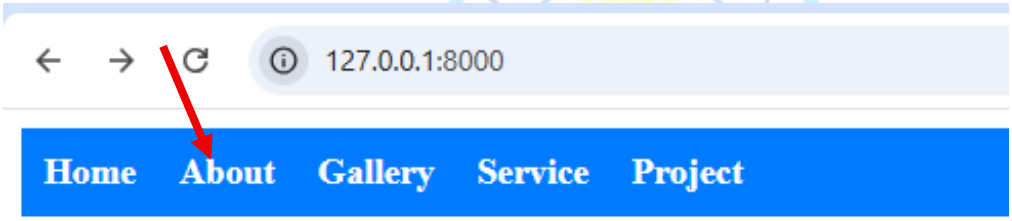
Output:



When user will click the about page then about page html file will display

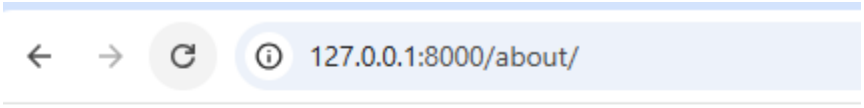


Output



Welcome to RID Home Page

After about page click then we will get the result below



About RID Organization

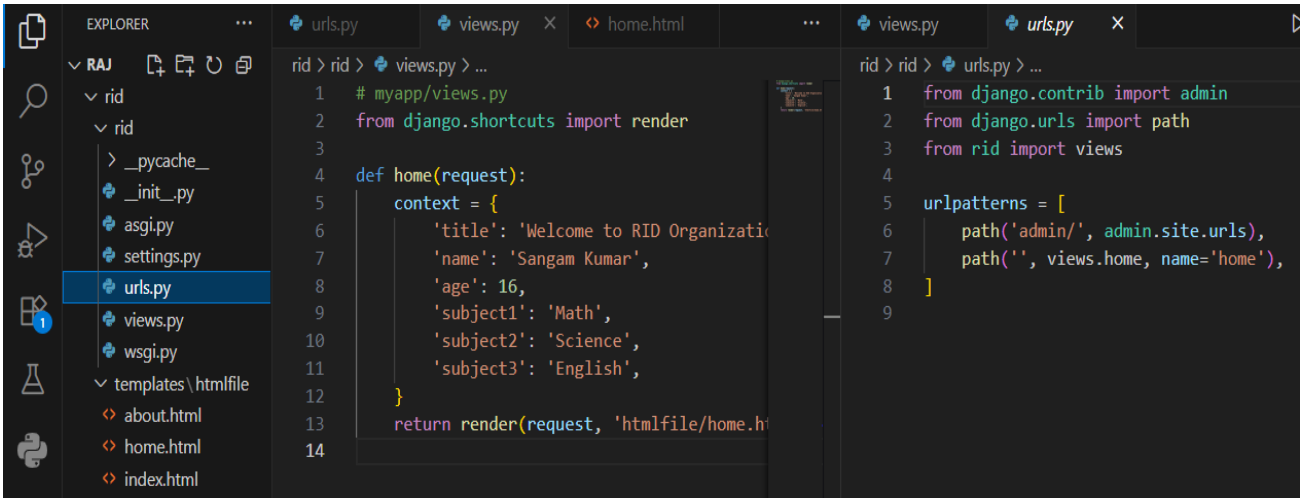
This is the about page content.

PASS THE DATA

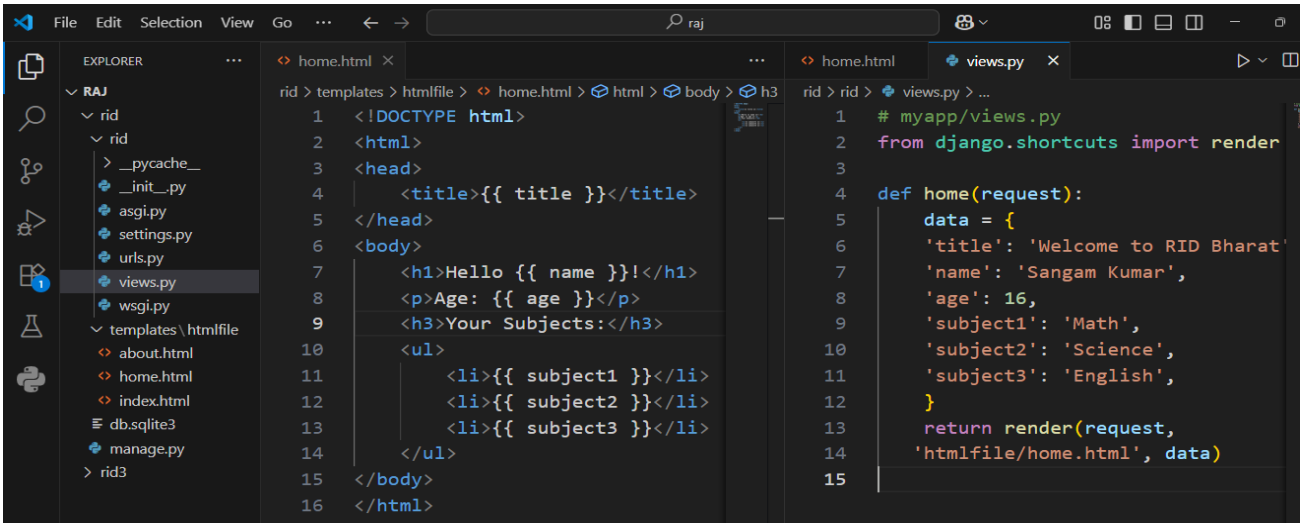
How to pass the Data from the Djnago to Templates means html file

1. **Imported render function**:- Used to send data and render HTML page.
2. from **django.shortcuts** import render
3. **Created a view function named home**: - Handles the request when user visits the page.
4. **Used a dictionary data**: - Stores all variables you want to pass to template (title, name, age, subjects).
5. **Returned render ()** Passes:
   - request
   - HTML file path: 'htmlfile/home.html'
   - context data: data
6. **return render**(request, 'htmlfile/home.html', data)
7. **htmlfile/home.html is the path inside the templates/ folder**

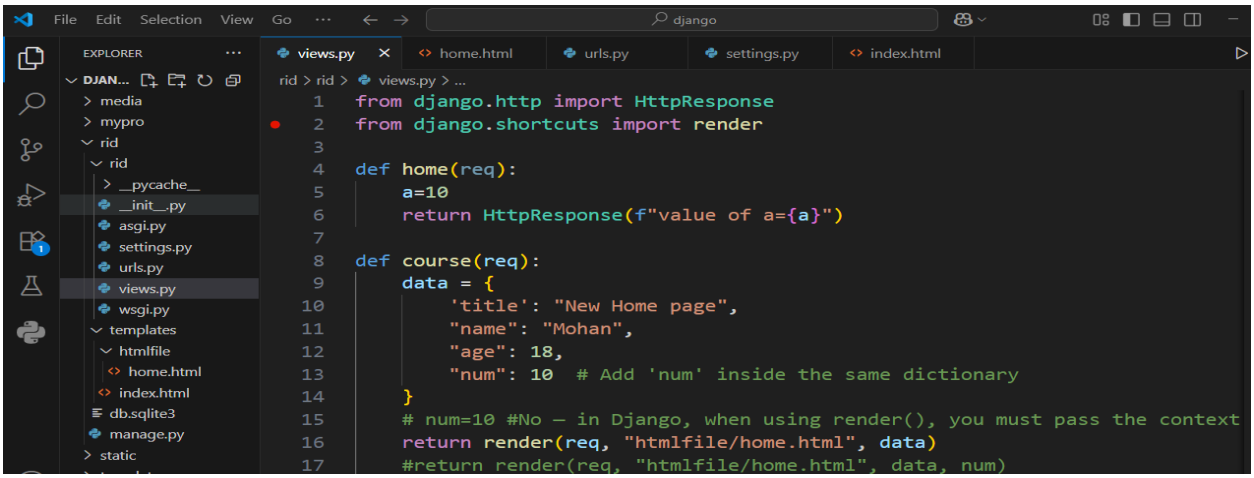
Example-1:- urls.py



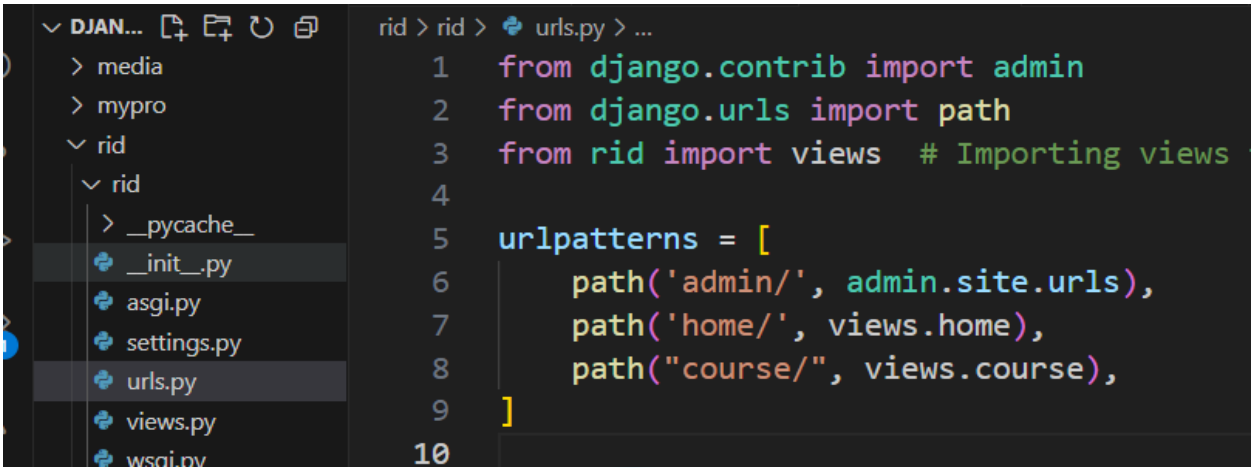
Views.py & index.html



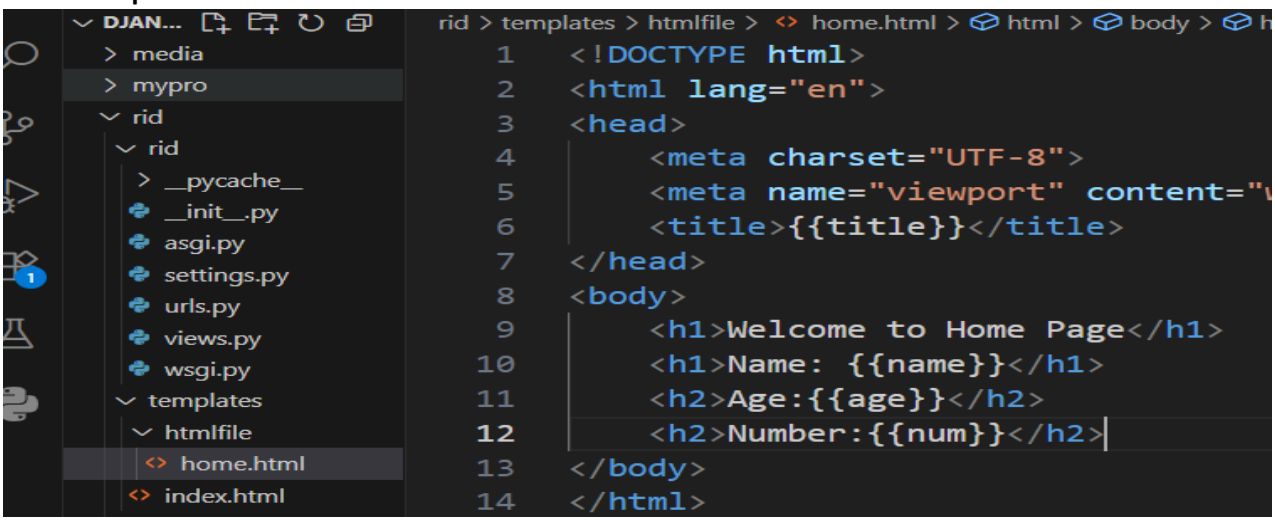
Example-2:- Views.py



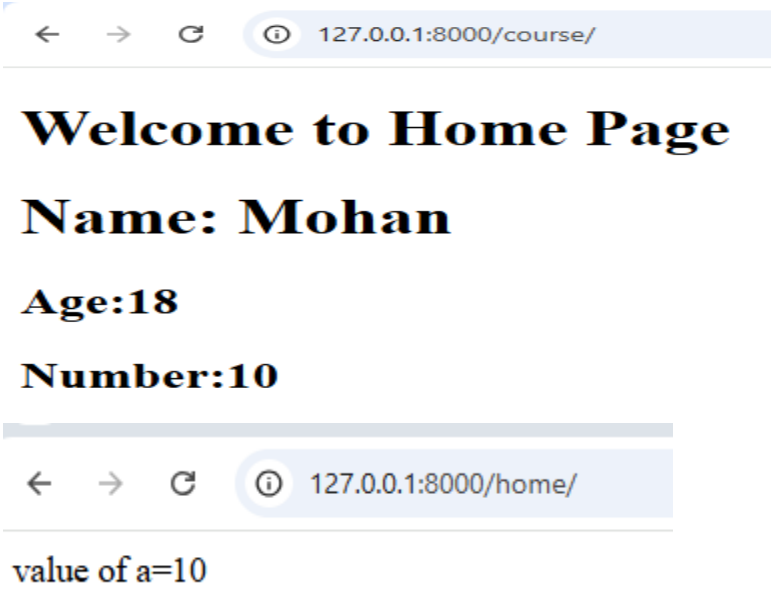
Urls.py



Inside templates: - Home.html



Output:

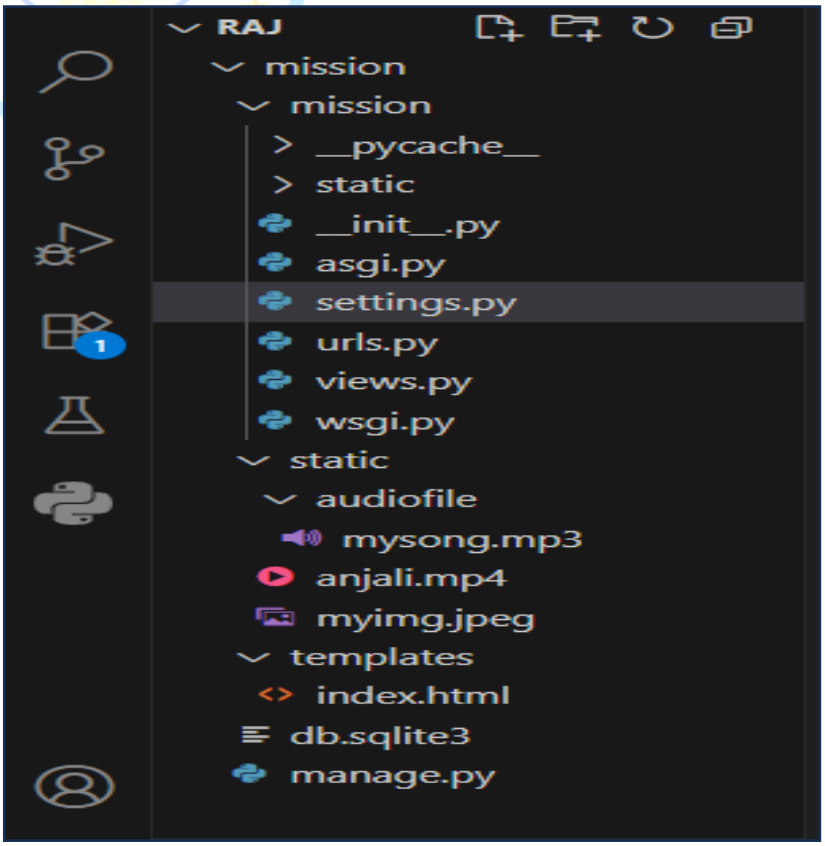


Example-3:

How to pass text, image audio and video

Folder Structure (for example)

```
myproject/
├── mission/
|   ├── templates/
|   |   └── index.html
|   ├── static/
|   |   └── mission/
|   |       ├── sangam.jpg
|   |       ├── music.mp3
|   |       └── video.mp4
|   ├── views.py
|   └── urls.py
├── myproject/
|   └── settings.py
```



Step By step Guide for display the text, image, audio and video

❖ **Main Requirements**

- To display **text, image, audio, video**:
  - **Use Django** views.py to pass data
  - **Store media** files in static folder that is create by user in project folder. (not in main folder)
  - **Write html** file in templates and Use {% static %} in index.html
  - **Configure settings.py properly**

Step-1: setup Project Structure

```
project/
├── mission/          ← Django app
├── static/
│   ├── myimg.jpeg    ← image
│   ├── anjali.mp4    ← video
│   └── audiofile/
│       └── mysong.mp3  ← audio
├── templates/
│   └── index.html    ← HTML template
```

2. settings.py

➢ **File:** mission/settings.py

➢ **Add:**

```
import os
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
INSTALLED_APPS = [
    ...
    'mission',  # your app name
]
```

3. views.py

➢ **File:** mission/views.py

➢ **Create view:**

```
from django.shortcuts import render
def info(request):
    data = {
        "Title": "Media Demo",
        "Name": "Sangam",
        "ImagePath": "myimg.jpeg",
        "AudioPath": "audiofile/mysong.mp3",
        "VideoPath": "anjali.mp4"
    }
    return render(request, "index.html", data)
```

4. urls.py

➢ **File:** mission/urls.py

➢ **Add:**

```
from django.contrib import admin
from django.urls import path
from mission import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.info),
]
```

5. index.html

➢ File: templates/index.html

➢ HTML:

```
{% load static %}
<!DOCTYPE html>
<html>
<head><title>{{ Title }}</title></head>
<body>
  <h1>{{ Title }}</h1>
  <p>Name: {{ Name }}</p>
  <h3>Image</h3>
  <img src="{% static ImagePath %}" width="300">
  <h3>Audio</h3>
  <audio controls>
<source src="{% static AudioPath %}" type="audio/mpeg">
  </audio>
  <h3>Video</h3>
  <video width="400" controls>
 <source src="{% static VideoPath %}" type="video/mp4">
  </video>
</body>
</html>
```

views.py and urls.py

```
# mission/views.py

from django.shortcuts import render

def info(request):
    data = {
        "Title": "Media Demo",
        "Name": "Sangam Kumar",
        "ImagePath": "myimg.jpeg",
        "AudioPath": "audiofile/mysong.mp3",
        "VideoPath": "anjali.mp4",
    }
    return render(request, "index.html", data)
```

```
# myproject/urls.py

from django.contrib import admin
from django.urls import path
from mission import views

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

setting.py and index.html

```
import os  # Required for STATICFILES_DIRS
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 's
BASE_DIR = Path(__file__).resolve().parent.parent

# SECURITY WARNING: keep the secret key used in productio
SECRET_KEY = 'django-insecure-q7u76ri31gye@2jnkj_4-b!2=w5

# SECURITY WARNING: don't run with debug turned on in pro
DEBUG = True

ALLOWED_HOSTS = []
# Static files (CSS, JavaScript, Images, Audio, Video)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mission',  # Your app
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware
    'django.middleware.common.CommonMiddleware',
```

```
<!-- templates/index.html -->
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ Title }}</title>
</head>
<body>
    <h1>{{ Title }}</h1>
    <p><strong>Name:</strong> {{ Name }}</p>

    <!-- Image Section -->
    <h3>Image:</h3>
    <img src="{% static ImagePath %}" alt="Student Image"

    <!-- Audio Section -->
    <h3>Audio:</h3>
    <audio controls>
        <source src="{% static AudioPath %}" type="audio/
        Your browser does not support the audio tag.
    </audio>

    <!-- Video Section -->
    <h3>Video:</h3>
    <video width="400" controls>
        <source src="{% static VideoPath %}" type="video/
        Your browser does not support the video tag.
    </video>
</body>
</html>
```

Key Points

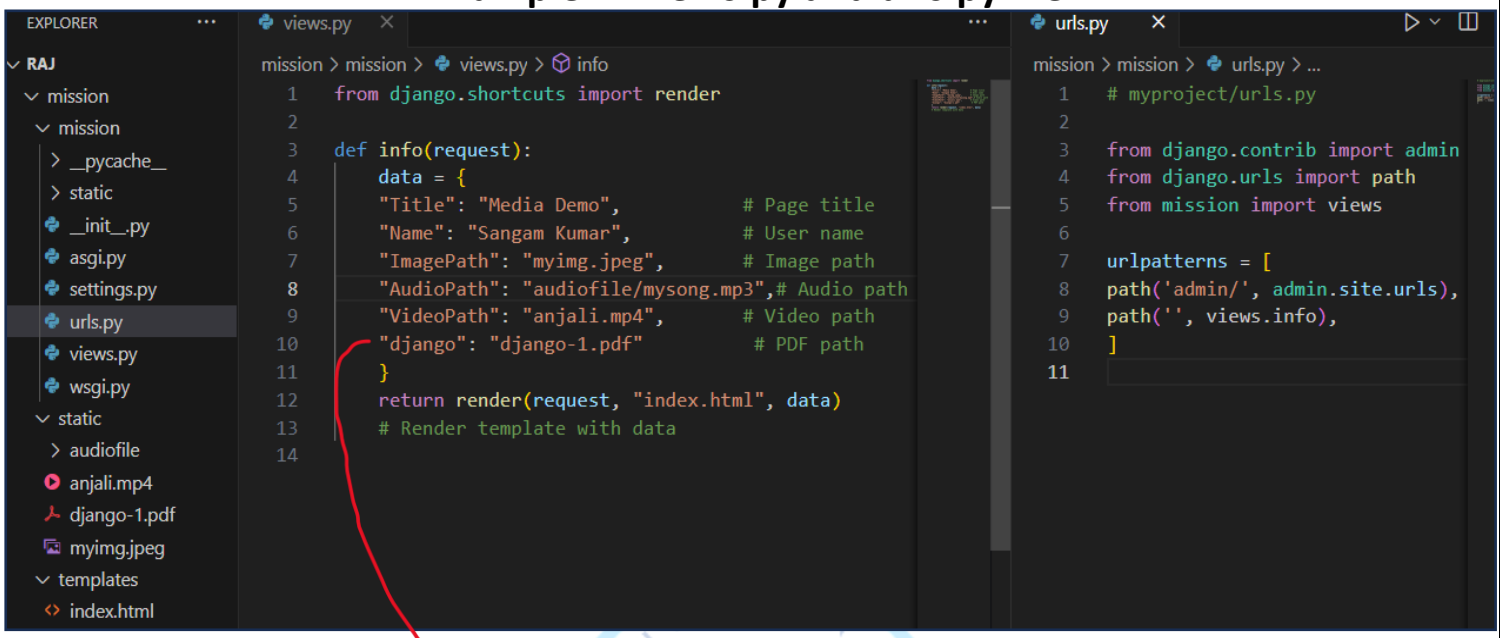
- **Text**: Use {{ Name }} in HTML.
- **Image**:
  - Path: "ImagePath": "myimg.jpeg"
  - HTML: <img src="{% static ImagePath %}">
- **Audio**:
  - "AudioPath": "audiofile/mysong.mp3"
  - <audio controls><source src="{% static AudioPath %}"> </audio>
- **Video**:
  - "VideoPath": "anjali.mp4"
  - <video controls><source src="{% static VideoPath %}"> </video>

Files & What to Do

- **views.py:** Pass data using a dictionary.
- **urls.py:** Map view to URL.
- **settings.py:**
  - Add 'mission' to INSTALLED_APPS

How to Pass data like (text, image, audio, video, and pdf) from Django to html

Example-1: views.py and urls.py file



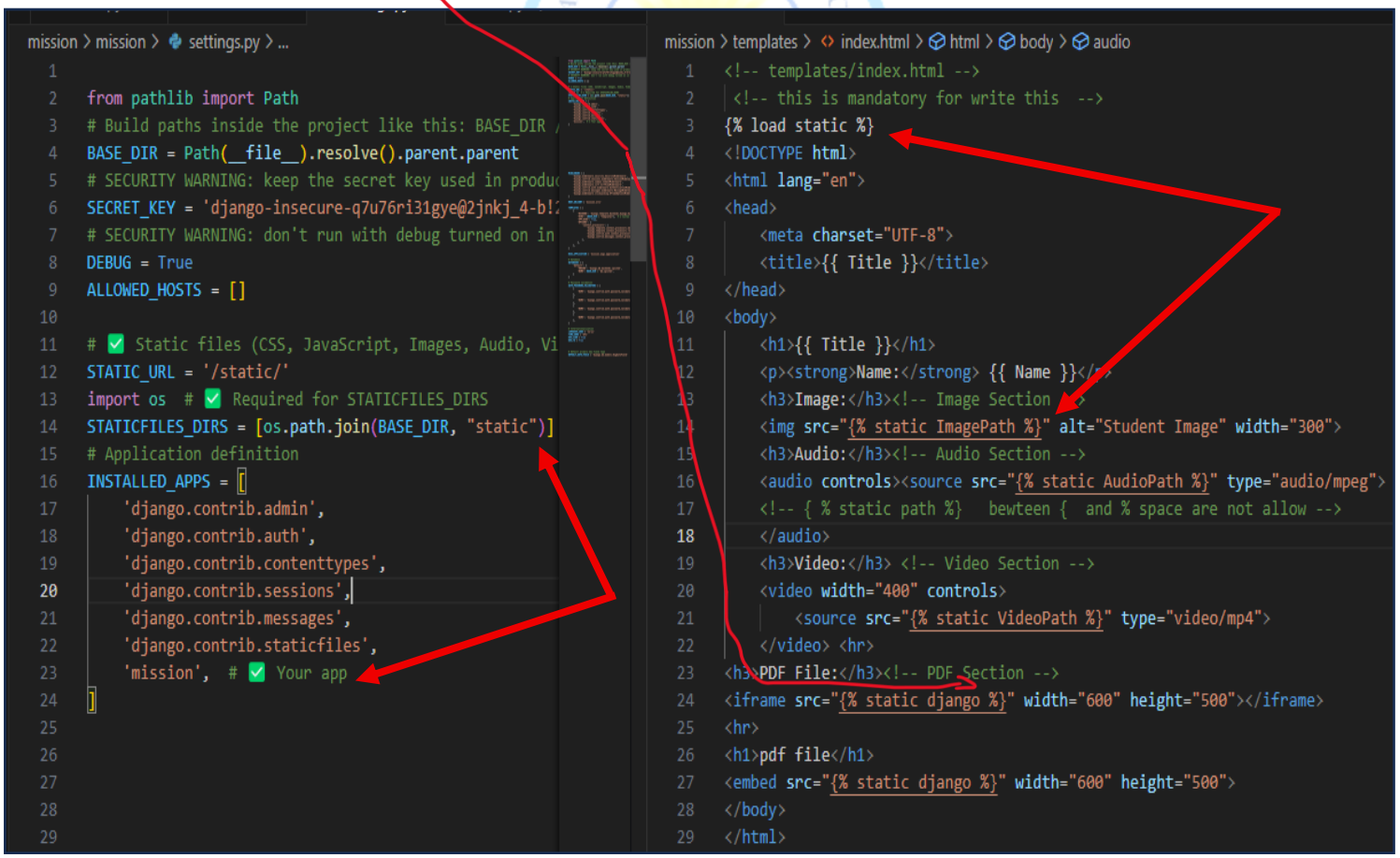
views.py

```
from django.shortcuts import render

def info(request):
    data = {
    "Title": "Media Demo",            # Page title
    "Name": "Sangam Kumar",           # User name
    "ImagePath": "myimg.jpeg",        # Image path
    "AudioPath": "audiofile/mysong.mp3",# Audio path
    "VideoPath": "anjali.mp4",        # Video path
    "django": "django-1.pdf"          # PDF path
    }
    return render(request, "index.html", data)
    # Render template with data
```

urls.py

```
# myproject/urls.py

from django.contrib import admin
from django.urls import path
from mission import views

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

setting.py file and index.html file



settings.py

```
from pathlib import Path
# Build paths inside the project like this: BASE_DIR /
BASE_DIR = Path(__file__).resolve().parent.parent
# SECURITY WARNING: keep the secret key used in produc
SECRET_KEY = 'django-insecure-q7u76ri31gye@2jnkj_4-b!7
# SECURITY WARNING: don't run with debug turned on in
DEBUG = True
ALLOWED_HOSTS = []

# ✅ Static files (CSS, JavaScript, Images, Audio, Vi
STATIC_URL = '/static/'
import os  # ✅ Required for STATICFILES_DIRS
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mission',  # ✅ Your app
]
```

index.html

```
<!-- templates/index.html -->
 <!-- this is mandatory for write this  -->
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ Title }}</title>
</head>
<body>
    <h1>{{ Title }}</h1>
    <p><strong>Name:</strong> {{ Name }}</p>
    <h3>Image:</h3><!-- Image Section -->
    <img src="{% static ImagePath %}" alt="Student Image" width="300">
    <h3>Audio:</h3><!-- Audio Section -->
    <audio controls><source src="{% static AudioPath %}" type="audio/mpeg">
    <!-- { % static path %}   bewteen {  and % space are not allow -->
    </audio>
    <h3>Video:</h3> <!-- Video Section -->
    <video width="400" controls>
        <source src="{% static VideoPath %}" type="video/mp4">
    </video> <hr>
<h3>PDF File:</h3><!-- PDF Section -->
<iframe src="{% static django %}" width="600" height="500"></iframe>
<hr>
<h1>pdf file</h1>
<embed src="{% static django %}" width="600" height="500">
</body>
</html>
```


Media Demo
Image:
Audio:
Video:
PDF File:

HTTP Request Methods in Django

- **HTTP Request Methods** define the type of action a user/browser wants to perform on the server. The most common methods are

Method	Action
GET	Retrieve/display
POST	Submit/create data
PUT	Replace data
PATCH	Partial update
DELETE	Remove data

Note: - GET and POST are most commonly used in Django views.

1. Get Method:

- GET method in Django is used to **retrieve data from the server**. It sends a request to the server with parameters in the **URL**, and the server responds with data.

❖ **Syntax:**
```
if request.method == 'GET':
    data = request.GET.get('key')
```

- ➢ **request.GET** is a dictionary-like object containing all **query parameters** passed via the URL.
- ➢ **get('key')** is used to extract the value of a specific key.

◆ **Example: views.py**
```
from django.shortcuts import render
def search(request):
    if request.method == 'GET':
        query = request.GET.get('q')  # getting data from URL
        return render(request, 'search.html', {'query': query})
```

- **Example URL:** http://127.0.0.1:8000/search?q=Django
- **search.html: -** <h2>Search Results for: {{query }}</h2>

❖ Use Cases:

- Displaying forms
- Searching content
- Viewing data
- Passing simple values via URL

❖ Advantages:

1. Easy to bookmark and share URLs.
2. Simple to use and debug (visible in address bar).
3. Can be cached by the browser.
4. Ideal for **read-only** operations.

❖ Disadvantages:

1. Limited data size (URL length limit).
2. Less secure (data visible in URL).
3. Cannot be used for sensitive data (e.g., passwords).
4. Modifying server data with GET is unsafe.

❖ Limitations:

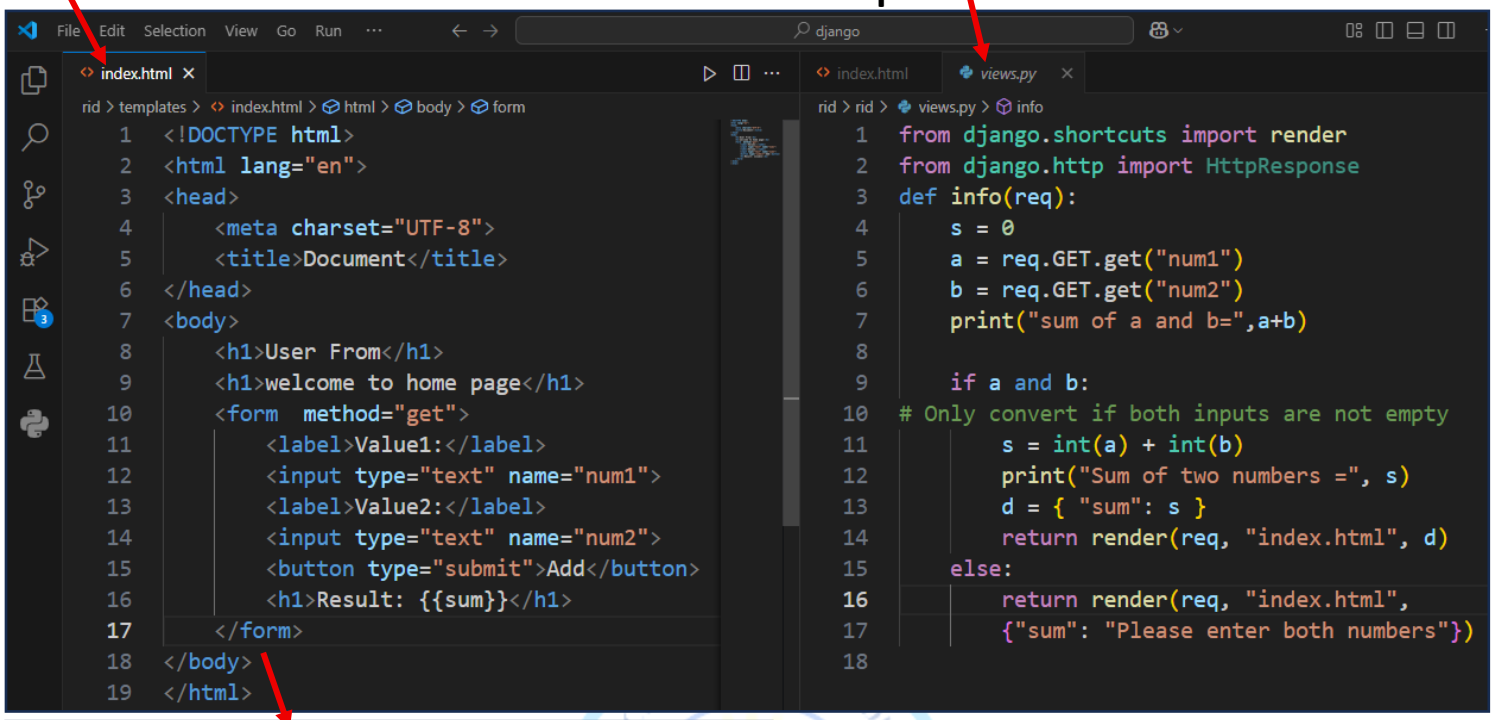
- **Max URL length** (depends on browser, usually ~2000 characters).
- No file upload support.
- Should not be used to change or delete data (not idempotent).
- Exposes data in browser history and server logs.

❖ Best Practice:

Use GET when:

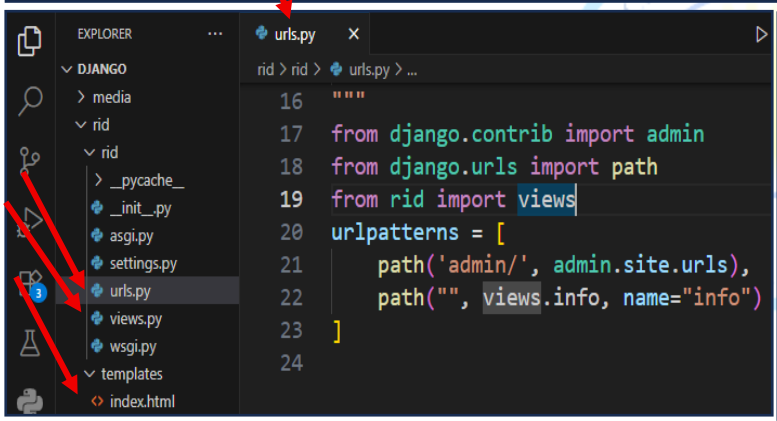
- You are **retrieving** data only.
- No **modification** to server-side data is needed.
- Data is not **sensitive**.

Get method Example:

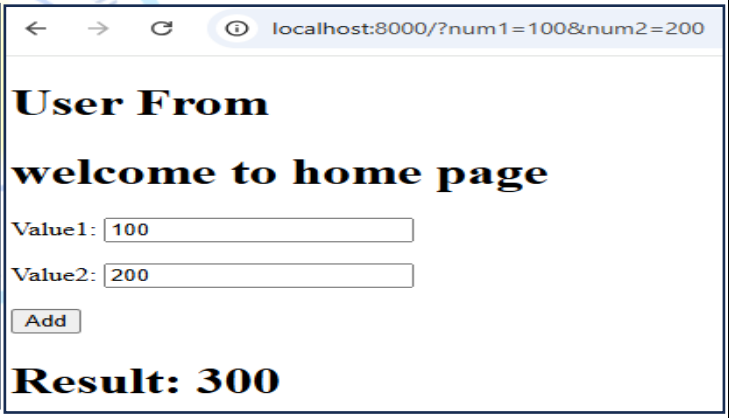


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h1>User From</h1>
    <h1>welcome to home page</h1>
    <form  method="get">
        <label>Value1:</label>
        <input type="text" name="num1">
        <label>Value2:</label>
        <input type="text" name="num2">
        <button type="submit">Add</button>
        <h1>Result: {{sum}}</h1>
    </form>
</body>
</html>
```

```
from django.shortcuts import render
from django.http import HttpResponse
def info(req):
    s = 0
    a = req.GET.get("num1")
    b = req.GET.get("num2")
    print("sum of a and b=",a+b)

    if a and b:
# Only convert if both inputs are not empty
        s = int(a) + int(b)
        print("Sum of two numbers =", s)
        d = { "sum": s }
        return render(req, "index.html", d)
    else:
        return render(req, "index.html",
        {"sum": "Please enter both numbers"})
```



```
"""
from django.contrib import admin
from django.urls import path
from rid import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.info, name="info")
]
```



1. **req**: Request object from the client.
2. **GET**: Holds data sent via URL (GET method).
3. **get("num1")**: Fetches the value of "num1" safely from GET data.

Note:-

- **get()** is optional but recommended.
  - You can access data like this:
- **req.GET["num1"]** # Raises error if 'num1' is missing
  - or safely with:
- **req.GET.get("num1")** # Returns None if 'num1' is missing
- So, **get()** is not mandatory, but it's safer to use to avoid errors if the key doesn't exist.

2. POST Method in Django

- POST method is used to **send data to the server** for processing — usually to create or update something. It is commonly used when submitting forms.

Syntax:

```
if request.method == 'POST':
    data = request.POST.get('key')
```

- request.POST is a dictionary-like object with form data.
- get('key') retrieves the submitted value.

◆ Example:

1. views.py

```
from django.shortcuts import render
def submit_form(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        return render(request, 'home.html',
                      {'name': name})
    return render(request, 'form.html')
```

2. form.html

```
<form method="POST">
    {% csrf_token %}
    <input type="text" name="name" >
    <input type="submit">
</form>
```

{% csrf_token %} is a Django template tag used inside HTML <form> tags to protect against **Cross-Site Request Forgery (CSRF) attacks**.

- It generates a unique token for each user session.
- Ensures that the form is submitted from your own website, not by a malicious third-party.
- Required for all POST forms in Django.

Usage:

```
<form method="POST">
    {% csrf_token %}
    <!-- form fields -->
</form>
```

Without it, Django will block form submission for security reasons.

Rule	Must/Optional
• Use method="POST" in **form**	: Must
• Use {% csrf_token %} in **form**	: Must (for security)
• Check if request.method == "POST"	:Must (to avoid errors) in views.py
• Use POST.get("key")	:Recommended for safe access

❖ Use Cases:

- Submitting forms
- Sending login data
- Uploading files
- Modifying server data

❖ Advantages:

1. Can send **secure/sensitive** data
2. Supports **large** data and file uploads.
3. Safer for data modification.

❖ Limitations:

- Must include **CSRF token** in Django forms.
- Should be used only for **data-changing operations**.
- Cannot be used for simple data retrieval (use GET for that).

❖ Best Practice:

Use POST when:

- You are submitting/modifying data.
- Data should remain **private** (not visible in URL).
- You want to avoid browser caching.

❖ Disadvantages:

1. Cannot be bookmarked or cached.
2. Not visible in URL (hard to debug sometimes).
3. Requires CSRF protection ({% csrf_token %}).

How to Take input from form tag in Django

Step 1: Create an HTML form to get two numbers

- Use the **<form>** tag with **method="POST"** to send data securely.
- Add two inputs for numbers with **name="num1" and name="num2".**
- Add **{% csrf_token %}** to protect against CSRF attacks.
- Add a submit button to send the form.

Example: - home.html:

```
<form method="POST">
  {% csrf_token %}
  Number 1: <input type="text" name="num1" >
  Number 2: <input type="text" name="num2" >
  <button type="submit">Sum</button>
</form>
{% if result is not None %}
  <h2>Result: {{ result }}</h2>
{% endif %}
```

{% csrf_token %} is a Django template tag used inside HTML <form> tags to protect against **Cross-Site Request Forgery (CSRF) attacks**.

- It generates a unique token for each user session.
- Ensures that the form is submitted from your own website, not by a malicious third-party.
- Required for all POST forms in Django.

Usage:

```
<form method="POST">
  {% csrf_token %}
  <!-- form fields -->
</form>
```

Without it, Django will block the form submission for security reasons.

Step 2: Write a Django view to process form data and calculate sum

- Import render to render templates.
- Define a view function, e.g., home.
- Initialize a variable result as None.
- Check if the request method is POST (form submitted).
- Retrieve num1 and num2 from request.POST.
- Convert the numbers to integers and calculate their sum.
- Pass the result and any other data to the template using a context dictionary.
- Render the template with the context.

Example views.py:

```
from django.shortcuts import render
def home(request):
  result = None
  if request.method == 'POST':
    num1 = request.POST.get('num1')
    num2 = request.POST.get('num2')
    if num1 and num2:
      result = int(num1) + int(num2)
  data = {
    'title': 'Welcome to rid home Page',
    'result': result
  } return render(request, 'home.html', data)
```

Example urls.py:

```
from django.contrib import admin
from django.urls import path
from rid import views
urlpatterns = [
  path('admin/', admin.site.urls),
  path('', views.home, name='home'),
]
```

Step 3: Configure URL to link to the view

- Import your view in urls.py.
- Create a URL pattern that routes the root URL to the home view.

Example-1 for post method.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Post Method</title>
</head>
<body>
    <h1>Welcome to Home Page!</h1>
    <p>Add the two numbers</p>
    <h2>Result: {{ sum }}</h2>
    <form method="POST">
        {% csrf_token %}
        <label>Value1:</label>
        <input type="text" name="num1">
        <label>Value2:</label>
        <input type="text" name="num2">
        <button type="submit">Add</button>
    </form>
</body>
</html>
```

views.py

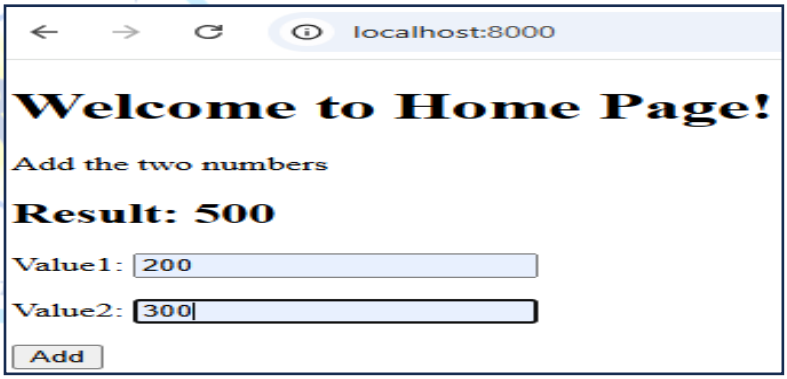
```
from django.shortcuts import render
from django.http import HttpResponse
def info(request):
    s = 0
    if request.method == "POST":
        a = request.POST.get("num1")
        b = request.POST.get("num2")
        if a and b:
            s = int(a) + int(b)
        else:
            s = "Please enter both numbers"
    return render(request, "index.html", {"sum": s})
```

EXPLORER

DJANGO
- media
- rid
  - rid
    - __pycache__
    - __init__.py
    - asgi.py
    - settings.py
    - urls.py
    - views.py
    - wsgi.py
  - templates
    - index.html

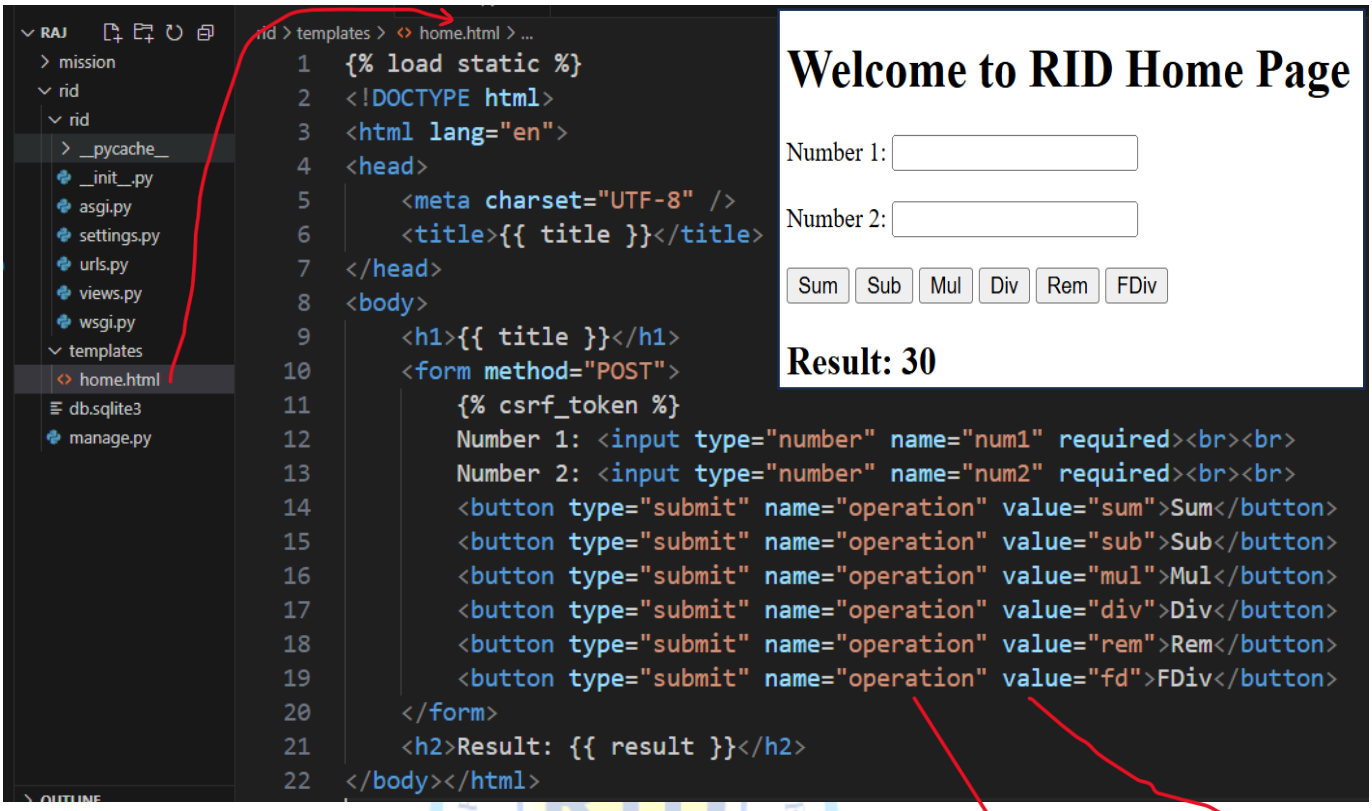
urls.py

```
"""
from django.contrib import admin
from django.urls import path
from rid import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.info, name="info")
]
```

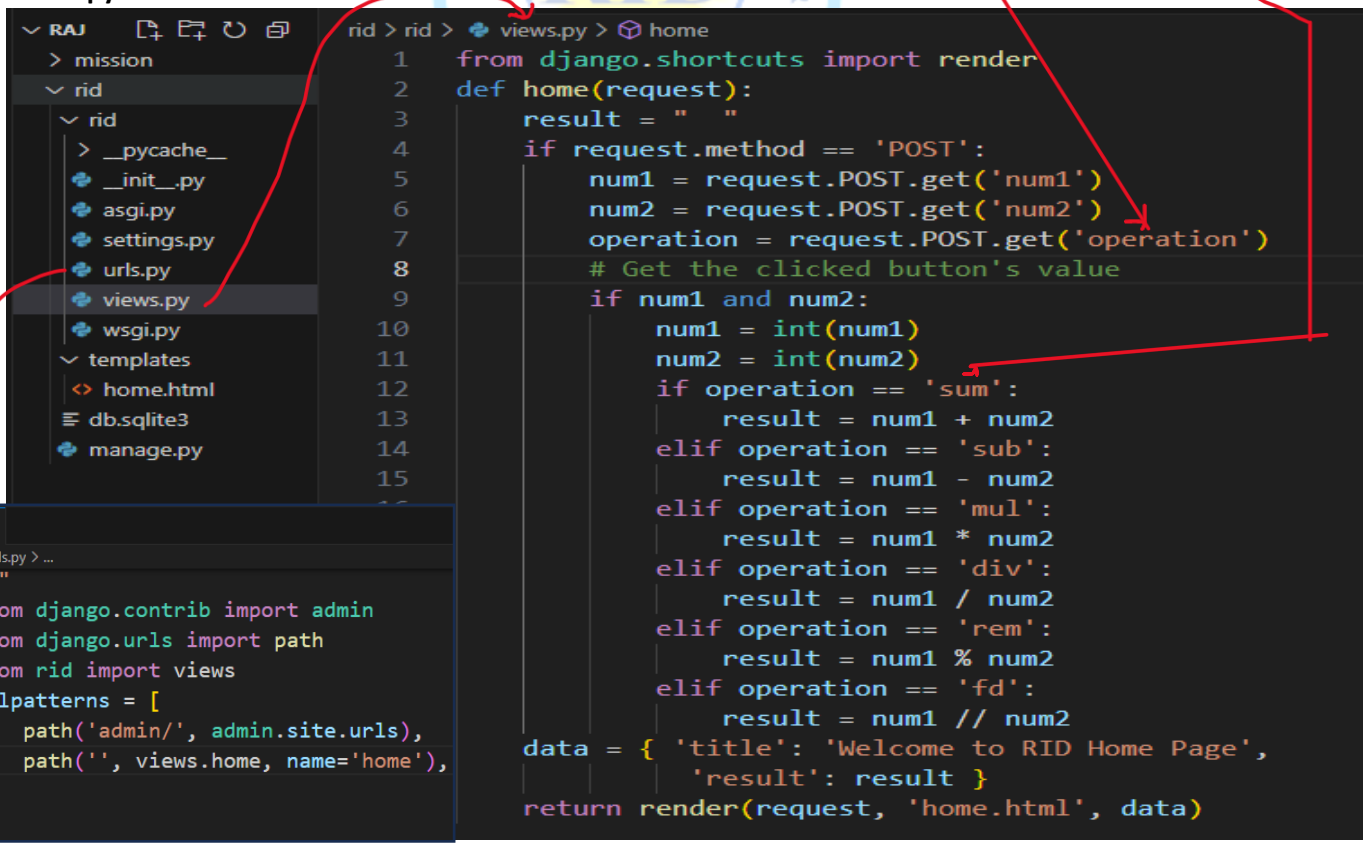


Rule	Must/Optional
• Use method="POST" in **form**	: Must
• Use {% csrf_token %} in **form**	: Must (for security)
• Check if request.method == "POST"	:Must (to avoid errors) in views.py
• Use POST.get("key")	:Recommended for safe access

Example-2: Perform the all-arithmetical operation templates index.html



Views.py



When user will click reset then number-1 and number-2 box will empty

views.py | home.html

rid > templates > home.html > html > body > h2

```
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ title }}</h1>
    <form method="POST">
        {% csrf_token %}
        Number 1: <input type="number" name="num1" value="{{num1}}" >
        Number 2: <input type="number" name="num2" value="{{ num2}}" >
        <button type="submit" name="op" value="sum">Sum</button>
        <button type="submit" name="op" value="sub">Sub</button>
        <button type="submit" name="op" value="mul">Mul</button>
        <button type="submit" name="op" value="div">Div</button>
        <button type="submit" name="op" value="rem">Rem</button>
        <button type="submit" name="op" value="fd">FDiv</button>
        <button type="submit" name="op" value="Reset">Reset</button>
    </form>
    <h2>Result: {{ result }}</h2>
</body>
</html>
```

127.0.0.1:8000

Welcome to RID

Number 1: 100

Number 2: 200

Sum | Sub | Mul | Div | Rem | FDiv | Reset

Result: 300

views.py

rid > rid > views.py > ...

```
from django.shortcuts import render
def home(request):
    result = ''
    num1 = ''
    num2 = ''
    if request.method == 'POST':
        operation = request.POST.get('op')
        if operation == 'Reset':
            num1 = '',num2 = '',result = ''
        else:
            num1 = request.POST.get('num1')
            num2 = request.POST.get('num2')
            if num1 and num2:
                num1_int = int(num1)
                num2_int = int(num2)
                if operation == 'sum':
                    result = num1_int + num2_int
                elif operation == 'sub':
                    result = num1_int - num2_int
                elif operation == 'mul':
                    result = num1_int * num2_int
                elif operation == 'div':
                    result = num1_int / num2_int
                elif operation == 'rem':
                    result = num1_int % num2_int
                elif operation == 'fdi':
                    result = num1_int // num2_int
    data = {
        'title': 'Welcome to RID ',
        'result': result,
        'num1': num1,
        'num2': num2
    }
    return render(request, 'home.html', data)
```

EXPLORER

RAJ

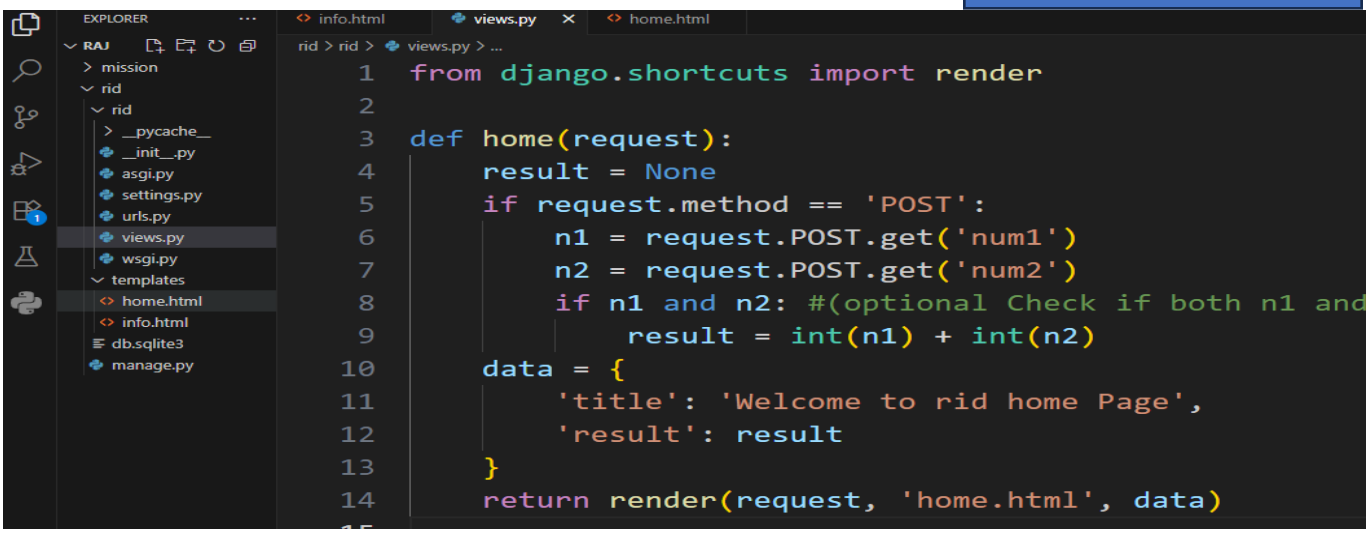
- mission
- rid
  - rid
    - __pycache__
    - __init__.py
    - asgi.py
    - settings.py
    - urls.py
    - views.py
    - wsgi.py
  - templates
    - home.html
  - db.sqlite3
  - manage.py

urls.py

rid > rid > urls.py > ...

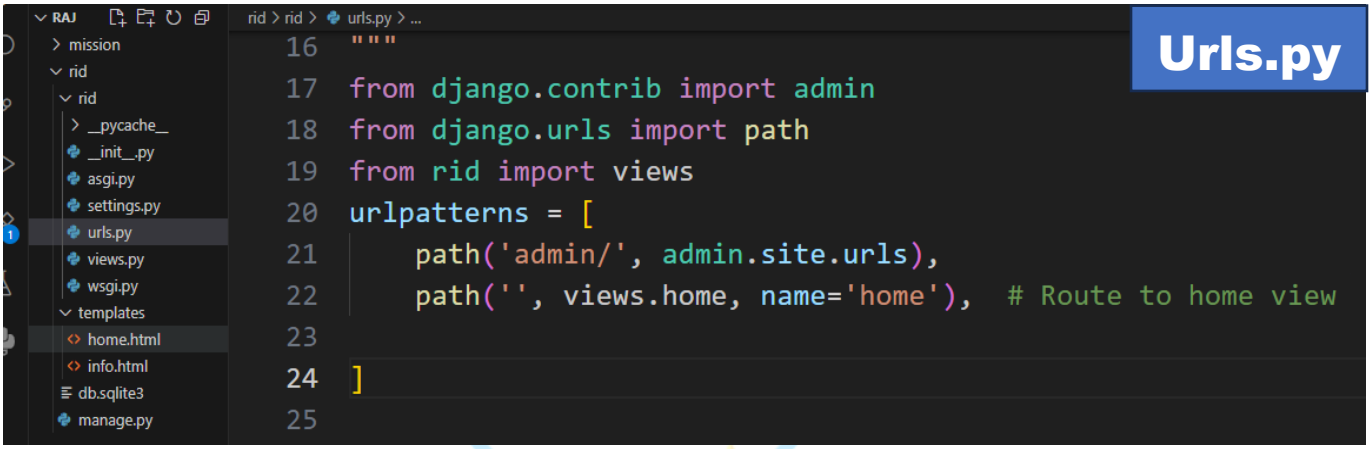
```
"""
from django.contrib import admin
from django.urls import path
from rid import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),

]

```

Example-3 Views.py



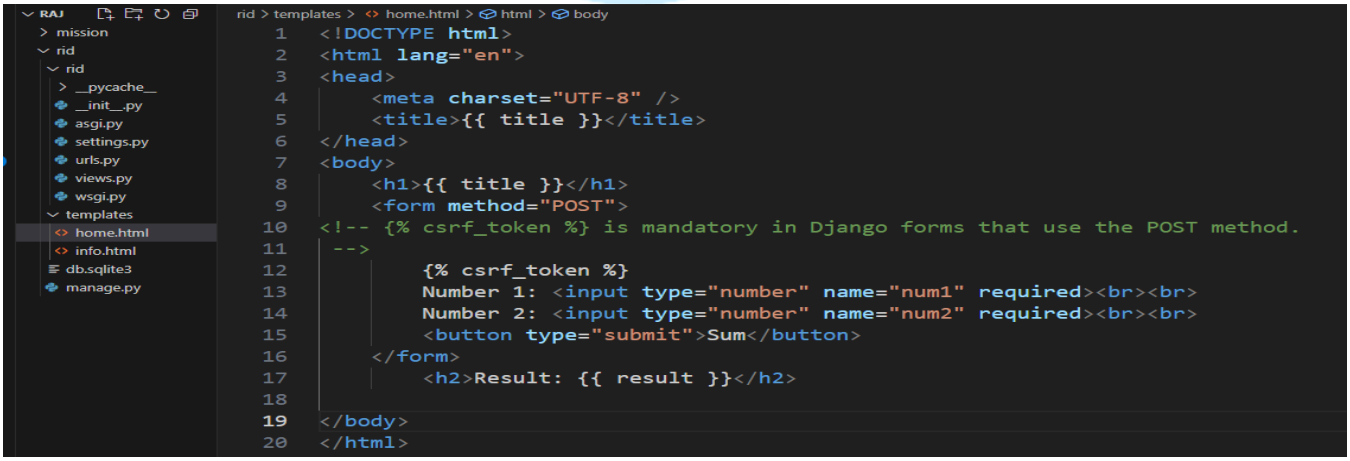
```
rid > rid > views.py > ...
1  from django.shortcuts import render
2
3  def home(request):
4      result = None
5      if request.method == 'POST':
6          n1 = request.POST.get('num1')
7          n2 = request.POST.get('num2')
8          if n1 and n2: #(optional Check if both n1 and
9              result = int(n1) + int(n2)
10     data = {
11         'title': 'Welcome to rid home Page',
12         'result': result
13     }
14     return render(request, 'home.html', data)
```

Urls.py

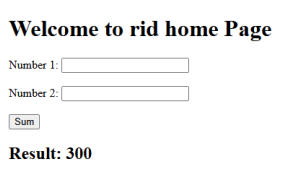


```
rid > rid > urls.py > ...
16 """
17 from django.contrib import admin
18 from django.urls import path
19 from rid import views
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path('', views.home, name='home'),  # Route to home view
23
24 ]
25
```

Home.html



```
rid > templates > home.html > html > body
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8" />
5      <title>{{ title }}</title>
6  </head>
7  <body>
8      <h1>{{ title }}</h1>
9      <form method="POST">
10 <!-- {% csrf_token %} is mandatory in Django forms that use the POST method.
11  -->
12         {% csrf_token %}
13         Number 1: <input type="number" name="num1" required><br><br>
14         Number 2: <input type="number" name="num2" required><br><br>
15         <button type="submit">Sum</button>
16     </form>
17         <h2>Result: {{ result }}</h2>
18
19 </body>
20 </html>
```



Django Templates for loop

- **Django Template for Loop – Step-by-Step with Syntax**

Step 1: Why We Use It? Note: Use in templates index.html file

- **We use the {% for %} loop in Django templates to:**
  - **Display multiple items** (like a list of names, products, subjects).
  - **Avoid repeating HTML code** manually.
  - Make the template **dynamic** and connected to data passed from views.py.

Step 2: Pass List from Django View (Python)

- In your views.py, pass a list using a context dictionary:

```
from django.shortcuts import render
    def home(request):
      data = {
        "subject": ["Math", "Science", "English"] }
      return render(request, "index.html", data)
```

Here, "subject" is a key and the list ["Math", "Science", "English"] is the value.

Step 3: Use for Loop in HTML Template (index.html)

Syntax:

```
{% for item in list %}
  {{ item }}
{% endfor %}
```

Example In your case:

```
<h2>Subjects:</h2>
<ul>
  {% for s in subject %}
    <li>{{ s }}</li>
  {% endfor %}
</ul>
```

Mandatory to write in before tag

Mandatory to write in end of tag

Step 4: Output in Browser

This will show:

Subjects:
- Math
- Science
- English

Step 5: Use Loop Helpers (Optional)

- Inside the loop, Django provides **extra built-in variables**:

Syntax	Meaning
❖ **{{ forloop.counter }}**	**: Position of the item (starting from 1)**
❖ **{{ forloop.first }}**	: True for the **first** item only
❖ **{{ forloop.last }}**	: True for the **last** item only

Step	Action
1	Define a list in views.py and pass it using context
2	Use {% for i in list %} ... {% endfor %} in template
3	Use {{ i }} to display each item
4	Use forloop.counter etc. for advanced control

Tag	Required?	Purpose
{% for i in list %}	✅ Yes	Start loop
{% endfor %}	✅ Yes	End loop
{{ i }}	✅ Yes	Display item inside loop

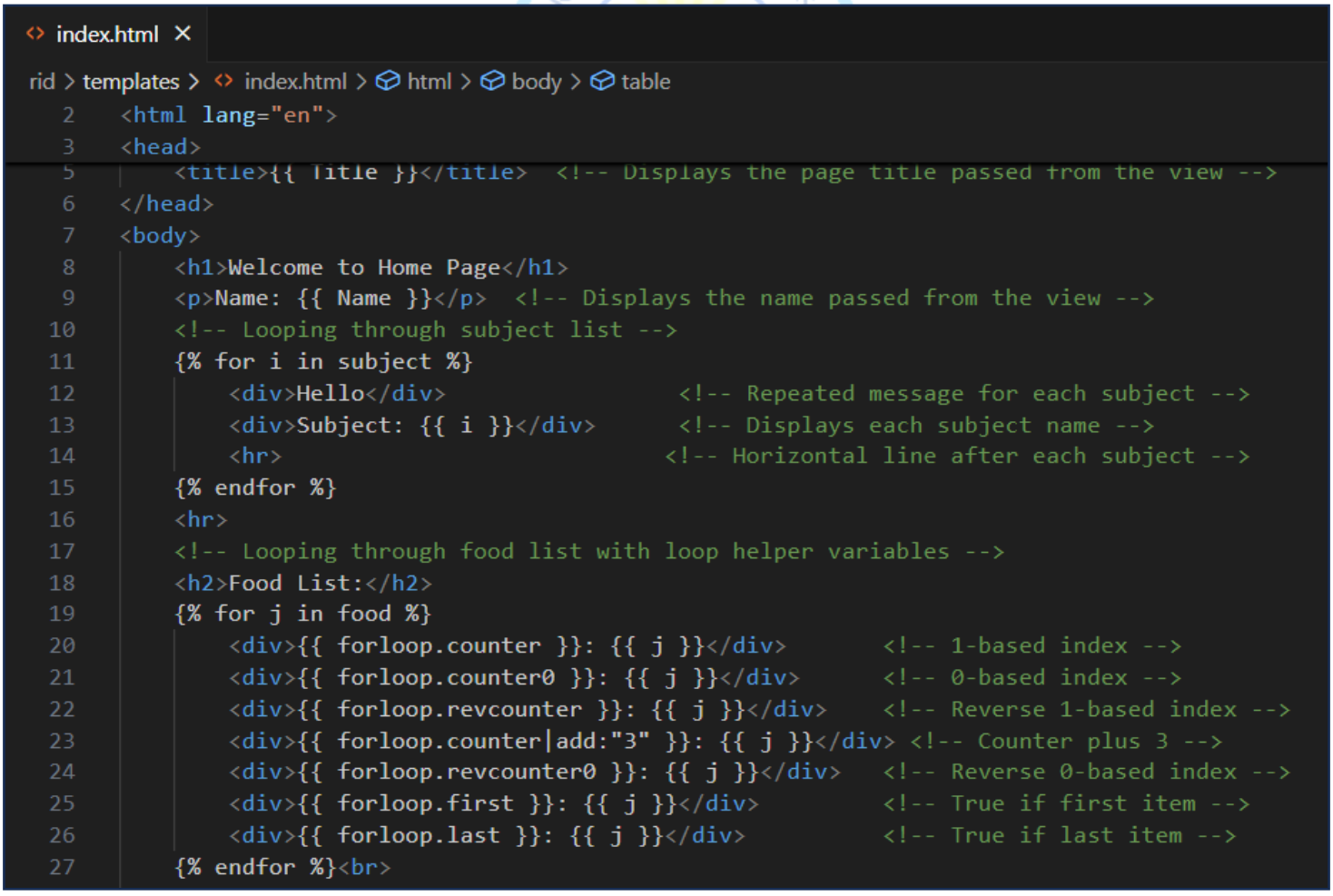
Example-1 views.py:

```
CODI...
myproject
rid
  rid
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    views.py
    wsgi.py
  templates
    index.html
  db.sqlite3
  manage.py
```

rid > rid > views.py > ...

```python
from django.http import HttpResponse
from django.shortcuts import render

def home(req):
    d = {
        "Title": "Home page",
        "Name": "Sangam Kumar",
        "Branch": "CSE",
        "Age": 18,
        "Roll_No": 16,
        "subject":["Python", "html", "css", "js", "Django"],
        "food":["Panipuri", "Burger", "Momos", "Samosa"],
        "info": [{"city":"Bhopal", "state":"M.p"},
                 {"city":"Patna", "state":"Bihar"},
                 {"city":"Delhi", "state":"New Delhi"}],
        "nest_dict":{"fend":"html css js", "Bend":"Pyton Djnago"}
    }
    return render(req, "index.html", d)
```

Index.html

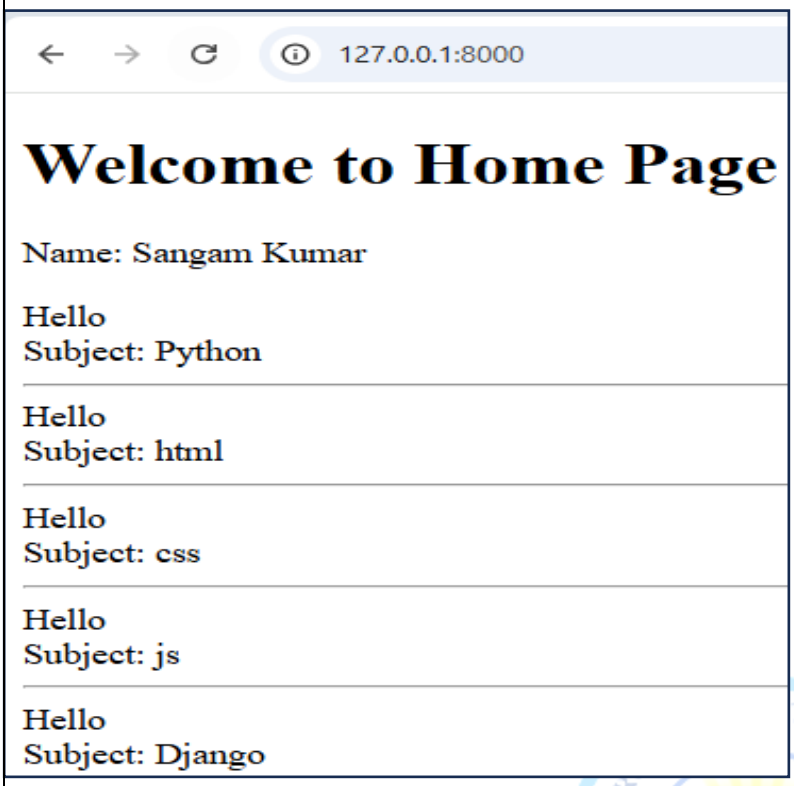


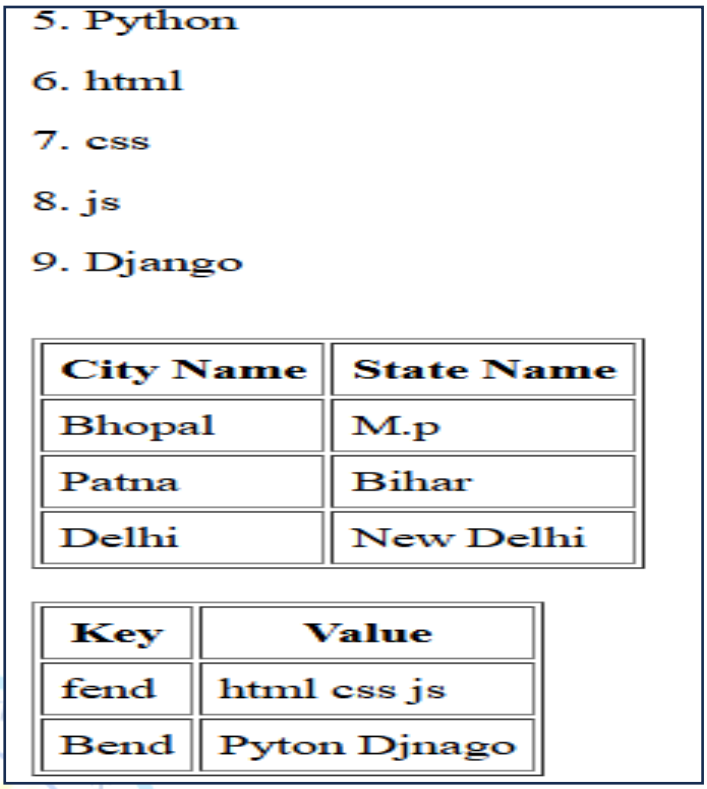
index.html

rid > templates > index.html > html > body > table

```
<html lang="en">
<head>
    <title>{{ Title }}</title>  <!-- Displays the page title passed from the view -->
</head>
<body>
    <h1>Welcome to Home Page</h1>
    <p>Name: {{ Name }}</p>  <!-- Displays the name passed from the view -->
    <!-- Looping through subject list -->
    {% for i in subject %}
        <div>Hello</div>                <!-- Repeated message for each subject -->
        <div>Subject: {{ i }}</div>     <!-- Displays each subject name -->
        <hr>                            <!-- Horizontal line after each subject -->
    {% endfor %}
    <hr>
    <!-- Looping through food list with loop helper variables -->
    <h2>Food List:</h2>
    {% for j in food %}
        <div>{{ forloop.counter }}: {{ j }}</div>       <!-- 1-based index -->
        <div>{{ forloop.counter0 }}: {{ j }}</div>      <!-- 0-based index -->
        <div>{{ forloop.revcounter }}: {{ j }}</div>    <!-- Reverse 1-based index -->
        <div>{{ forloop.counter|add:"3" }}: {{ j }}</div> <!-- Counter plus 3 -->
        <div>{{ forloop.revcounter0 }}: {{ j }}</div>   <!-- Reverse 0-based index -->
        <div>{{ forloop.first }}: {{ j }}</div>         <!-- True if first item -->
        <div>{{ forloop.last }}: {{ j }}</div>          <!-- True if last item -->
    {% endfor %}<br>
```

```
    <!-- Looping through subject list with counter starting from 5 -->
    {% for item in subject %}
        <p>{{ forloop.counter|add:"4" }}. {{ item }}</p>  <!-- Index starts from 5 -->
    {% endfor %}
    <!-- Displaying dictionary data inside a table -->
    <br>
    <table border="1" cellpadding="5">
        <tr>
            <th>City Name</th>
            <th>State Name</th>
        </tr>
        {% for n in info %}
        <tr>
            <td>{{ n.city }}</td>    <!-- Displays city from dictionary -->
            <td>{{ n.state }}</td>   <!-- Displays state from dictionary -->
        </tr>
        {% endfor %}  <!-- End the for loop -->
    </table>
    <!-- Displaying nested dictionary data in a new table -->
    <br>
    <table border="1" cellpadding="5">
        <tr>
            <th>Key</th>
            <th>Value</th>
        </tr>
        {% for key, value in nest_dict.items %}
        <tr>
            <td>{{ key }}</td>     <!-- Display dictionary key -->
            <td>{{ value }}</td>   <!-- Display dictionary value -->
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```





More about for loop in Django

By default, Django's {% for %} loop always starts from 0 or 1 using:

- {{ forloop.counter0 }} → starts from 0
- {{ forloop.counter }} → starts from 1

Note: - Can we change the starting number of the loop counter?

No, Django templates do not support custom loop starting indexes directly (like starting from 5 instead of 1). The loop index is **auto-managed** by Django.

Solution: - But here's how you can simulate it using simple math:

```
{% for item in subject %}
  <p>{{ forloop.counter|add:"4" }}. {{ item }}</p>  <!-- starts from 5 -->
{% endfor %}
```

Explanation:

- forloop.counter normally starts at 1
- |add:"4" adds 4 to it, so the loop shows: 5, 6, 7...

Here is the corrected code with explanations of the fixes:.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{ Title }}</title>  <!-- Displays the page title from view context -->
```

```
</head>
<body>
  <h1>Welcome to Home Page</h1>
  <p>Name: {{ Name }}</p>          <!-- Displays the name passed from the view -->
                                   <!-- Looping through subject list -->
  {% for i in subject %}
    <div>Hello</div>               <!-- Repeated message -->
    <div>Subject: {{ i }}</div>    <!-- Shows each subject name -->
    <hr>                           <!-- Adds a horizontal line after each item -->
  {% endfor %}
  <hr>                             <!-- Looping through food list with loop helper variables -->
  <h2>Food List:</h2>
  {% for j in food %}
    <div>{{ forloop.counter }}: {{ j }}</div>          <!-- 1-based index -->
    <div>{{ forloop.counter0 }}: {{ j }}</div>         <!-- 0-based index -->
    <div>{{ forloop.revcounter }}: {{ j }}</div>       <!-- Reverse 1-based (from end) -->
    <div>{{ forloop.revcounter0 }}: {{ j }}</div>      <!-- Reverse 0-based (from end) -->
    <div>{{ forloop.first }}: {{ j }}</div>            <!-- True for first item, else False -->
    <div>{{ forloop.last }}: {{ j }}</div>             <!-- True for last item, else False -->
  {% endfor %}
                                                       <!-- How to fetch list inside dictionary data  -->
  <br>
  <table border="1" cellpadding="5">
    <tr>
      <th>City Name</th>
      <th>State Name</th>
    </tr>
    {% for n in info %}
    <tr>
      <td>{{ n.city }}</td>                            <!-- Corrected access: no brackets needed -->
      <td>{{ n.state }}</td>
    </tr>
    {% endfor %}  <!-- End the for loop -->
  </table>

</body>
</html>
```

Explanation of fixes:

- **Access dictionary keys without brackets**:
  Instead of {{ n.[city] }}, use {{ n.city }} (dot notation) in Django templates.
- **Closed the for loop in table**:
  Added {% endfor %} to properly close the loop for info.
- **Proper indentation and HTML structure**:
  Made sure the <table> and loops are properly closed and formatted.

How to display the 100 box and image by using the for loop in Django in template's

EXPLORER

- DEMO
  - rid
    - rid
      - __pycache__
      - __init__.py
      - asgi.py
      - settings.py
      - urls.py
      - views.py
      - wsgi.py
    - static
      - allimage
        - img2.jpeg
        - panipuri.jpeg
      - css
        - style.css
      - templates
        - index.html
    - db.sqlite3
    - manage.py

index.html

rid > templates > index.html > ...

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8">
    <title>{{ Title }}</title>
<!-- ✅ Link external CSS from static -->
<link rel="stylesheet"
 href="{% static 'css/style.css' %}">
</head>
<body>
    <h1>Upload 100 Image in onediv </h1>
    <div class="p">
{% for i in Number %}
<!-- <div class="box">{{ i }}</div> -->
<div class="box">
<img src="{% static myimg %}"
width="100px" height="100px">
</div>
  {% endfor %}
</div>
</body></html>
```

style.css

rid > static > css > # style.css > .p

```
/* static/css/style.css */
h1{text-align: center;}
.p {
    width: 100%;
    min-height: 100vh;
    border: 2px solid red;
    display: flex;
    flex-wrap: wrap;
justify-content: space-evenly;
}
.box {
    width: 100px;
    height: 100px;
    border: 2px solid black;
    background-color: red;
    margin: 10px;
    text-align: center;
    line-height: 100px;
    font-weight: bold;
    color: white;
}
```

views.py

rid > rid > views.py > home

```
from django.http import HttpResponse
from django.shortcuts import render

def home(req):
    d={
        "Title":"Home page",
        "Name":"Sangam Kumar",
        "Food":["panipuri", "samosa", "Burger"
        "AGE":[18,20,22,30,44,56],
        "Number":range(50),
        "myimg":"allimage/panipuri.jpeg"
    }
    return render(req, "index.html", d)
```

urls.py

rid > rid > urls.py > ...

```
    2. Add a URL to urlpatterns:  path('blog/'
"""
from django.contrib import admin
from django.urls import path
from rid import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.home)
]

```

Setting.py

```
import os
STATIC_URL = '/static/'
STATICFILES_DIRS = [ os.path.join(BASE_DIR, "static")]
```

Upload 100 Image in onediv

{% if %} tag in Django

- In Django templates, the {% if %} tag is used to conditionally render content based on the value of variables passed from the view. It works similarly to if statements in Python but follows Django Template Language (DTL) syntax.

❖ Concept of {% if %} in Django Templates

- The {% if %} template tag evaluates a condition and renders a block of code only if the condition is True. This helps in *displaying dynamic content*, like showing messages, hiding sections, or changing appearance based on context.

Basic Syntax:

```
{% if condition %}
   <!-- Content if condition is true -->
{% elif another_condition %}
   <!-- Content if another condition is true -->
{% else %}
   <!-- Content if none of the conditions are true -->
{% endif %}
```

❖ Django {% if %} Tag – Key Points:

- Used to **conditionally display content** based on values from the view.
- Works like Python if, but in Django Template Language (DTL).
- If the **variable is missing or False**, the block is not rendered.
- Always close with {% endif %} to avoid errors.
- Use it to **show/hide content**, display messages, or switch layouts.

➢ In Django templates, the {% if %} tag supports the following comparison and logical operators:

	Operator	Description	Example
1.	==	Equal to	{% if x == 5 %}
2.	!=	Not equal to	{% if user != None %}
3.	>	Greater than	{% if score > passing_mark %}
4.	>=	Greater than or equal to	{% if attempts >= max_attempts %}
5.	<	Less than	{% if age < 18 %}
6.	<=	Less than or equal to	{% if qty <= stock %}
7.	in	Membership (left value is contained in right)	{% if "Math" in subjects %}
8.	not in	Negated membership	{% if item not in cart %}
9.	and	Logical AND	{% if x > 0 and x < 10 %}
10.	or	Logical OR	{% if is_staff or is_superuser %}
11.	not	Logical NOT	{% if not user.is_authenticated %}

Note: - Additionally, you can group conditions with parentheses for clarity:

```
{% if (x >= 0 and x <= 100) or is_admin %}
  <!-- do something -->
{% endif %}
```

Special "boolean" tests

- A variable by itself is treated as "truthy" or "falsy."

```
{% if messages %}
  <p>You have new messages.</p>
{% endif %}
```

- You can test for empty lists or strings:

```
{% if not errors %}
  <p>No validation errors!</p>
{% endif %}
```

❖ **Using filters inside {% if %}**

➢ Many of Django's built-in filters can be used to turn values into something testable. For

Example-1: **displays the divisible by 3**

```
{% if value|divisibleby:3 %}
  <!-- value is a multiple of 3 -->
{% endif %}
```

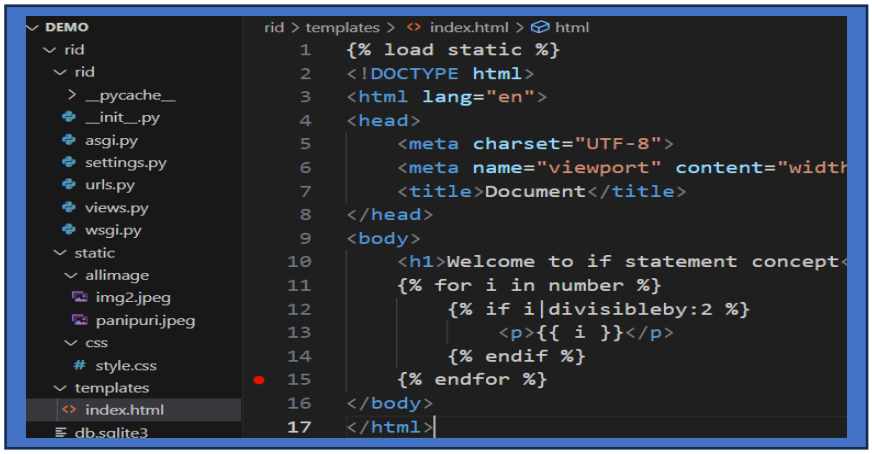
Example-2: displays only the even numbers:

```
{% for num in numbers %}
  {% if num|divisibleby:2 %}
    <p>{{ num }} is even</p>
  {% endif %}
{% endfor %}
```

Example: -3: divisible by **both** 3 **and** 5 in a Django template

```
{% for num in numbers %}
  {% if num|divisibleby:3 and num|divisibleby:5 %}
    <p>{{ num }} is divisible by both 3 and 5.</p>
  {% endif %}
{% endfor %}
```

Example:



How to Work with Django forms

Django Forms allow you to create and handle HTML forms easily. They provide a way to define form fields (like text input, email, file upload), validate user input, and render forms in templates. Django forms help securely process data submitted by users and integrate smoothly with views and models.

Common Django Form Fields Quick Reference

Field Name	Description	Example Usage
CharField	Text box	name = forms.CharField()
IntegerField	Number input	age = forms.IntegerField()
EmailField	Email input with validation	email = forms.EmailField()
PasswordInput	Password box (hidden input)	password = forms.CharField(widget=forms.PasswordInput)
DateField	Date picker	dob = forms.DateField()
BooleanField	Checkbox (True/False)	agree = forms.BooleanField()
ChoiceField	Dropdown/select box	color = forms.ChoiceField(choices=[('r','Red'),('g','Green')])
RadioSelect	Radio buttons (with ChoiceField)	rating = forms.ChoiceField(choices=..., widget=forms.RadioSelect)

File Upload Fields

Field Name	Description	Example Usage
FileField	Upload any file (e.g., PDF)	pdf = forms.FileField()
ImageField	Upload image files (jpg, png)	photo = forms.ImageField()
AudioField	*No built-in field* — use FileField for audio	audio = forms.FileField()
VideoField	*No built-in field* — use FileField for video	video = forms.FileField()

Text Area Field

Field Name	Description	Example Usage
Textarea	Multi-line text input	comments = forms.CharField(widget=forms.Textarea)

Notes:

- For **audio** and **video**, Django does **not** have special fields, use FileField and validate file types manually.
- For **ImageField**, install Pillow (pip install Pillow).
- File uploads require proper handling in **views**, **templates**, and **settings** (MEDIA_ROOT, MEDIA_URL).

Working with Django Forms: Step by Step

1. **Importing Django Forms**

Example: from django import forms

Desc: This imports Django's forms module to create form classes and fields.

2. **Create a Form Class**

Example:

```
class MyForm(forms.Form): # Define form fields
    name = forms.CharField(label="Name", max_length=100)
    age = forms.IntegerField(label="Age")
```

Desc:
- Define a form class inheriting from **forms.Form.**
- Add form fields like **CharField**, IntegerField, etc.
- Each field represents an **input element**.

3. **Create a View to Handle the Form**

Example:

```
from django.shortcuts import render
from .forms import MyForm
def my_view(request):
    if request.method == 'POST':
        form = MyForm(request.POST)  # Bind POST data to form
        if form.is_valid():        # Validate form data(built in method)
            # Access cleaned (validated) data
            name = form.cleaned_data['name']
            age = form.cleaned_data['age']
            # Process data (e.g., save to DB, compute something)
        result = f"Hello {name}, you are {age} years old."
        else:
            result = None
    else:
    form = MyForm()  # Empty form for GET request
        result = None
    return render(request, 'template.html', {'form': form, 'result': result})
```

Desc:
- Create view to show the form and process submitted data.
- Use `form.is_valid()` to validate input.
- Access validated data via `form.cleaned_data`.
- Render a template passing form and result.

4. **Create an HTML Template to Display the Form**

Example:

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}  <!-- Render form fields as paragraphs -->
    <button type="submit">Submit</button>
</form>

{% if result %}
    <p>Result: {{ result }}</p>
{% endif %}
```

Desc:
- Use {{ form.as_p }}, {{ form.as_table }}, or {{ form.as_ul }} to render form fields.
- Include {% csrf_token %} for security.
- Display results or error messages.

5. **Configure URL**

```
from django.urls import path
from . import views
urlpatterns = [
    path('myform/', views.my_view, name='myform'),]
```

Example-1

Explorer (Ctrl+Shift+E)

demo1.py | demo3.js | demo5.html | forms.py | views.py | urls.py | index.h

raj > raj > forms.py > ...

```python
from django import forms
class UserForms(forms.Form):  #(forms.Form): Means UserForms inherits from Django's built-in Form clas
    num1 = forms.CharField(
        label="Value 1",
        widget=forms.TextInput(attrs={'class': 'myinput', 'placeholder': 'Enter value 1'})
    )
    num2 = forms.CharField(
        label="Value 2",
        widget=forms.TextInput(attrs={'class': 'myinput', 'placeholder': 'Enter value 2'})
    )

```

EXPLORER

views.py

raj > raj > views.py > ...

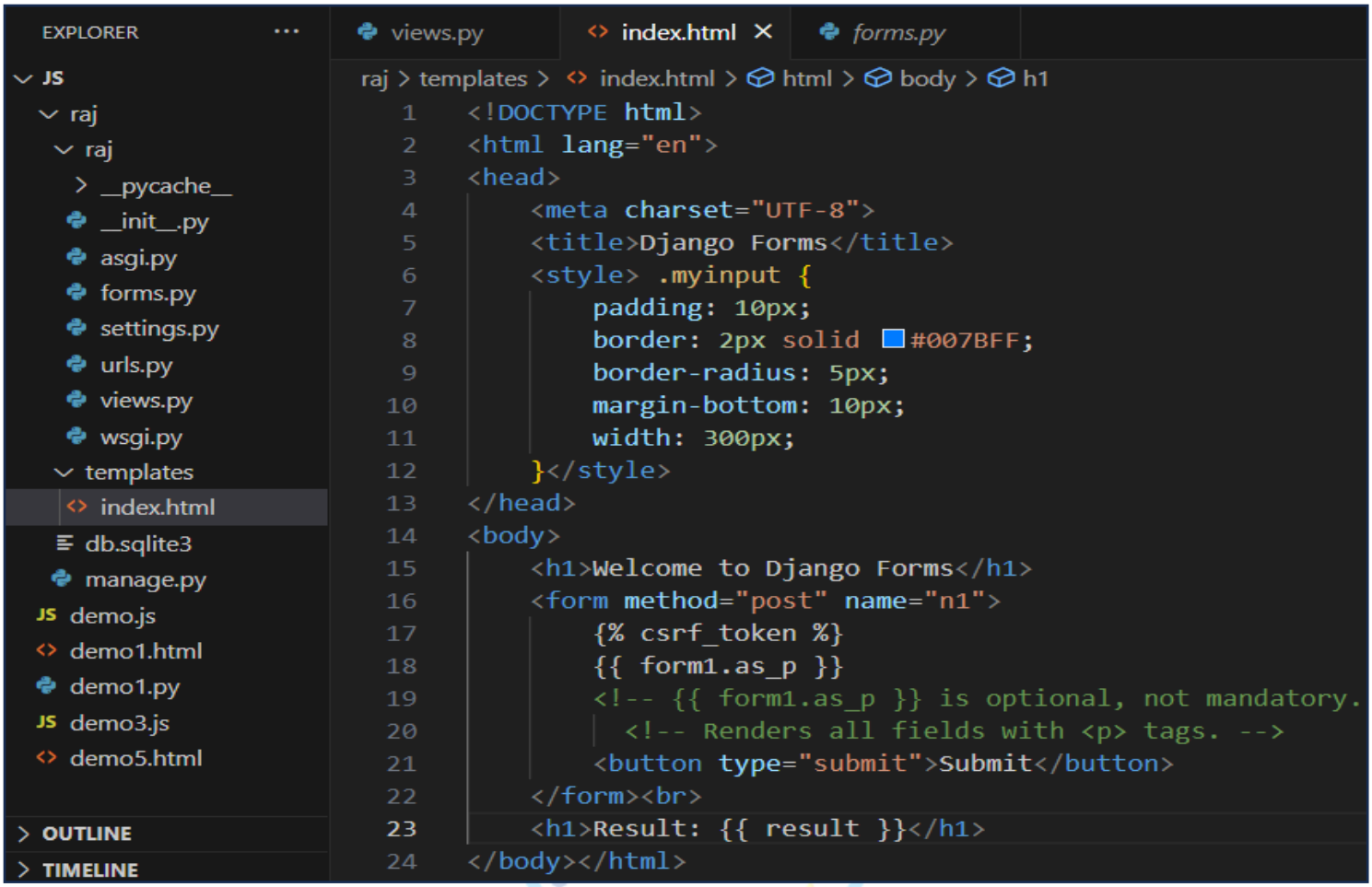
```python
from django.shortcuts import render
from .forms import UserForms
def uf(request):
    result = None
    if request.method == 'POST':
        form = UserForms(request.POST)  # Bind POST data to the form
        if form.is_valid():             # Validate the form data
            # Access cleaned and validated form data
            num1 = int(form.cleaned_data['num1'])
            num2 = int(form.cleaned_data['num2'])
            result = num1 + num2        # Perform addition
    else:
        form = UserForms()              # Create an empty form for GET
    context = {
        'form1': form,
        'result': result
    }

    return render(request, 'index.html', context)
```

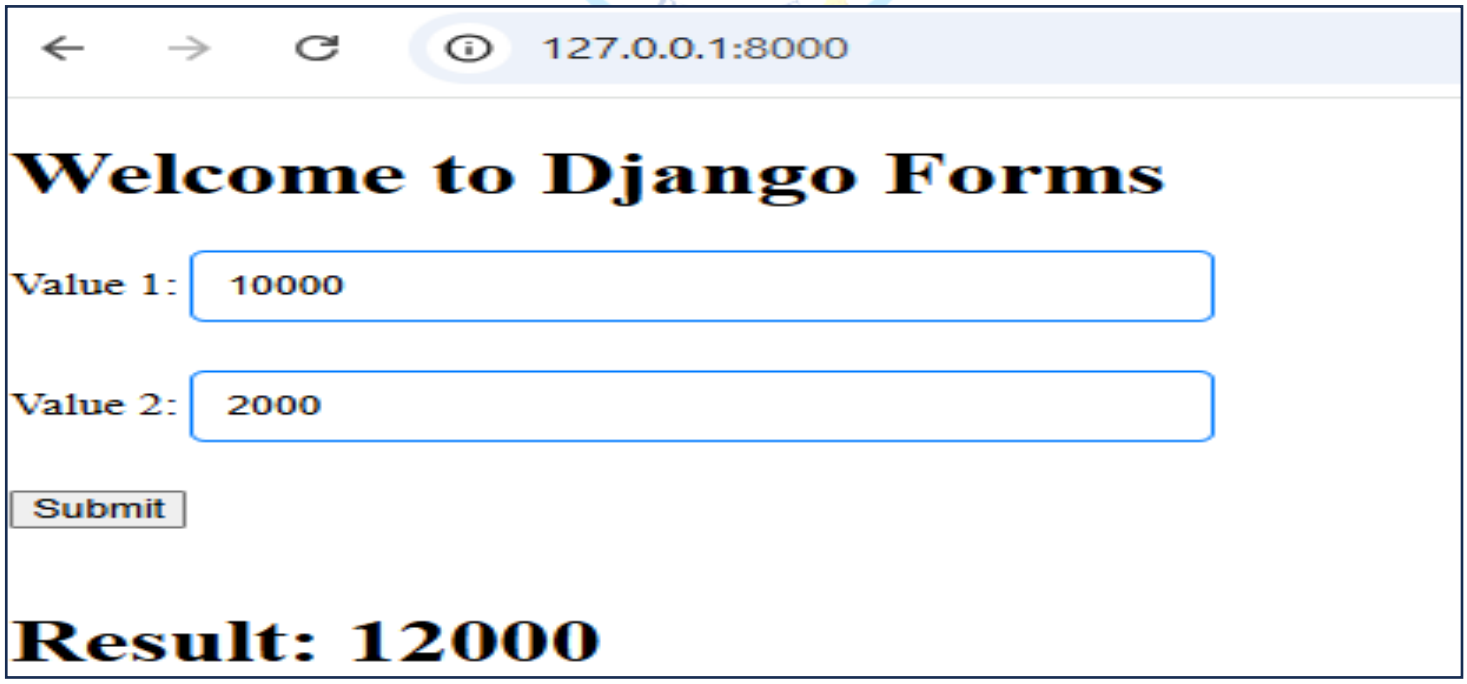
EXPLORER

views.py | index.html | urls.py

raj > raj > urls.py > ...

```python
from django.contrib import admin
from django.urls import path
from raj import views

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


EXPLORER
views.py
index.html
forms.py
raj > templates > index.html > html > body > h1
JS
raj
raj
__pycache__
__init__.py
asgi.py
forms.py
settings.py
urls.py
views.py
wsgi.py
templates
index.html
db.sqlite3
manage.py
demo.js
demo1.html
demo1.py
demo3.js
demo5.html
OUTLINE
TIMELINE
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8">
5 <title>Django Forms</title>
6 <style> .myinput {
7 padding: 10px;
8 border: 2px solid #007BFF;
9 border-radius: 5px;
10 margin-bottom: 10px;
11 width: 300px;
12 }</style>
13 </head>
14 <body>
15 <h1>Welcome to Django Forms</h1>
16 <form method="post" name="n1">
17 {% csrf_token %}
18 {{ form1.as_p }}
19 <!-- {{ form1.as_p }} is optional, not mandatory.
20 <!-- Renders all fields with <p> tags. -->
21 <button type="submit">Submit</button>
22 </form>
23 <h1>Result: {{ result }}</h1>
24 </body></html>


127.0.0.1:8000
Welcome to Django Forms
Value 1: 10000
Value 2: 2000
Submit
Result: 12000

List of common Django form fields

from django import forms

```
class SimpleForm(forms.Form):
    name = forms.CharField()         # Text input
    age = forms.IntegerField()       # Integer input
    email = forms.EmailField()       # Email input
    password = forms.CharField(widget=forms.PasswordInput)  # Password input
    dob = forms.DateField()          # Date input
    agree = forms.BooleanField()     # Checkbox (True/False)
    rating = forms.ChoiceField(choices=[('1','1'), ('2','2'), ('3','3')])  # Dropdown select
```

- **CharField:** Text box
- **IntegerField:** Number input
- **EmailField:** Email input with validation
- **PasswordInput:** Password textbox (hides input)
- **DateField:** Date picker input
- **BooleanField:** Checkbox
- **ChoiceField:** Dropdown/select box with options

Example

from.py:

```
from django import forms
class FeedbackForm(forms.Form):
    # Text input for name or feedback text
    name = forms.CharField(label="Your Name", max_length=100)
    # Email input field
    email = forms.EmailField(label="Your Email")

    # Radio buttons: use ChoiceField with RadioSelect widget
    RATING_CHOICES = [
        ('1', 'Poor'),
        ('2', 'Average'),
        ('3', 'Good'),
        ('4', 'Excellent'),
    ]
    rating = forms.ChoiceField(
        choices=RATING_CHOICES,
        widget=forms.RadioSelect,
        label="Rate our service"
    )
    # File upload field for PDF
    pdf_upload = forms.FileField(
        label="Upload PDF",
        required=False,
```

```
    help_text="Upload a PDF file."
  )

  # Image upload field
  image_upload = forms.ImageField(
    label="Upload Image",
    required=False,
    help_text="Upload an image file."
  )

  # Textarea for detailed feedback
  comments = forms.CharField(
    label="Comments",
    widget=forms.Textarea(attrs={'rows':4, 'cols':40}),
    required=False
  )
```

Additional Notes:

- For handling file uploads (PDF/image), make sure your Django settings have MEDIA_ROOT and MEDIA_URL configured.
- In your view, use request.FILES along with request.POST to process uploaded files.
- You may want to add validation to restrict file types or sizes.

Field	Purpose	Notes
CharField	Single-line text input	Use max_length to limit size
EmailField	Email input with validation	
ChoiceField + RadioSelect	Radio buttons for selecting one option	choices defines options
FileField	Upload files (like PDF)	Handle file validation & upload
ImageField	Upload images	Requires Pillow library installed
Textarea	Multi-line text input (comments)	Use widget=forms.Textarea

Example-2:

1. forms.py

```
from django import forms
class FeedbackForm(forms.Form):
    name = forms.CharField(label="Your Name", max_length=100)
    email = forms.EmailField(label="Your Email")

    RATING_CHOICES = [
        ('1', 'Poor'),
        ('2', 'Average'),
        ('3', 'Good'),
        ('4', 'Excellent'),
    ]
    rating = forms.ChoiceField(
        choices=RATING_CHOICES,
        widget=forms.RadioSelect,
        label="Rate our service"
    )

    pdf_upload = forms.FileField(
        label="Upload PDF",
        required=False,
        help_text="Upload a PDF file."
    )

    image_upload = forms.ImageField(
        label="Upload Image",
        required=False,
        help_text="Upload an image file."
    )

    comments = forms.CharField(
        label="Comments",
        widget=forms.Textarea(attrs={'rows':4, 'cols':40}),
        required=False
    )
```

3. views.py

```
from django.shortcuts import render
from .forms import FeedbackForm

def feedback_view(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST, request.FILES)  # Include FILES for uploads
        if form.is_valid():
            # Access cleaned data
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            rating = form.cleaned_data['rating']
            pdf = form.cleaned_data['pdf_upload']
            image = form.cleaned_data['image_upload']
            comments = form.cleaned_data['comments']

            # Here you can save the data or process the files
            result = f"Thanks {name} for your feedback!"

        else:
            result = None
    else:
        form = FeedbackForm()
        result = None

    return render(request, 'feedback.html', {'form': form, 'result': result})
```

4. feedback.html (template)

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Feedback Form</title>
</head>
<body>
    <h1>Feedback Form</h1>
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Submit</button>
    </form>

    {% if result %}
```

```
    <p>{{ result }}</p>
  {% endif %}
</body>
</html>
```

4. Settings (in settings.py)

```
python
CopyEdit
# Add these to handle media uploads
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
```

5. URL Configuration

```
python
CopyEdit
from django.urls import path
from . import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
  path('feedback/', views.feedback_view, name='feedback'),
]

# Serve media files during development
if settings.DEBUG:
  urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Summary

- **Form**: Handles inputs (text, email), radio buttons, file uploads, textarea.
- **View**: Processes POST & file data, validates form, and prepares result.
- **Template**: Renders form and result, includes enctype="multipart/form-data" for file uploads.
- **Settings**: Configure media URL and root folder.
- **URLs**: Map URL to view and serve media in development.

Base Conversion Project in Django

index.html cal\templates\index.html

```
64  </head>
65  <body>
66      <form method="post">
67          <h2>RID Base Converter</h2>
68          {% csrf_token %}
69          <label>Number:</label>
70          <input type="text" name="number" required>
71          <label>From Base:</label>
72          <select name="from_base">
73              <option value="2">Binary</option>
74              <option value="8">Octal</option>
75              <option value="10" selected>Decimal</option>
76              <option value="16">Hexadecimal</option>
77          </select>
78          <label>To Base:</label>
79          <select name="to_base">
80              <option value="2">Binary</option>
81              <option value="8">Octal</option>
82              <option value="10">Decimal</option>
83              <option value="16">Hexadecimal</option>
84          </select>
85          <button type="submit">Convert</button>
86          {% if result %}
87              <h3>Result: {{ result }}</h3>
88          {% endif %}
89      </form>
90  </body></html>
```

views.py

```
1  from django.shortcuts import render
2  def info(req):
3      result = None
4      if req.method == 'POST':
5          number = req.POST.get('number')
6          from_base = int(req.POST.get('from_base'))
7          to_base = int(req.POST.get('to_base'))
8          # Convert the input number to decimal (base 10)
9          decimal_number = int(number, from_base)
10         # Convert decimal to the desired base
11         if to_base == 2:
12             result = bin(decimal_number)[2:]
13         elif to_base == 8:
14             result = oct(decimal_number)[2:]
15         elif to_base == 10:
16             result = str(decimal_number)
17         elif to_base == 16:
18             result = hex(decimal_number)[2:]
19         else:
20             result = "Unsupported base."
21     return render(req, 'index.html', { 'result': result})
```

Why [2:] is used

Functions like bin(), oct(), and hex() in Python **return strings with a prefix**:

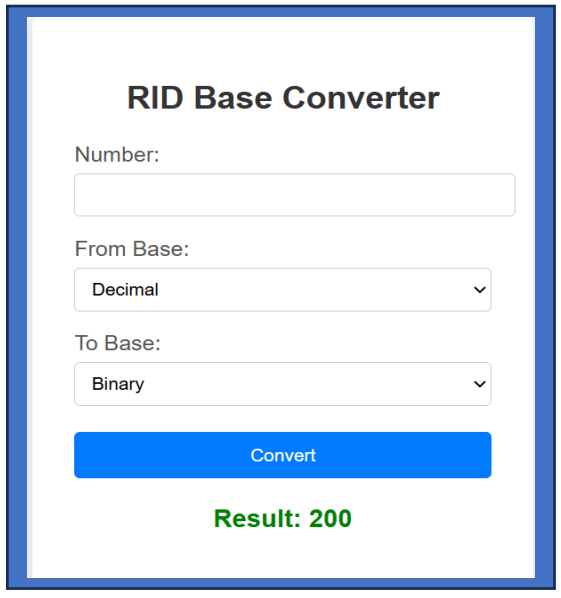
- bin(10) → '0b1010'
- oct(10) → '0o12'
- hex(10) → '0xa'

These prefixes (0b, 0o, 0x) indicate the base of the number.

Example:

bin(10) → '0b1010'
bin(10)[2:] → '1010'

```
<style> body { display: flex;
        justify-content: center;
        align-items: center;
        height: 100vh; } form {
        padding: 30px;
        border-radius: 8px;
        box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
        width: 300px; } label { display: block; margin-top: 15px;
    } input[type="text"], select {
     width: 100%;
        padding: 8px;
        margin-top: 5px;
        border-radius: 4px;
        border: 1px solid #ccc; }
    button { width: 100%; padding: 10px; margin-top: 20px;
        background-color: #007bff;  border-radius: 4px; cursor: pointer;
    } button:hover {background-color: #0056b3;} </style>
```



RID project Folder Structure

```
rid/                          ← Project root folder
├── manage.py                 ← Django's command-line utility
├── db.sqlite3                ← (Created after first migrate)
│
├── rid/                      ← Main Django project (configuration) folder
│   ├── __init__.py
│   ├── settings.py           ← Settings (configured for static + templates)
│   ├── urls.py               ← URL routing (connected to views below)
│   ├── views.py              ← You added this here (contains page views)
│   ├── wsgi.py
│   └── asgi.py
│
├── static/                   ← Static files folder
│   ├── css/
│   │   └── style.css         ← Custom styling
│   ├── js/
│   │   └── script.js         ← Optional JavaScript
│   └── allimg/
│       └── logo.png          ← Logo for navbar
│
├── templates/                ← HTML templates folder
│   ├── index.html            ← Main landing page
│   ├── home.html
│   ├── about.html
│   ├── ebook.html
│   ├── service.html
│   ├── news.html
│   └── gallery.html
│
└── staticfiles/              ← Auto-created when running `collectstatic` (optional)
```

❖ **You Can Add Later**

- **media/:** for user-uploaded files.
- **main/:** separate Django app (best practice) with its own views.py, urls.py
- **templates/base.html:** for layout reuse via {% extends %}.

❖ **Setting.py**

 # Static files (CSS, JavaScript, Images)
STATIC_URL = '/static/'
#add this line
STATICFILES_DIRS = [BASE_DIR / "static"]
 # Folder for your static files (CSS, JS, images, etc.)
STATIC_ROOT = BASE_DIR / "staticfiles"
 # (optional, used when running collectstatic)

❖ **Templates (index.html) In your index.html, link like this**

- {% load static %}
- <link rel="stylesheet" href="{% static 'css/style.css' %}">
- <img src="{% static 'images/logo.png' %}" alt="Logo">
- <script src="{% static 'js/script.js' %}"></script>

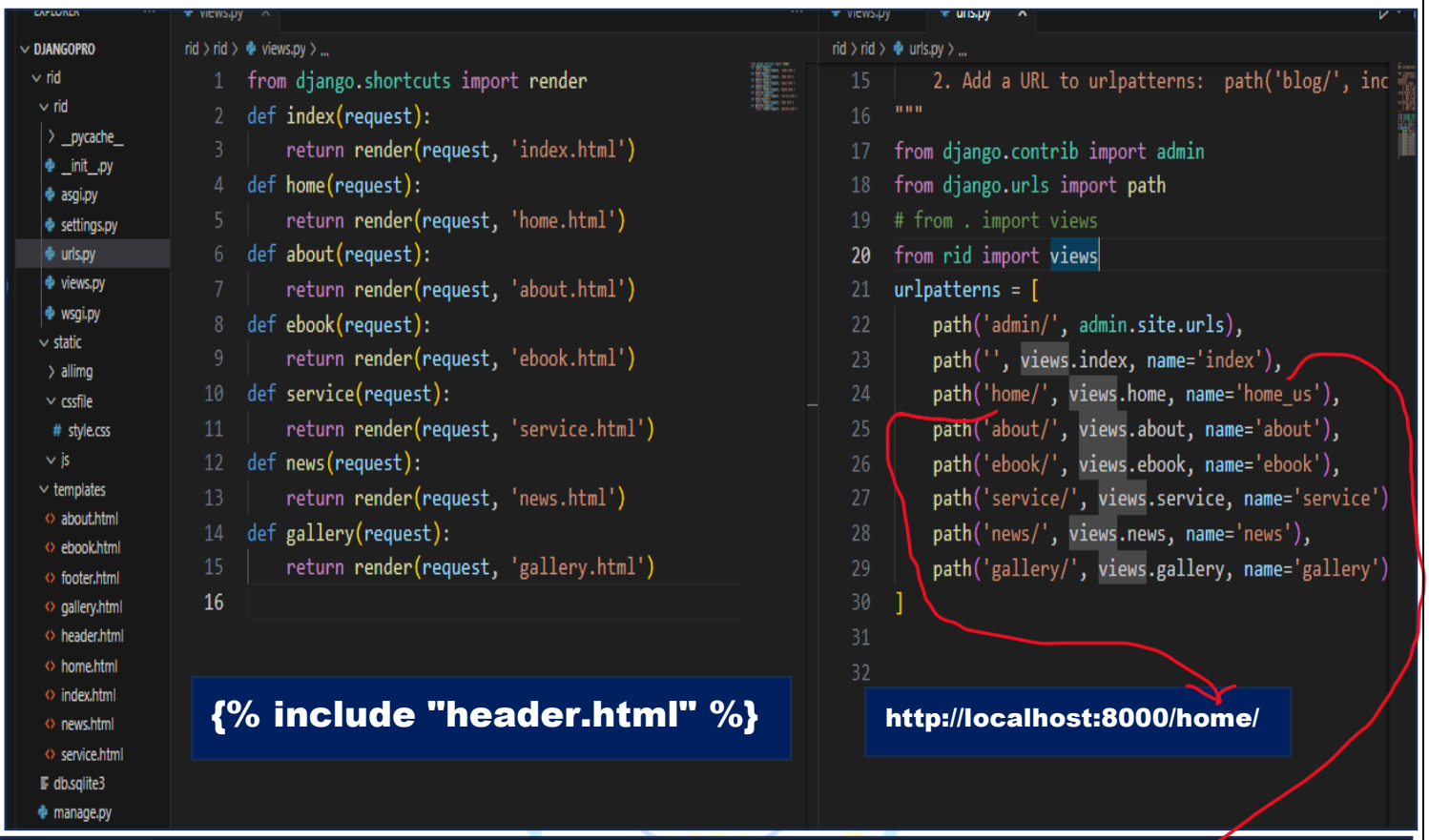
To **access static files** such as **CSS, images, JavaScript, PDFs**, etc., in Django, you need to configure your settings.py to properly locate and serve those files — especially during development.
Here's what you need to **add or modify** in your settings.py Update this section at the bottom of your file

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

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rid',  # your app
]
```

{% include "header.html" %}

Example-1 Mini Project:



index.html

rid > templates > index.html > html > body > header > nav.navbar > ul.nav-links > li

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>RID Bharat</title>
    <link rel="stylesheet" href="{% static 'cssfile/style.css' %}">
</head>
<body>
    <header>
        <nav class="navbar">
            <div class="logo">
                <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="RID Bharat Logo">
            </div>
            <ul class="nav-links">
                <li><a href="{% url 'home_us' %}" target="_self">Home</a></li>
                <li><a href="{% url 'about' %}" target="_blank">About</a></li>
                <li><a href="{% url 'ebook' %}">Ebook</a></li>
                <li><a href="{% url 'service' %}">Service</a></li>
                <li><a href="{% url 'news' %}">News</a></li>
                <li><a href="{% url 'gallery' %}">Gallery</a></li>
            </ul>
        </nav>
```

```
<!-- Links using standard HTML file paths -->
<ul class="nav-links">
    <li><a href="home.html" target="_self">Home</a></li>
    <li><a href="about.html" target="_blank">About</a></li>
    <li><a href="ebook.html">Ebook</a></li>
    <li><a href="service.html">Service</a></li>
    <li><a href="news.html">News</a></li>
    <li><a href="gallery.html">Gallery</a></li>
</ul>
<!-- Links using Django template URL tags -->
<ul class="nav-links">
    <li><a href="{% url 'home_us' %}" target="_self">Home</a></li>
    <li><a href="{% url 'about' %}" target="_blank">About</a></li>
    <li><a href="{% url 'ebook' %}">Ebook</a></li>
    <li><a href="{% url 'service' %}">Service</a></li>
    <li><a href="{% url 'news' %}">News</a></li>
    <li><a href="{% url 'gallery' %}">Gallery</a></li>
</ul>
```

Header.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>RID Bharat</title>
  <link rel="stylesheet" href="{% static 'cssfile/style.css' %}">
</head>
<body>
  <header>
    <nav class="navbar">
      <div class="logo">
        <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="RID Bharat Logo">
      </div>
      <ul class="nav-links">
        <li><a href="{% url 'home_us' %}" target="_self">Home</a></li>
        <li><a href="{% url 'about' %}" target="_blank">About</a></li>
        <li><a href="{% url 'ebook' %}">Ebook</a></li>
        <li><a href="{% url 'service' %}">Service</a></li>
        <li><a href="{% url 'news' %}">News</a></li>
        <li><a href="{% url 'gallery' %}">Gallery</a></li>
      </ul>
    </nav>
  </header>
```

Example: index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>RID Bharat</title>
  <link rel="stylesheet" href="{% static 'cssfile/style.css' %}">
</head>
<body>
  <header>
    <nav class="navbar">
      <div class="logo">
        <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="RID Bharat Logo">
      </div>
      <!-- Links using Django template URL tags -->
      <ul class="nav-links">
        <li><a href="{% url 'home_us' %}" target="_self">Home</a></li>
        <li><a href="{% url 'about' %}" target="_blank">About</a></li>
        <li><a href="{% url 'ebook' %}">Ebook</a></li>
        <li><a href="{% url 'service' %}">Service</a></li>
        <li><a href="{% url 'news' %}">News</a></li>
        <li><a href="{% url 'gallery' %}">Gallery</a></li>
      </ul>
    </nav>
  </header>
  <!-- Landing Page Content -->
  <section class="landing">
    <h1>Welcome to RID Bharat</h1>
    <p>We focus on Research, Innovation, and Discovery to build the future of India through science and technology.
      Our mission is to empower students, scientists, and innovators to bring impactful change with new ideas and.</p>
  </section> <!-- Team Member Section -->
  <h1>All Team Member Images</h1>
  <div class="tmimg">
    <div class="member">
      <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="Member 1">
      <p>Rohit Sharma</p>
    </div>
    <div class="member">
      <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="Member 2">
      <p>Anita Verma</p>
    </div>
    <div class="member">
      <img src="{% static 'allimg/ridmainlogo.jpg' %}" alt="Member 3">
      <p>Vikram Singh</p>
    </div>
```

In this place access header like this {% include "header.html"

footer.html

In this place access header like this {% include "footer.html"

```
<!-- Footer -->
  <footer class="footer">
    <div class="footer-container">
      <div class="footer-section about">
        <h2>RID Bharat</h2>
        <p>Research, Innovation, and Discovery for a better India. We empower students, scientists, and
          innovators to build the future through science and technology.</p>
      </div>
      <div class="footer-section links">
        <h3>Quick Links</h3>
        <ul>
          <li><a href="{% url 'home_us' %}">Home</a></li>
          <li><a href="{% url 'about' %}">About</a></li>
          <li><a href="{% url 'ebook' %}">Ebook</a></li>
          <li><a href="{% url 'service' %}">Service</a></li>
          <li><a href="{% url 'news' %}">News</a></li>
          <li><a href="{% url 'gallery' %}">Gallery</a></li>
        </ul> </div>
      <div class="footer-section contact">
        <h3>Contact Us</h3>
        <p>Email: info@ridbharat.com</p><br>
        <p>Phone: +91-9202707903</p><br>
        <p>Location: Bhopal, India</p><br>
        <a href="#"><img src="{% static 'allimg/facebook.png' %}" alt="Facebook" id="img1"></a>
        <a href="#"><img src="{% static 'allimg/twitter.png' %}" alt="Twitter" id="img1"></a>
        <a href="#"><img src="{% static 'allimg/linkedin.jpeg' %}" alt="LinkedIn" id="img1"></a>
      </div> </div>
    <div class="footer-bottom"> &copy; 2025 RID Bharat | All rights reserved. </div>
  </footer> <script src="{% static 'js/script.js' %}"></script> </body> </html>
```

home.html

rid > templates > home.html > html > head > title

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Home | RID Bharat</title>
6   </head>
7   <body>
8       {% include "header.html" %}
9     <h1 style="margin: 30vh;">Welcome to RID Bharat - Home Page</h1>
10    {% include "footer.html" %}
11
12  </body>
13  </html>
```

templates
- about.html
- ebook.html
- footer.html
- gallery.html
- header.html
- home.html
- index.html
- news.html
- service.html

Reaming all this write like home page and include {% include "header.html" %} & {% include "footer.html" %}

Base.html in Django

- In Django, the base.html file is a **template base layout** that contains common HTML structure (like header, footer, and navigation) which is shared across multiple pages of a website.
- **Why base.html is used in Django:**
  - ✓ Acts as a **master template** for all pages.
  - ✓ Contains common layout (header, footer, nav).
  - ✓ Helps avoid **code repetition**.
  - ✓ Child templates use {% extends "base.html" %} and fill in custom content with {% block %}.
  - ✓ Keeps your project **clean**, **DRY**, and **easy to manage**.

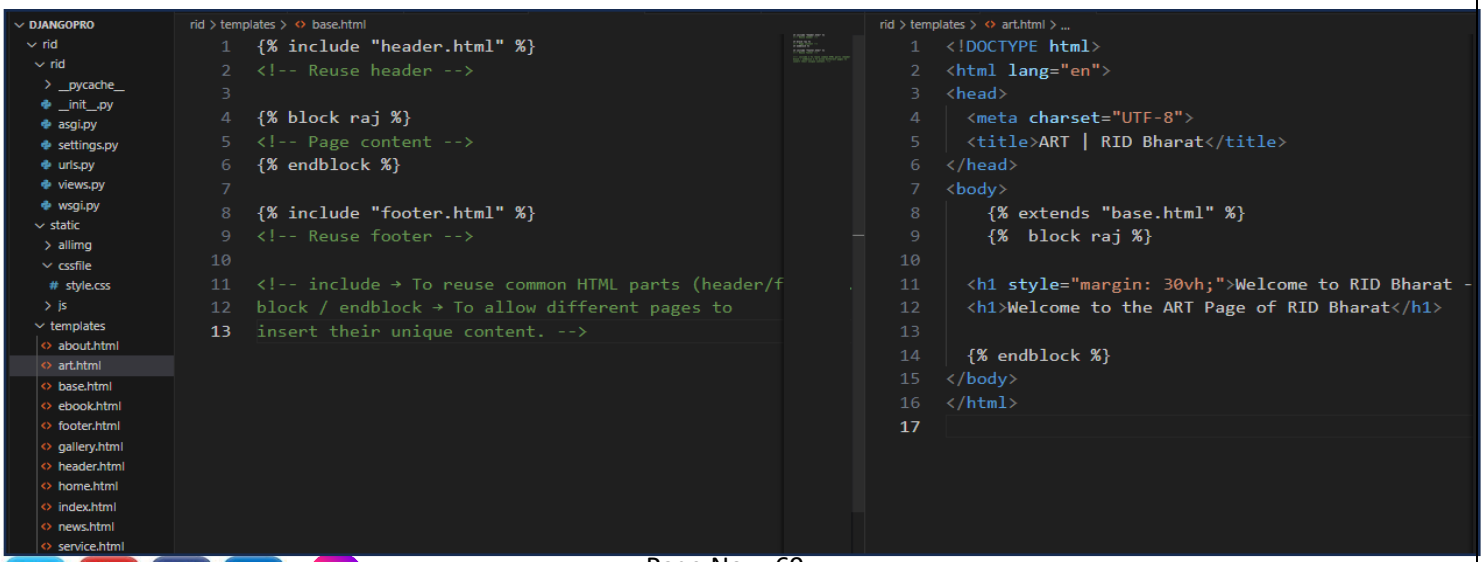
❖ **Conclusion:** base.html helps make your Django project more organized, DRY (Don't Repeat Yourself), and easier to maintain. It's a best practice for building scalable websites.

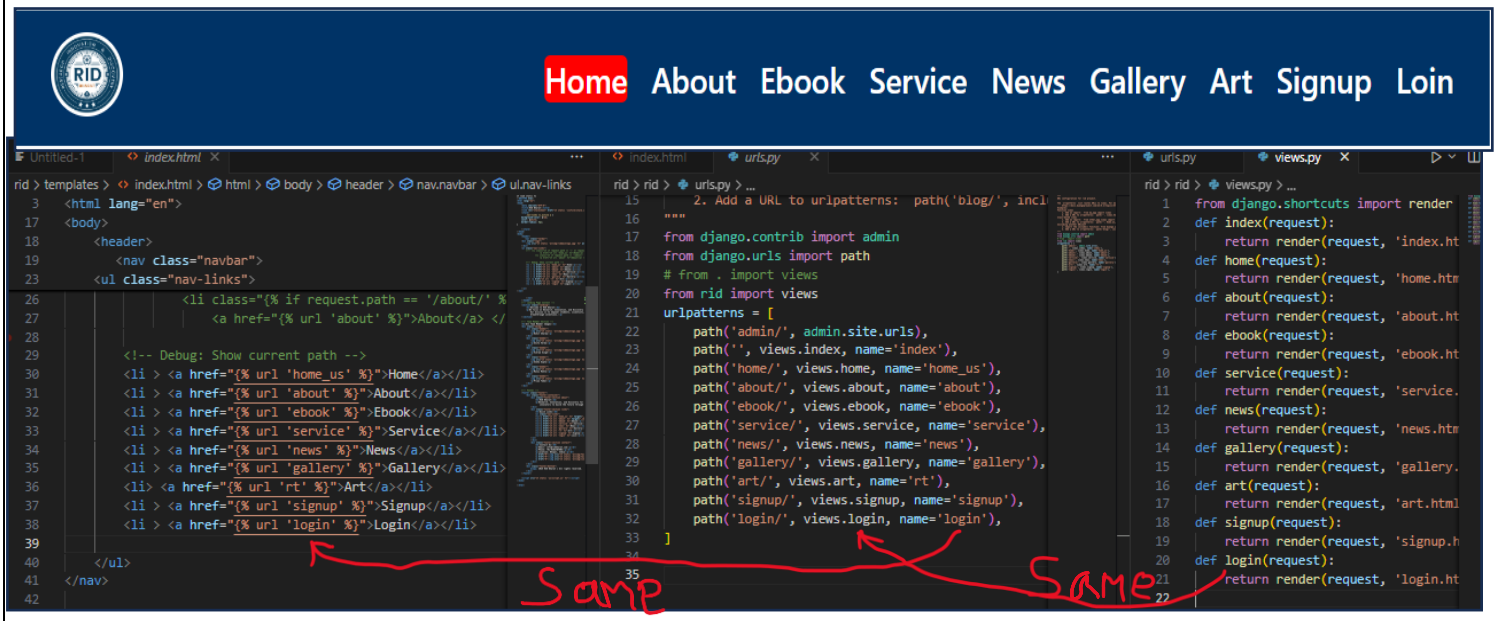
Example base.html

```
<html>
<head>
 <title>{% block title %}My Site{% endblock %}</title>
</head>
<body>
 {% include 'header.html' %}
 {% block content %}
 <!-- page-specific content goes here -->
 {% endblock %}
 {% include 'footer.html' %}
</body>
</html>
```

about.html

```
{% extends "base.html" %}
{% block title %}About Us{% endblock %}
{% block content %}
 <h1>About RID Bharat</h1>
 <p>We focus on research and innovation.</p>
{% endblock %}
```





signup data views.py

```
def signup(request): # Get data from form inputs
    if request.method == "POST":
        fullname = request.POST.get('fullname')
        email = request.POST.get('email')
        mobile = request.POST.get('mobile')
        password = request.POST.get('password')
    confirm_password = request.POST.get
        ('confirm_password')
        print("Full Name:", fullname)
        print("Email:", email)
        print("Mobile:", mobile)
        print("Password:", password)
        print("Confirm Password:",confirm_password)
        return render(request,
        'signup.html', {'message': 'Singup successfully!'})
```

Step: Save Signup Data in Local Storage

Step-1: Update your HTML form to add an ID

Ex: Sinup.html(Templates)

```
<form class="form-box" action="" method="post" id="signupForm">
  {% csrf_token %}
  <input type="text" name="fullname" placeholder="Full Name" >
  <input type="email" name="email" placeholder="Email" required>
  <input type="tel" name="mobile" placeholder="Mobile Number"
pattern="[0-9]{10}" title="Enter 10 digit mobile number" required>
  <input type="password" name="password" placeholder="Password" >
  <input type="password" name="confirm_password"
placeholder="Confirm Password" required>
  <button type="submit">Register</button>
  <p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
</form>
   <!-- JavaScript file to handle localStorage -->
   <script src="{% static 'js/signup.js' %}"></script>
```

2. Add JavaScript to save data into local storage Place this **before the closing </body> tag** in **signup.html:**

Static 'js/signup.js'

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.getElementById("signupForm");
  form.addEventListener("submit", () => {
    const data = {
      fullname: form.fullname.value,
      email: form.email.value,
      mobile: form.mobile.value,
      password: form.password.value,
      confirm_password: form.confirm_password.value,
    };
    localStorage.setItem("signupData", JSON.stringify(data));
  });
});
```

Note: You cannot directly access browser localStorage data from Django's views.py on the server side because:

- localStorage is a browser-side feature (JavaScript).
- Django views run on the server and receive HTTP requests.
- browser does **not** send localStorage data automatically with HTTP requests.

How to send localStorage data to your Django views?

You must **send the data from the client (browser) to the server** via an HTTP request, such as:

- Form inputs (hidden or visible fields),
- AJAX / Fetch requests (POST/GET)

Line-by-line Explanation above code

document.addEventListener("DOMContentLoaded", () => {

- ➔ DOMContentLoaded is a special event in JavaScript that fires when the **HTML document is fully loaded and parsed**, but **before images, stylesheets, and other resources** are finished loading.
- ➔ Runs the code **after the page is fully loaded**.

const form = document.getElementById("signupForm"); ➔ Gets the signup form using its **ID**.

form.addEventListener("submit", () => { ➔ When the form is **submitted**, run the function.

const **data** = { → object

fullname: form.fullname.value,
email: form.email.value,
mobile: form.mobile.value,
password: form.password.value,
confirm_password: form.confirm_password.value,
};

- ➔ Collects all form input values into an object called data.
  localStorage.setItem("signupData", JSON.stringify(data));
- ➔ Saves the data to **local storage** as a string.

Note: This script should be placed **just before </body>** in signup.html and in Static 'js/signup.js' file.

❖ **Step 2: JSON.stringify(data)**

- Converts the data object into a **string** format.
- Reason: localStorage **only stores strings**.
- **Result:**
  '{"fullname":"Sangam","email":"sangam@example.com","mobile":"1234567890","password":"abc123","confirm_password":"abc123"}'

➢ **localStorage stores** data in the browser permanently (until manually cleared).
➢ **setItem(key, value)** stores a value under a given key name.
➢ **JSON.stringify()** converts JavaScript objects to strings so they can be saved.
➢ **JSON.stringify()** is a JavaScript function that **converts an object into a JSON-formatted string**.

❖ Why use JSON.stringify()?

Because:

- **localStorage**, **sessionStorage**, and **APIs** can only store **strings**.
- JSON.stringify() turns objects into strings so they can be saved or sent.

What is JSON.stringify(data)?

It **converts a JavaScript object** into a **string format** (JSON) so it can be stored in places like localStorage.

◇ **Example:**

```
const data = {
  name: "Sangam Kumar",
  email: "sangam@example.com"
};
const jsonString = JSON.stringify(data);
console.log(jsonString);
```

- **Output:** {"name":"Sangam Kumar","email":"sangam@example.com"}

What you should do:

if you want to get that data from localStorage inside signup and login views.

1. **Add hidden input fields to your form in signup.html**
   Before form submission, **copy localStorage data into hidden fields**, so it gets submitted with form.
2. **Update your JavaScript** to set those hidden fields before form submission.
3. **In Django views, access the data via request.POST.get(...)** just like any form field.

Step: - get Signup Data from Local

rid > static > js > JS signup.js > ...

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.getElementById("signupForm");
  // On form submit, save visible inputs to localStorage
  form.addEventListener("submit", () => {
    const data = {
      fullname: form.fullname.value,
      email: form.email.value,
      mobile: form.mobile.value,
      password: form.password.value,
      confirm_password: form.confirm_password.value,
    };
    localStorage.setItem("signupData", JSON.stringify(data));
  });
  // Before form submission, copy localStorage data to hidden inputs
  form.addEventListener("submit", () => {
    const savedData = JSON.parse(localStorage.getItem("signupData")) || {};
    document.getElementById("ls_fullname").value = savedData.fullname || "";
    document.getElementById("ls_email").value = savedData.email || "";
    document.getElementById("ls_mobile").value = savedData.mobile || "";
    document.getElementById("ls_password").value = savedData.password || "";
  });});
```

// Before form submission, copy localStorage data to hidden inputs

form.addEventListener("submit", () => {

➔ When the form is about to be submitted, this code runs.

const savedData = JSON.parse(localStorage.getItem("signupData")) || {};

➔ Get saved signup data from **localStorage** and convert it back to an **object**. If no data exists, use an empty object.

document.getElementById("ls_fullname").value = savedData.fullname || "";

➔ Set the hidden input ls_fullname to the saved name. If not found, set it to empty.

document.getElementById("ls_email").value = savedData.email || "";

➔ Set ls_email input to saved email or empty.

document.getElementById("ls_mobile").value = savedData.mobile || "";

➔ Set ls_mobile input to saved mobile number or empty.

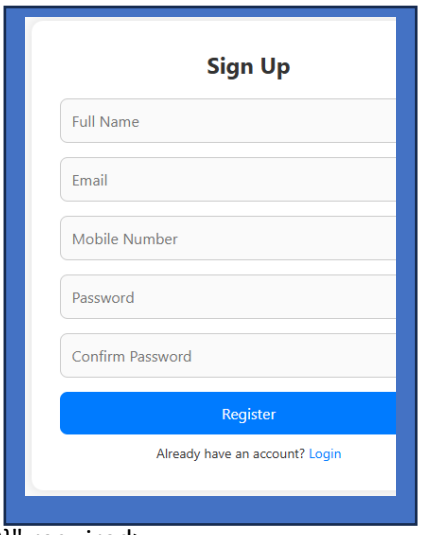
document.getElementById("ls_password").value = savedData.password || "";

➔ Set ls_password input to saved password or empty.

Summary:

This code **fills hidden inputs with saved localStorage data** before submitting form — so the server can receive that data.

Example for Signup.html (Templates)

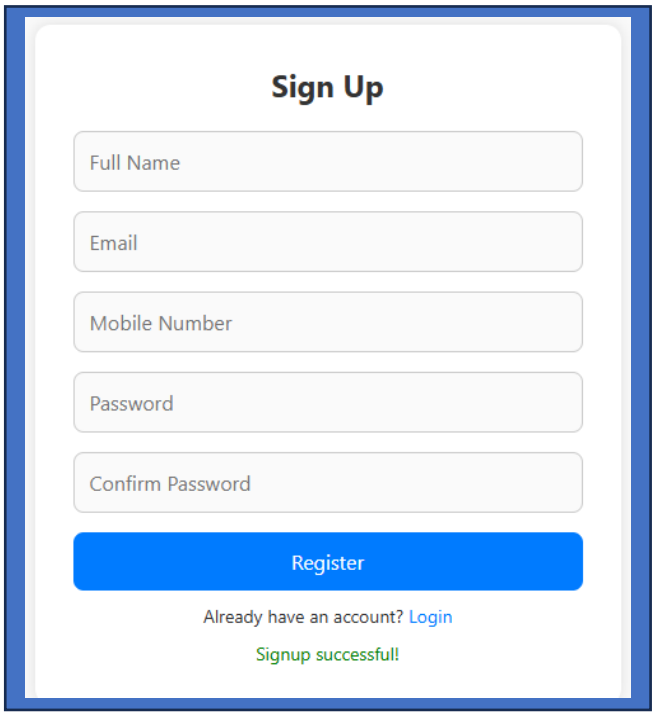



```
{% load static %}  {# Load Django static tag to use static files like CSS, JS, images e#}
<!DOCTYPE html> <html lang="en">
<head> <meta charset="UTF-8">
 <title>Sign Up</title>
 <link rel="stylesheet" href="{% static 'cssfile/sinuplogin.css' %}">
</head> <body>
  {% include "header.html" %}
 <div class="container">
  <form class="form-box" action="#" method="post" id="signupForm">
   {% csrf_token %}
   <h2>Sign Up</h2>
   <input type="text" name="fullname" placeholder="Full Name" required>
   <input type="email" name="email" placeholder="Email" required>
   <input type="tel" name="mobile" placeholder="Mobile Number" pattern="[0-9]{10}" required>
   <input type="password" name="password" placeholder="Password" required>
   <input type="password" name="confirm_password" placeholder="Confirm Password" required>
   <!-- Hidden fields for localStorage data -->
   <input type="hidden" name="ls_fullname" id="ls_fullname">
   <input type="hidden" name="ls_email" id="ls_email">
   <input type="hidden" name="ls_mobile" id="ls_mobile">
   <input type="hidden" name="ls_password" id="ls_password">
   <button type="submit">Register</button>
<p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
    {% if message %}
    <p style="color: green;">{{ message }}</p>
   {% endif %}
  </form>  </div>
 {% include "footer.html" %}
  <!-- JavaScript file to handle localStorage -->
  <script src="{% static 'js/signup.js' %}"></script>
</body> </html>
```

```
rid > templates > <> signup.html > html > body > div.contain
 1   {% load static %}  {# Load Django static tag to us
 2   <!DOCTYPE html>
 3   <html lang="en">
 4   <head> <meta charset="UTF-8">
 5     <title>Sign Up</title>
 6     <link rel="stylesheet" href="{% static 'cssfile/sinuplogin.css' %}">
 7   </head>
 8   <body>
 9     {% include "header.html" %}
10     <div class="container">
11       <form class="form-box" action="#" method="post" id="signupForm">
12         {% csrf_token %}
13         <h2>Sign Up</h2>
14         <input type="text" name="fullname" placeholder="Full Name" required>
15         <input type="email" name="email" placeholder="Email" required>
16         <input type="tel" name="mobile" placeholder="Mobile Number" pattern="[0-9]{10}" required>
17         <input type="password" name="password" placeholder="Password" required>
18         <input type="password" name="confirm_password" placeholder="Confirm Password" required>
19         <!-- Hidden fields for localStorage data -->
20         <input type="hidden" name="ls_fullname" id="ls_fullname">
21         <input type="hidden" name="ls_email" id="ls_email">
22         <input type="hidden" name="ls_mobile" id="ls_mobile">
23         <input type="hidden" name="ls_password" id="ls_password">
24         <button type="submit">Register</button>
25         <p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
26         {% if message %}
27         <p style="color: green;">{{ message }}</p>
28         {% endif %}
29       </form>
30     </div>
31     {% include "footer.html" %} <!-- JavaScript file to handle localStorage -->
32     <script src="{% static 'js/signup.js' %}"></script>
33   </body> </html>
```

Views.py

```
def signup(request):
    if request.method == "POST":
        # Get visible form inputs
        fullname = request.POST.get('fullname')
        email = request.POST.get('email')
        mobile = request.POST.get('mobile')
        password = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')
        # Get data sent from localStorage via hidden fields
        ls_fullname = request.POST.get('ls_fullname')
        ls_email = request.POST.get('ls_email')
        ls_mobile = request.POST.get('ls_mobile')
        ls_password = request.POST.get('ls_password')
        print("Visible Fullname:", fullname)
        print("LS Fullname:", ls_fullname)
        # Similarly for other fields...
        # Your validation, save to DB, etc.
        return render(request, 'signup.html', {'message': 'Signup successful!'})
    return render(request, 'signup.html')    return render(request, 'signup.html')
```




```
18  #sinup data
19  def signup(request):
20      if request.method == "POST":
21          # Get visible form inputs
22          fullname = request.POST.get('fullname')
23          email = request.POST.get('email')
24          mobile = request.POST.get('mobile')
25          password = request.POST.get('password')
26          confirm_password = request.POST.get('confirm_password')
27          # Get data sent from localStorage via hidden fields
28          ls_fullname = request.POST.get('ls_fullname')
29          ls_email = request.POST.get('ls_email')
30          ls_mobile = request.POST.get('ls_mobile')
31          ls_password = request.POST.get('ls_password')
32          print("Visible Fullname:", fullname)
33          print("LS Fullname:", ls_fullname)
34          # Similarly for other fields...
35          return render(request, 'signup.html', {'message': 'Signup successful!'})
36      return render(request, 'signup.html')
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[16/Jul/2025 12:49:22] "POST /signup/ HTTP/1.1" 200 3928
Visible Fullname: nand kumar
LS Fullname: nand kumar
```

Views.py

```
def login(request):
    if request.method == "POST":
        # Get visible form inputs
        email = request.POST.get('email')
        password = request.POST.get('password')
        # Get hidden localStorage values (optional)
        ls_email = request.POST.get('ls_email')
        ls_password = request.POST.get('ls_password')
        # Debug output
        print("Visible Email:", email)
        print("Visible Password:", password)
        print("LS Email:", ls_email)
        print("Local_Storage Password:", ls_password)
        context = {'message': 'Login successful!'}
        return render(request, 'login.html', context)
    # If GET request, just render the page without message
    return render(request, 'login.html')
```

rid > rid > views.py > ...

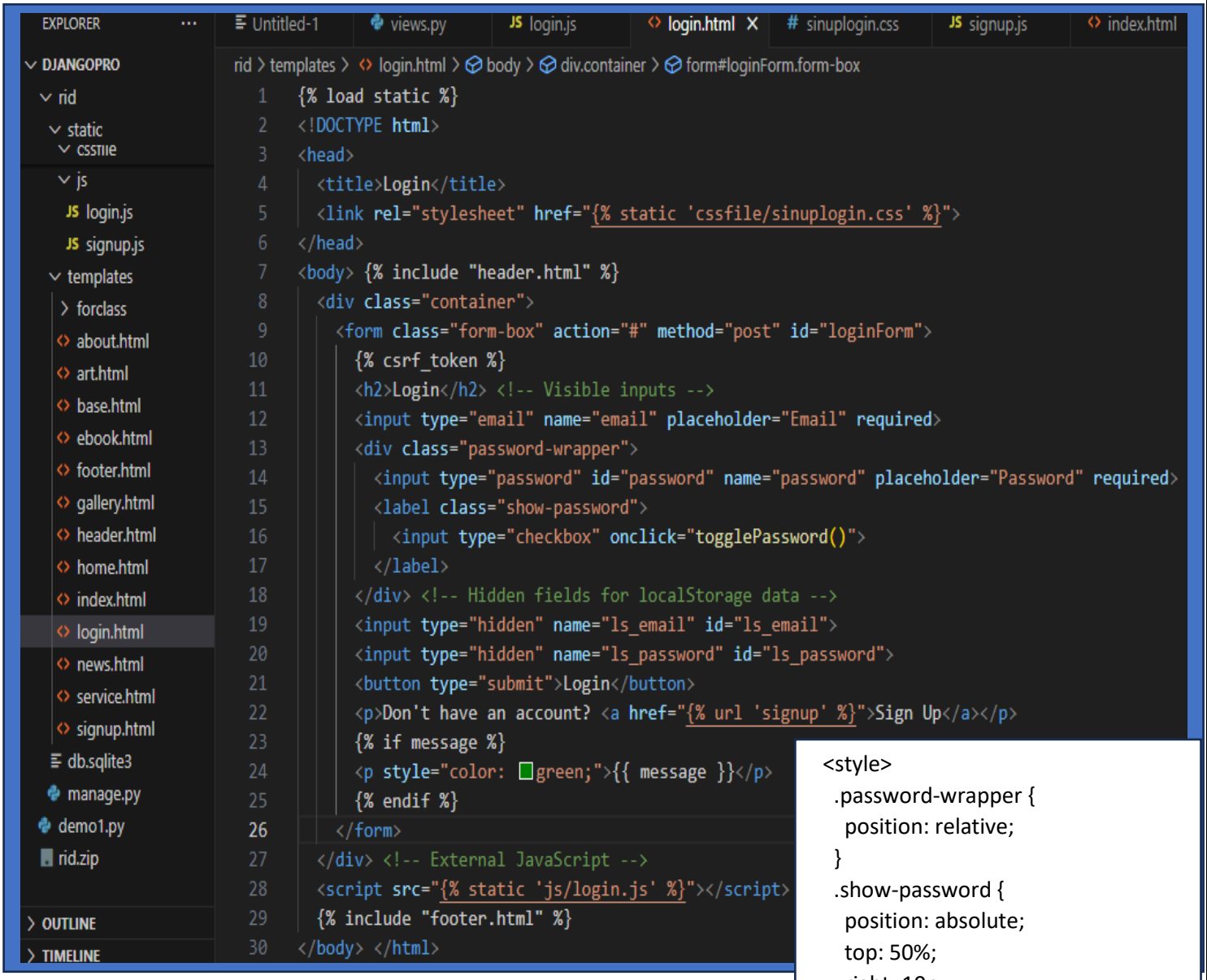
```
37  def login(request):
38      if request.method == "POST":
39          # Get visible form inputs
40          email = request.POST.get('email')
41          password = request.POST.get('password')
42          # Get hidden localStorage values (optional)
43          ls_email = request.POST.get('ls_email')
44          ls_password = request.POST.get('ls_password')
45          # Debug output
46          print("Visible Email:", email)
47          print("Visible Password:", password)
48          print("LS Email:", ls_email)
49          print("Local_Storage Password:", ls_password)
50          context = {'message': 'Login successful!'}
51          return render(request, 'login.html', context)
52      # If GET request, just render the page without message
53      return render(request, 'login.html')
```

DJANGOPRO

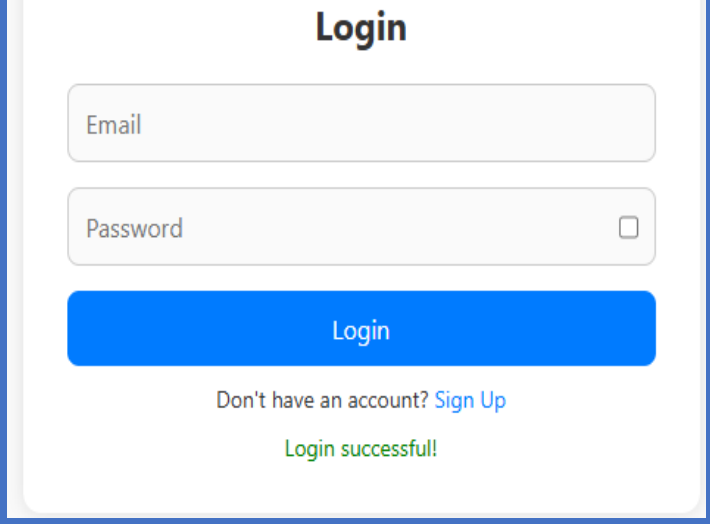
- rid
  - rid
    - asgi.py
    - settings.py
    - urls.py
    - views.py
    - wsgi.py
  - static
    - allimg
    - cssfile
      - sinuplogin.css
      - style.css
    - js
      - login.js
      - signup.js
  - templates
    - forclass
    - about.html
    - art.html
    - base.html

rid > static > js > login.js > document.addEventListener("DOMContentLoaded") callback

```
1   // 🔐 Toggle the visibility of the password field
2   // The togglePassword() function switches the input type between "password" (hidden)
3   // and "text" (visible) when the checkbox is clicked.
4   function togglePassword() {
5     const passwordField = document.getElementById("password");
6     passwordField.type = passwordField.type === "password" ? "text" : "password";
7   }
8   // This ensures the script runs only after the full HTML is loaded
9   document.addEventListener("DOMContentLoaded", () => {
10    const form = document.getElementById("loginForm");
11    // Before form submission, copy localStorage data to hidden fields
12    // This sends the localStorage data to the backend via POST
13    form.addEventListener("submit", () => {
14      const savedData = JSON.parse(localStorage.getItem("signupData")) || {};
15
16      // 📨 Populate hidden inputs with values from localStorage
17      document.getElementById("ls_email").value = savedData.email || "";
18      document.getElementById("ls_password").value = savedData.password || "";
19    });
20  });
21
```




```
<style>
 .password-wrapper {
  position: relative;
 }
 .show-password {
  position: absolute;
  top: 50%;
  right: 10px;
  transform: translateY(-50%);
  font-size: 13px;
 }
 .show-password input {
  margin-right: 4px;
 } </style>
```



Output

```
[16/Jul/2025 13:27:43] "GET /login/ HTTP/1.1" 200 39
Visible Email: raj@gmail.com
Visible Password: 123
LS Email: raj@gmail.com
Local_Storage Password: 123
```

After signup successful login page should be open

Steps to Redirect to Login Page After Signup

Step 1: Import redirect in views.py

- At the top of your views.py file:
  - from django.shortcuts import render, redirect

Step 2: Replace render(...) with redirect('login')

- In your signup view, after a successful POST (form submission), replace:
  - return render(request, 'signup.html', {'message': 'Signup successful!'})

with:

  - return redirect('login')

❖ **Updated signup view example:**

```
def signup(request):
    if request.method == "POST":
        # Process form data
        # ...
        # Redirect to login page after successful signup
        return redirect('login')
    return render(request, 'signup.html')
```

Example:

```
#sinup data
from django.shortcuts import render, redirect
def signup(request):
    if request.method == "POST":
        # Get visible form inputs
        fullname = request.POST.get('fullname')
        email = request.POST.get('email')
        mobile = request.POST.get('mobile')
        password = request.POST.get('password')
        confirm_password = request.POST.get('confirm_password')
        # Get data sent from localStorage via hidden fields
        ls_fullname = request.POST.get('ls_fullname')
        ls_email = request.POST.get('ls_email')
        ls_mobile = request.POST.get('ls_mobile')
        ls_password = request.POST.get('ls_password')
        print("Visible Fullname:", fullname)
        print("LS Fullname:", ls_fullname)

        # ✅ Redirect to login page after successful signup
        return redirect('login')
    return render(request, 'signup.html')
```

How to Redirect to any Page on Successful Login (Matching localStorage Data)

- To redirect to the **ebook** page when the login form's email and password **match the localStorage values** (sent via hidden fields), update your login view like this: **Steps to Redirect to ebook Page After Successful Login**

Step 1: Add Hidden Fields in login.html

- Add hidden fields to store localStorage values:
  <input type="hidden" name="ls_email" id="ls_email">
  <input type="hidden" name="ls_password" id="ls_password">
- Also, make sure your login.js sets these values before form submission.

Step 2: Make Sure login.js Copies localStorage Data

- **In login.js:**

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.getElementById("loginForm");
  form.addEventListener("submit", () => {
    const savedData = JSON.parse(localStorage.getItem("signupData")) || {};
    document.getElementById("ls_email").value = savedData.email || "";
    document.getElementById("ls_password").value = savedData.password || "";
  }); });
```

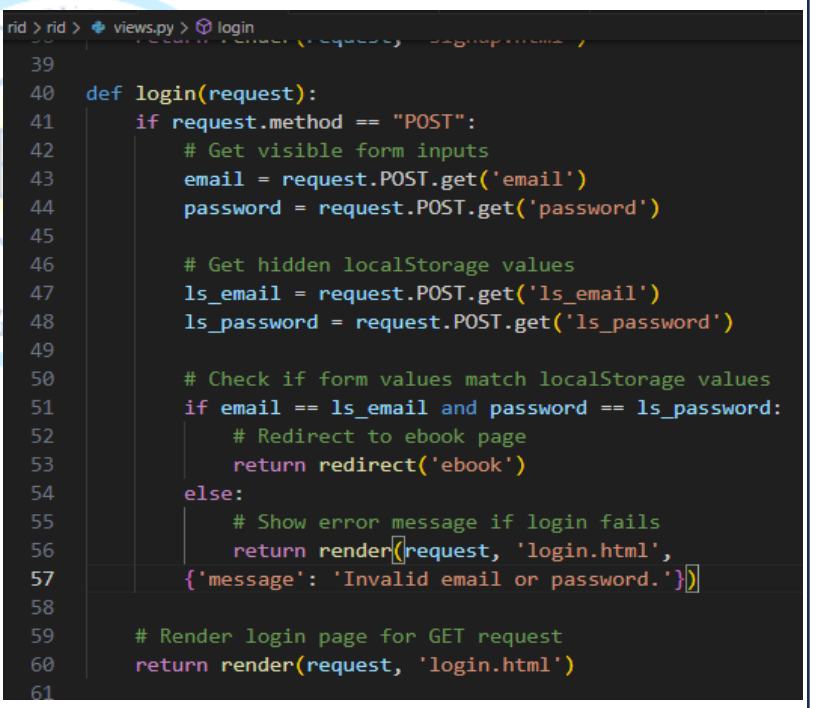
Step 3: Import redirect in views.py

- At the top of your views.py file:
  - from django.shortcuts import render, redirect

Step 4: Update the login View

- Replace your existing login view with this:

```
def login(request):
    if request.method == "POST":
        # Get visible form inputs
        email = request.POST.get('email')
        password = request.POST.get('password')
        # Get hidden localStorage values
        ls_email = request.POST.get('ls_email')
        ls_password = request.POST.get('ls_password')
        # Check if form values match localStorage values
        if email == ls_email and password == ls_password:
            # Redirect to ebook page
            return redirect('ebook')
        else: # Show error message if login fails
            return render(request, 'login.html',
            {'message': 'Invalid email or password.'})
    # Render login page for GET request
    return render(request, 'login.html')
```



Step 5: Define the ebook URL in urls.py

In your urls.py:

```
path('ebook/', views.ebook, name='ebook')
```

Make sure the ebook view is defined in views.py:

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

Final Result:

- User signs up → data stored in localStorage.
- User opens login page → localStorage values are copied to hidden fields.
- On login submit:
  - If visible email/password match the hidden localStorage values → ☑ **redirects to ebook.html**
  - If not → ✗ shows an **error message** on the login page.

How to Store Multiple Signup Entries in localStorage

Step 1: Update JavaScript to Store Array of Users, Step 2: Make Sure Hidden Fields Exist in signup.html

Create or update your signup.js to store **multiple users** in localStorage

Example:

```
<!-- Hidden fields to copy the last signed-up user into POST -->
<input type="hidden" name="ls_fullname" id="ls_fullname">
<input type="hidden" name="ls_email" id="ls_email">
<input type="hidden" name="ls_mobile" id="ls_mobile">
<input type="hidden" name="ls_password" id="ls_password">
```

```
document.addEventListener("DOMContentLoaded", () => {
 const form = document.getElementById("signupForm");
 form.addEventListener("submit", () => {
  const newUser = {
   fullname: form.fullname.value,
   email: form.email.value,
   mobile: form.mobile.value,
   password: form.password.value,
   confirm_password: form.confirm_password.value,};
  // Get existing users array from localStorage (or create empty array)
  let users = JSON.parse(localStorage.getItem("users")) || []
  users.push(newUser);                                    // Add the new user to the array
  localStorage.setItem("users", JSON.stringify(users));   // Save the updated array back to localStorage
  // Also store the latest user separately (for hidden field usage)
  localStorage.setItem("signupData", JSON.stringify(newUser))  }); });
```

Code Expiation

- document.addEventListener("DOMContentLoaded", () => {
  - ➔ **Waits for the full page to load before running the code.**
- const form = document.getElementById("signupForm") → **Gets the signup form element.**
- form.addEventListener("submit", () => { → **When the form is submitted, run this function.**
- const newUser = { ... } → **Collect form values into a newUser object.**
- let users = JSON.parse(localStorage.getItem("users")) || [];
  - ➔ Get the existing users array from localStorage, or create an empty one if none exists.
- users.push(newUser); → **Add the new user to the users array.**
- localStorage.setItem("users", JSON.stringify(users)); → **Save the updated users array back into localStorage.**
- localStorage.setItem("signupData", JSON.stringify(newUser));
  - ➔ Also save the latest user separately for use in hidden fields (e.g., login form).

What does this do?--> localStorage.setItem("signupData", JSON.stringify(newUser));
It saves the **latest signed-up user's data** separately in localStorage as "signupData".

Is it mandatory?

- **Yes, mandatory** if you're using hidden fields (ls_email, ls_password, etc.) and want to send this data to Django.
- **No, optional** if you're not using "signupData" or hidden fields.

Step 3: Update Hidden Fields with Latest Signup Data

Step 4: Keep Django signup View As-Is(no change anything)

:

```
form.addEventListener("submit", () => {
 const savedData = JSON.parse(localStorage.getItem("signupData")) || {};
 document.getElementById("ls_fullname").value = savedData.fullname || "";
 document.getElementById("ls_email").value = savedData.email || "";
 document.getElementById("ls_mobile").value = savedData.mobile || "";
 document.getElementById("ls_password").value = savedData.password || "";});
```

rid > static > js > JS signup.js > ...

```
1  form.addEventListener("submit", (e) => {
2    e.preventDefault(); // Stop immediate page reload
3    const newUser = {
4      fullname: form.fullname.value,
5      email: form.email.value,
6      mobile: form.mobile.value,
7      password: form.password.value,
8      confirm_password: form.confirm_password.value,
9    };// Get all existing users
10   let users = JSON.parse(localStorage.getItem("users")) || [];
11   // Add new user to array
12   users.push(newUser);
13   // Save back to localStorage
14   localStorage.setItem("users", JSON.stringify(users));
15   localStorage.setItem("signupData", JSON.stringify(newUser));
16   // Optional: Log to confirm
17   console.log("Users after adding:", JSON.parse(localStorage.getItem("users")));
18   // Submit form manually after storing
19   form.submit(); // Now let Django handle it
20 });
21 /* `form.submit()` is a **predefined JavaScript method** that **manually submits
22  the form to the server.
23 It's used after `e.preventDefault()` to:
24 * Save data (e.g., to localStorage)
25 * Then submit the form manually to Django/backend. */
```

How to forget or reset the Django admin

Step: -1 you should know your user's name

- **Note: -** if you don't know then open your database and see there

Step: -2: run this command in terminal

Syntax: python manage.py changepassword username

Example:

```
PS C:\Users\hp\OneDrive\Desktop\djangopro> cd demo3
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py changepassword raj
Changing password for user 'raj'
Password:
Password (again):
Password changed successfully for user 'raj'
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3>
```

MODEL IN DJANGO

- In **Django**, a **Model** is a Python class that represents a table in a database. It is used to define the **structure of your database** – the fields (columns) and their behavior. Django uses models to interact with the database through its built-in ORM (Object-Relational Mapper).
- **What is ORM (Object-Relational Mapper) in Django?**
  - **ORM (Object-Relational Mapper)** is a tool in Django that lets you **interact with the database using Python code instead of SQL**.

In Simple Words:

ORM allows you to:

- Create, read, update, and delete database records using **Python classes and objects**, not raw SQL queries.

Example:

Instead of writing SQL:

SELECT * FROM student;

You use Django ORM:

Student.objects.all()

To add a record:

Student.objects.create(name="Sangam", age=20)

Benefits of ORM:

- No need to write SQL manually.
- Code is **cleaner** and **database-independent**.
- Easier to maintain and read.
- Automatically protects against SQL injection.

Summary:

- **Django ORM** connects your **Python classes (models)** to **database tables**, so you can manage your database using just Python code.

❖ **SQL vs ORM (Django) – Short Comparison Table**

Action	SQL Query	Django ORM
Select all	SELECT * FROM student;	Student.objects.all()
Select by ID	SELECT * FROM student WHERE id=1;	Student.objects.get(id=1)
Insert record	INSERT INTO student (name, age) VALUES ('John', 20);	Student.objects.create(name='John', age=20)
Update record	UPDATE student SET age=21 WHERE id=1;	student.age = 21; student.save()
Delete record	DELETE FROM student WHERE id=1;	student.delete()

Step involve in models

1 Define model in models.py
2 Register in admin.py (optional)
3 Add app to INSTALLED_APPS
4 Run makemigrations and migrate
5 Use model in views or Django shell

Field Name list in Django

1. Text and String Fields

	Field Name	Description
A.	CharField()	:- Short text with max_length required
B.	TextField()	:- Long text, no length limit
C.	EmailField()	:- Validated email format, stored as string
D.	SlugField()	:- Short label, URL-friendly (slug)
E.	URLField()	:- Validates and stores a URL
F.	UUIDField()	:- Stores a UUID
G.	FilePathField()	:- Stores file paths from the filesystem

2. Numeric Fields

	Field Name	Description
A.	IntegerField()	:- Whole numbers
B.	BigIntegerField()	:- Large integers
C.	PositiveIntegerField()	:- Only positive numbers
D.	SmallIntegerField()	:- Smaller integers
E.	DecimalField()	:- Fixed-point decimals
F.	FloatField()	:- Floating-point numbers
G.	PositiveSmallIntegerField()	:- Small positive numbers

3. Date and Time Fields

Field Name	Description

es date + time
es a time duration
enting integer (used for IDs)

4. Boolean and Null Fields

	Field Name	Description
A.	BooleanField()	:- True/False
B.	NullBooleanField()	:- Deprecated; use BooleanField(null=True)

5. File and Media Fields

	Field Name	Description
A.	FileField()	:- Uploads and stores a file
B.	ImageField()	:- Uploads and stores an image

6. Relational Fields

	Field Name	Description
A.	ForeignKey()	:- Many-to-one relationship
B.	OneToOneField()	:- One-to-one relationship
C.	ManyToManyField()	:- Many-to-many relationship

7. Miscellaneous Fields

	Field Name	Description
A.	GenericIPAddressField()	:- Stores an IP address
B.	JSONField()	:- Stores JSON data (Django 3.1+)
C.	BinaryField()	:- Stores raw binary

Step-by-Step Explanation of a Model in Django

Step 1: What is a Model?

- A **model** is a Python class that inherits from django.db.models.Model.
- Each attribute in the class represents a **database field** (column).
- Django automatically generates SQL queries behind the scenes based on the model.

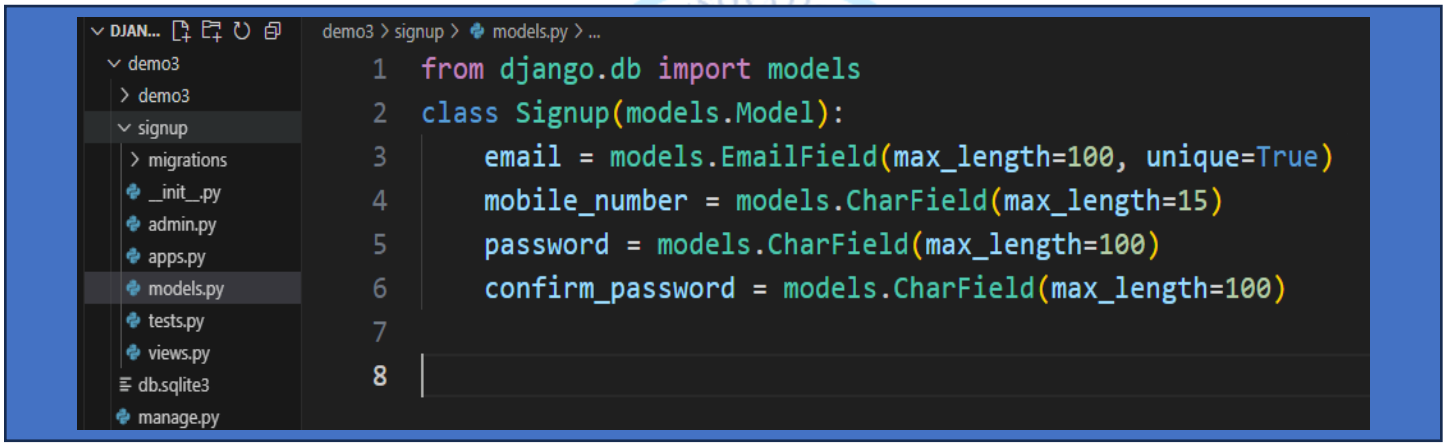
Step 2: Create a Model

- First, create a Django app):
- ➢ **Syntax:** python manage.py startapp model_name
- ➢ Example: python manage.py startapp signup
- ➢ it will create the separate folder with app name in project folder

Step-3: Create the class inside the mode.py that we want to design the database

- **Then open myapp/models.py and define your model:**
- **Inside model folder import model from Django.db**
- **Example: from django.db import models**
- **Jo model me field desing krenge vhi tables me table create hoga**

MODEL signup (model.py)



explanation

from django.db import models :- Imports Django's model system so you can define database tables.

class Signup(models.Model):- Defines a model (database table) named **Signup**.

email = models.EmailField(max_length=100, unique=True)

➢ Field to store **email addresses** (with format validation), max 100 characters, and must be **unique** (no duplicates).

mobile_number = models.CharField(max_length=15)

➢ Field to store **mobile numbers** as plain text, up to 15 characters.

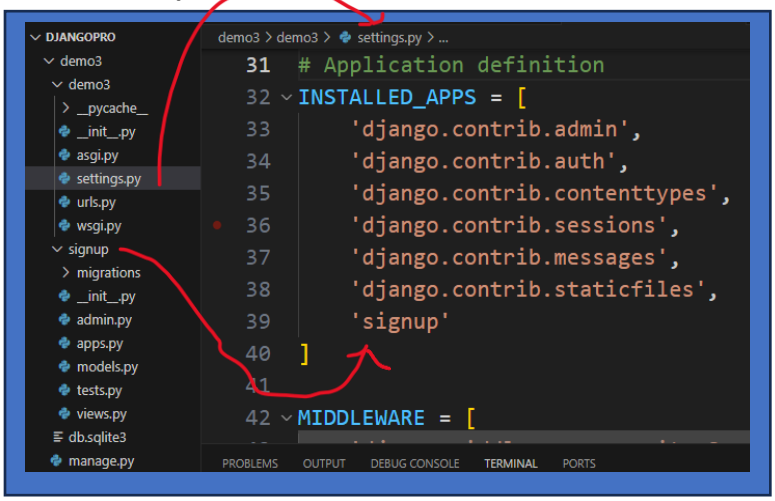
password = models.CharField(max_length=100)

confirm_password = models.CharField(max_length=100)

➢ These fields store the **password** and **confirm password** as text (usually both are the same during signup).

Step 4: add your model name inside setting.py INSTALLED_APPS

Example:



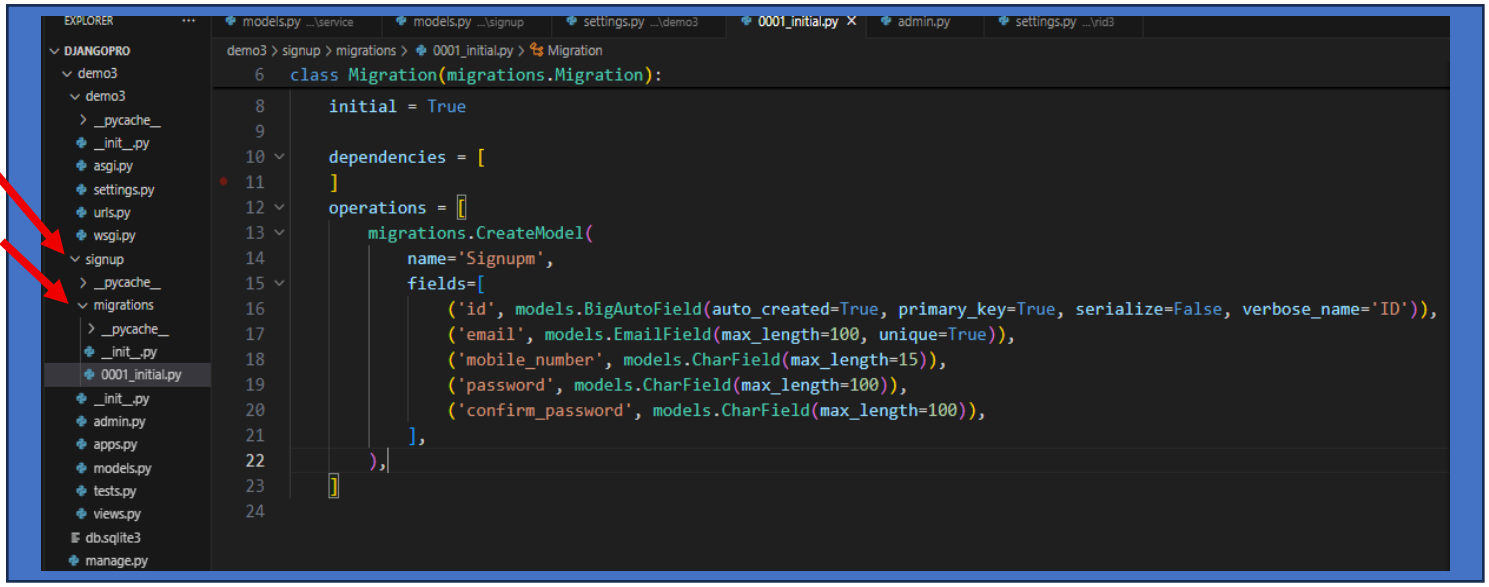
Step 5: Apply makemigrations in your Django project

Example: python manage.py makemigrations

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py startapp signup
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py makemigrations
Migrations for 'signup':
  signup\migrations\0001_initial.py
    + Create model Signupm
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3>
```

Why we use this (in short):

- makemigrations is used to **generate migration files** based on the changes you made in your models.py. These files tell Django how to **create or update database tables**. When anything in database applies this
- after run this command it will create the migration inside your models like in this case singup
- **Example: it create the model not table for creating apply migrate command Ex:**



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py startapp signup
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py makemigrations
Migrations for 'signup':
  signup\migrations\0001_initial.py
    + Create model Signupm
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, signup
Running migrations:
  Applying contenttypes.0001_initial.
```

Why we use migrate (in short):

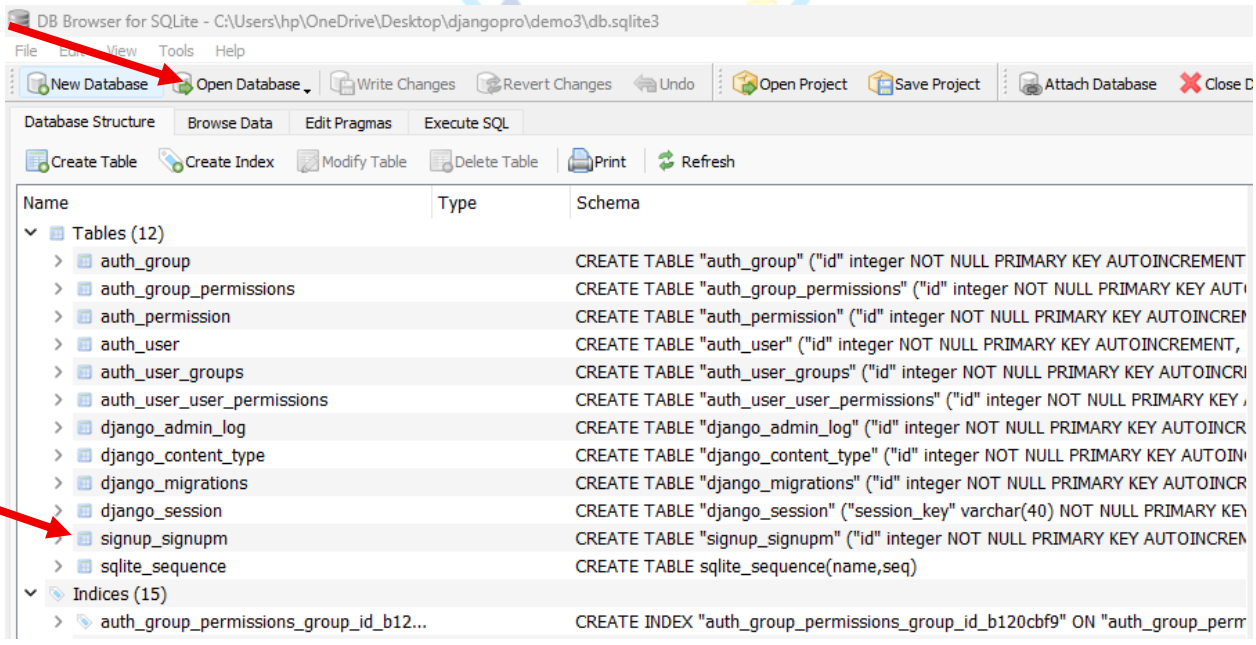
- The migrate command is used to **apply the changes defined in migration files** to the actual **database**. It creates or updates tables based on your models.

Step 6: How to view your table

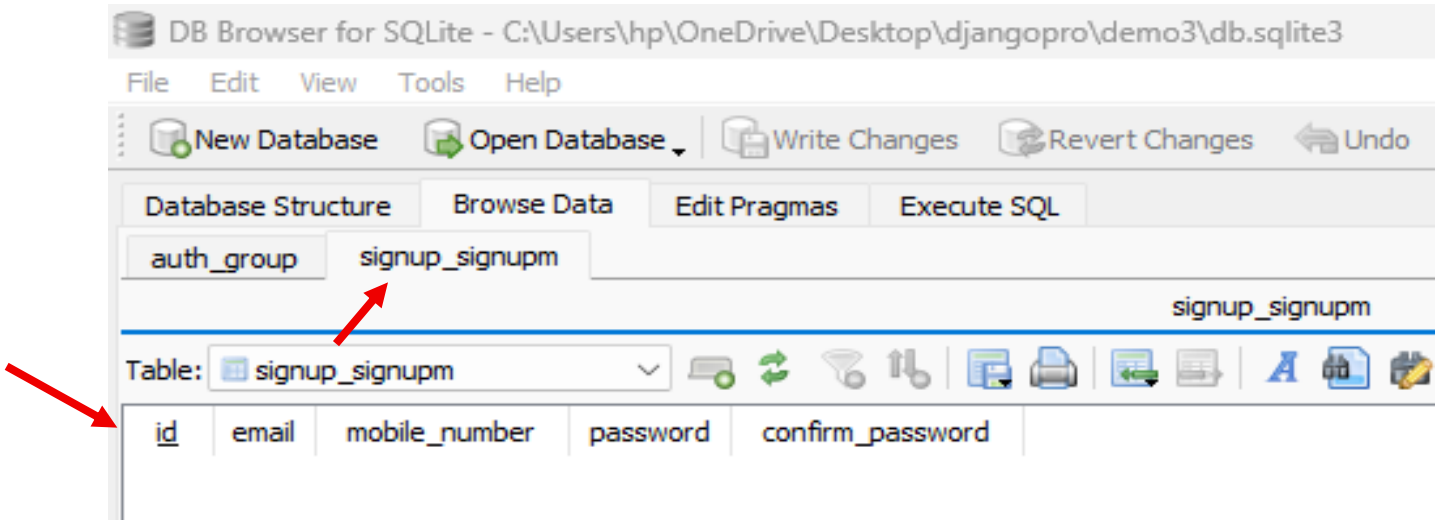
- Open the default Django database db.sqlite3 using DB Browser for SQLite, and then view the table for the singup model.

Steps:

1. Open **DB Browser for SQLite**.
2. Click **"Open Database"** and select your project's db.sqlite3 file.
3. Go to the **"Browse Data"** tab.
4. From the table dropdown, select **singup_singup** (format: appname_modelname).



Right click mouse and on the signup_signupm view the table data are in model



Steps to create a superuser:

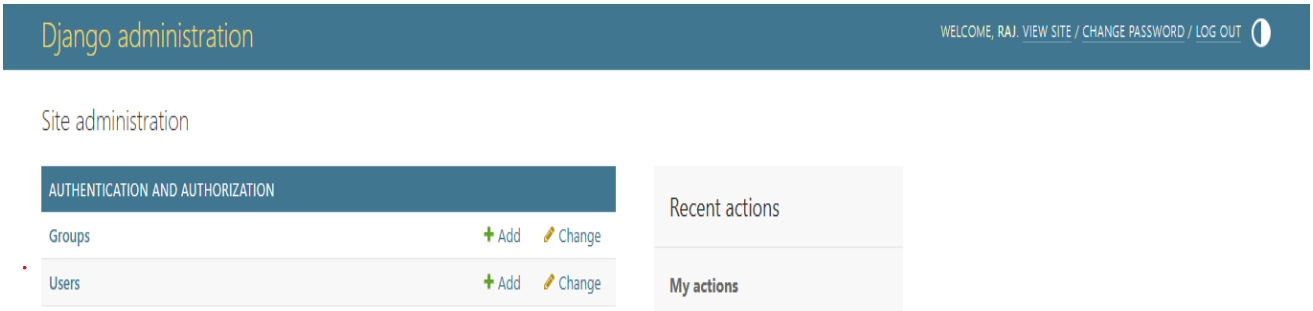
1. Open your terminal or command prompt.
2. Navigate to your Django project directory (where manage.py is located).
3. Run this command:

python manage.py createsuperuser

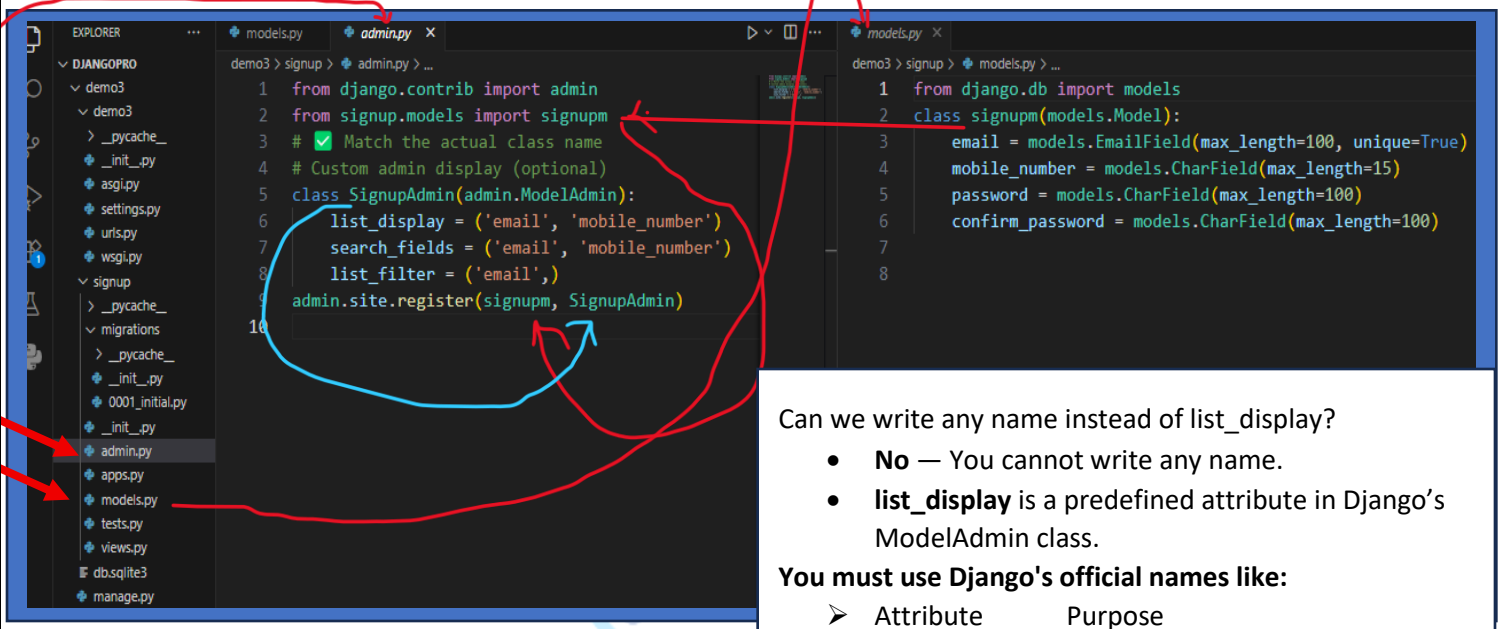
4. Follow the prompts to enter:
   - Username
   - Email address (optional)
   - Password (twice)

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> ls


    Directory: C:\Users\hp\OneDrive\Desktop\djangopro\demo3


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
dar--l         20-07-2025     08:39                demo3
dar--l         20-07-2025     09:07                signup
-a---l         20-07-2025     09:23         139264 db.sqlite3
-a---l         20-07-2025     08:37            683 manage.py


PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py createsuperuser
Username (leave blank to use 'hp'): raj
Email address: raj@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: n
Password:
Password (again):
Superuser created successfully.
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3>
```

After login we will see this



Step: -7: apply this in admin.py to see the model in admin



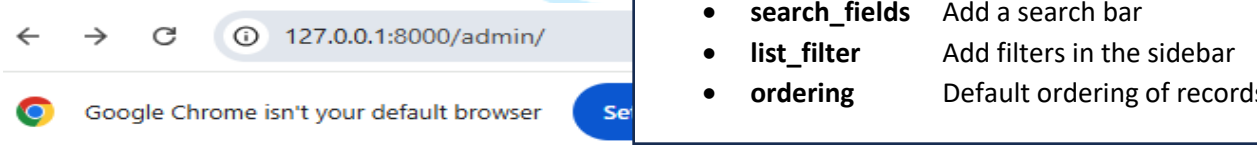
Can we write any name instead of list_display?

- **No** — You cannot write any name.
- **list_display** is a predefined attribute in Django's ModelAdmin class.

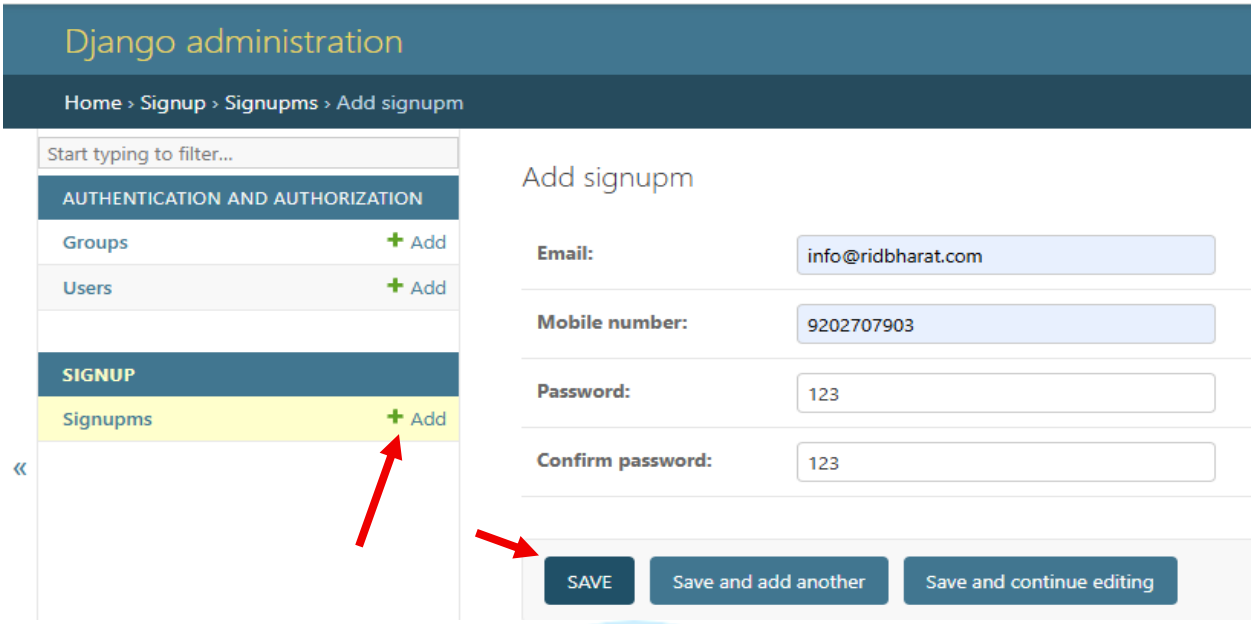
You must use Django's official names like:

Attribute	Purpose
list_display	Show specific fields in list view
search_fields	Add a search bar
list_filter	Add filters in the sidebar
ordering	Default ordering of records

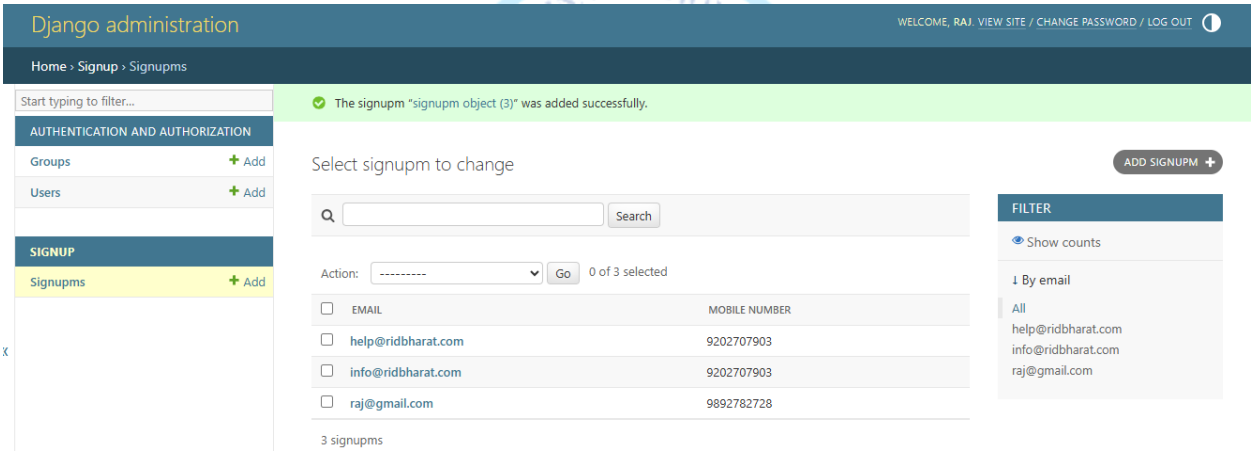
Open Brower and move to /admin



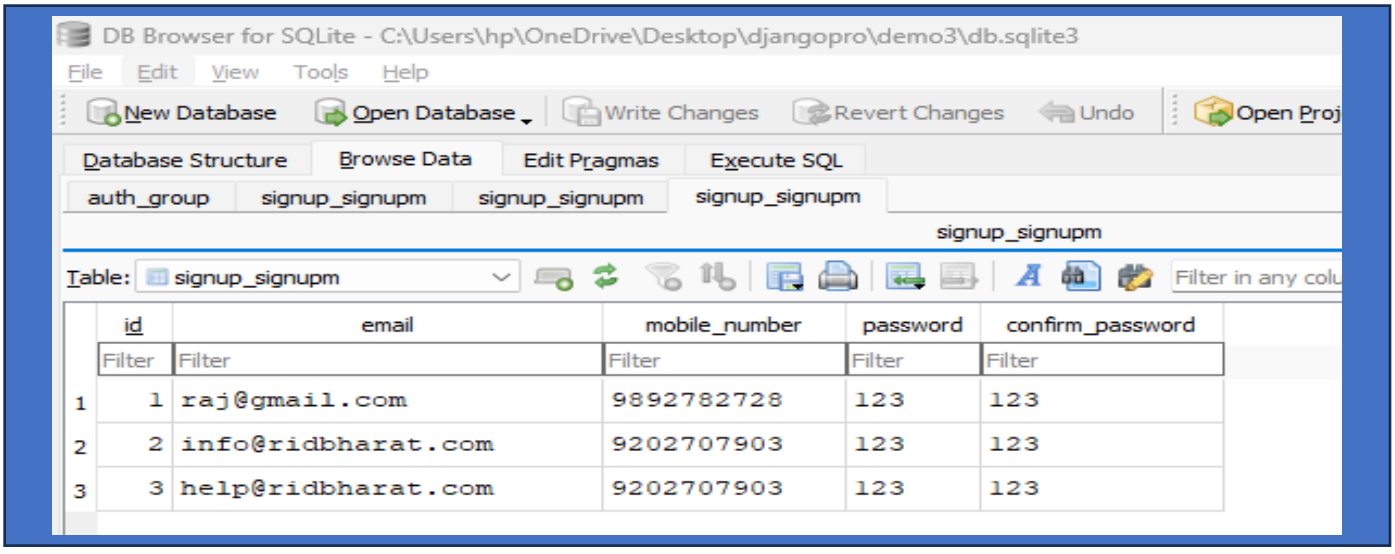
→click on Add button and add the data



→click the save button and then the show the data



→ Open db browser and see the data



How to Connect Front End, Backend & Database

Step-1: Open main project's views.py file, import the model, and write view code

Syntax: from models_name.models import class_name

Example: from signup.models import signupm

MVC vs MVT Model in Backend

Views.py

Backen End

```
EXPLORER
DJANGOPRO
demo3
  demo3
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    views.py
    wsgi.py
  signup
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  templates
    home.html
  db.sqlite3
  manage.py
```

```python
from django.shortcuts import render
from signup.models import signupm
def info(req):
    signupdata = signupm.objects.all()
    data = {
        "alldata": signupdata
    # This should match the template variable
    }
    return render(req, "home.html", data)
    # print(signupdata)
    # for i in signupdata:
    #     print(i.email)
    # return render (req, "home.html",data)

```

Models.py

Database

```python
from django.db import models
class signupm(models.Model):
    email = models.EmailField(max_length=100, unique=True)
    mobile_number = models.CharField(max_length=15)
    password = models.CharField(max_length=100)
    confirm_password = models.CharField(max_length=100)
```

admin.py

```python
from django.contrib import admin
from signup.models import signupm
# ✅ Match the actual class name
# Custom admin display (optional)
class SignupAdmin(admin.ModelAdmin):
    list_display = ('email', 'mobile_number')
    search_fields = ('email', 'mobile_number')
    list_filter = ('email',)
admin.site.register(signupm, SignupAdmin)
```

Step-2: Home.html(templates)

Front End

```
EXPLORER
DJANGOPRO
demo3
  demo3
    __pycache__
    __init__.py
    asgi.py
    settings.py
    urls.py
    views.py
    wsgi.py
  signup
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
  templates
    home.html
  db.sqlite3
  manage.py
new
raj
rid
rid3
demo1.py
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
    <style>
        .c1 {width: 300px;height: auto;
             border: 2px solid black;
             padding: 10px;margin: 10px;}
    </style>
</head>
<body>
    <h1>Welcome to the Home Page</h1>
    {% for i in alldata %}
    <div class="c1">
        <p><strong>Email ID:</strong> {{ i.email }}</p>
        <p><strong>Mobile No:</strong> {{ i.mobile_number }}</p>
    </div>
    {% endfor %}
</body></html>
```

Urls.py

```python
from django.urls import path
from demo3 import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("",views.info)
]
```

Output on Browser

Welcome to the Home Page

Email ID: raj@gmail.com
Mobile No: 9892782766

Email ID: info@ridbharat.com
Mobile No: 9202707903

Email ID: help@ridbharat.com
Mobile No: 9202707903

Email ID: mohan@gmail.com
Mobile No: 9202707903

```python
# Application definition
INSTALLED_APPS = [
    'signup',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

Add this two in Setting.py

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

How to access data from an SQLite database in Django:

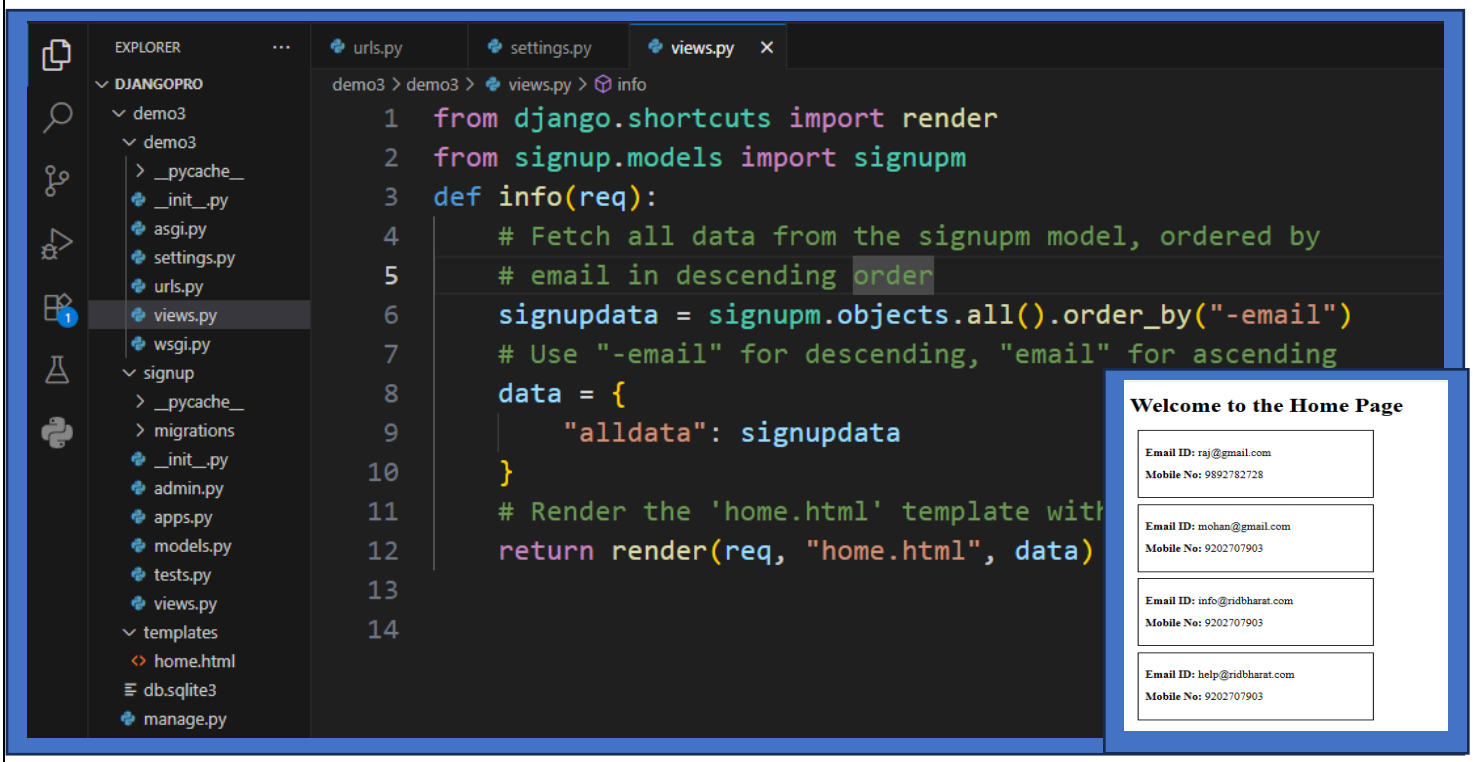
❖ **Steps to Access Data from SQLite Database in Django**:

1. Create a Django project and app
2. Configure the SQLite database in settings.py (default setup)
3. Define a model in models.py
4. Run makemigrations and migrate to create the table
5. Insert data using Django admin or shell
6. Import the model in views.py
7. Query the database using Django ORM (Model.objects.all(), etc.)
8. Pass the data to the template via context
9. Create an HTML template to display the data
10. Map the view to a URL in urls.py
11. Run the server and visit the URL in the browser

➢ **admin.site.register(...) in Django:**

- **admin**: Django's admin module.
- **site**: default admin interface. Or Default admin site instance
- **register()**: Adds your model to the admin so you can manage it via /admin.

How to Fetch all data from model, ordered by any filed in ascending or descending



Common Django ORM Methods to Access Data from database

1. Basic Query Methods

➢ Code	Description
A. **ModelName.objects.all()**	:- Get all records
B. **ModelName.objects.get(id=1)**	:- Get a single record (raises error if not found)
C. **ModelName.objects.first()**	:- Get the first record
D. **ModelName.objects.last()**	:- Get the last record
E. **ModelName.objects.count()**	:- Count total number of records
F. **ModelName.objects.exists()**	: -Returns True if records exist

2. Filter and Search Methods

➢ Code	Description
A. **Model.objects.filter(field=value)**	:- Filter by exact value
B. **Model.objects.exclude(field=value)**	:- Exclude matching records
C. **Model.objects.filter(field__contains='abc')**	:- Field contains a substring
D. **Model.objects.filter(field__startswith='abc')**	:- Field starts with substring
E. **Model.objects.filter(field__endswith='abc')**	:- Field ends with substring
F. **Model.objects.filter(field__iexact='abc')**	:- Case-insensitive exact match
G. **Model.objects.filter(field__in=[v1, v2])**	:- Match any value in a list
H. **Model.objects.filter(field__gt=10)**	:- Greater than
I. Model.objects.filter(field__lt=10)	:- Less than

3. Ordering and Limiting

➢ Code	Description
A. Model.objects.order_by('field')	:- Order ascending by field
B. Model.objects.order_by('-field')	:- Order descending by field
C. Model.objects.all()[:5]	:- Get first 5 records
D. Model.objects.all()[5:10]	:- Get records 6 to 10

4. Distinct and Values

➢ Code	Description
A. Model.objects.values('field1', 'field2')	:- Return list of dictionaries
B. Model.objects.values_list('field', flat=True)	:- Return list of values (flat=True for 1 field)
C. Model.objects.distinct()	:- Remove duplicates

5. Reverse Querying (Related Models)

Assuming a ForeignKey:

```
Child.objects.filter(parent__name='Sangam')   # Inside Child model
parent.child_set.all()                        # Inside Parent model
```

6. Q Objects for Complex Queries

```
from django.db.models import Q
Model.objects.filter(Q(email__startswith='a') | Q(mobile_number__endswith='9'))
```

7. F Expressions (Field-to-field comparison)

```
from django.db.models import F
Model.objects.filter(field1=F('field2'))
```

8. Accessing Fields in Views (Loop)

```
for i in Model.objects.all():
    print(i.email)
    print(i.mobile_number)
```

❖ **Example Usage in views.py**

```
from django.shortcuts import render
from signup.models import signupm
def user_list(request):
    data = signupm.objects.filter(email__contains='@gmail.com').order_by('-id')[:10]
    return render(request, 'users.html', {"data": data})
```

How to add new filed in exiting models

Step: -1 Update models.py Add the new field to your signupm model:

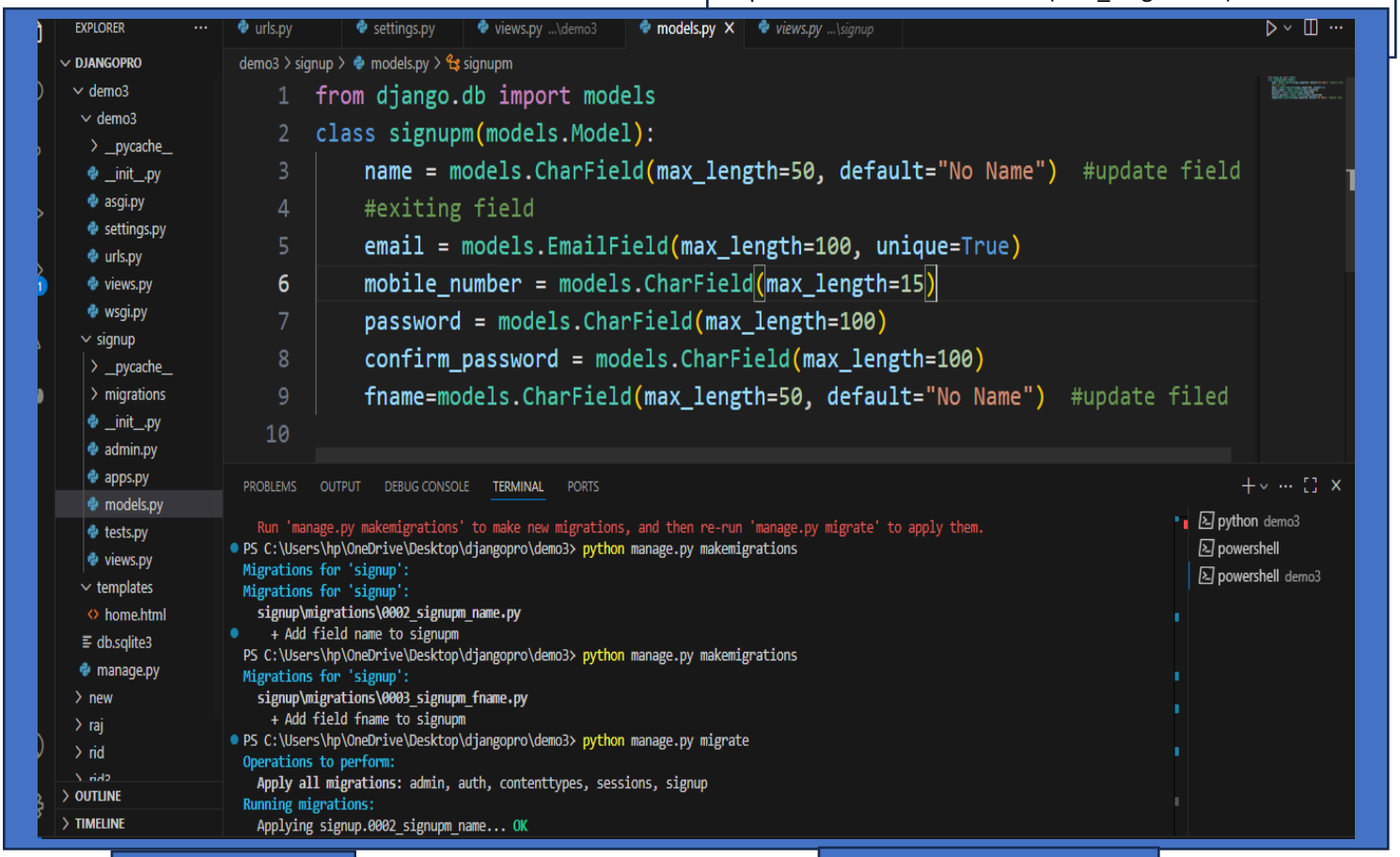
Option A (RECOMMENDED): Set a default value in models.py

Example: exiting models.py

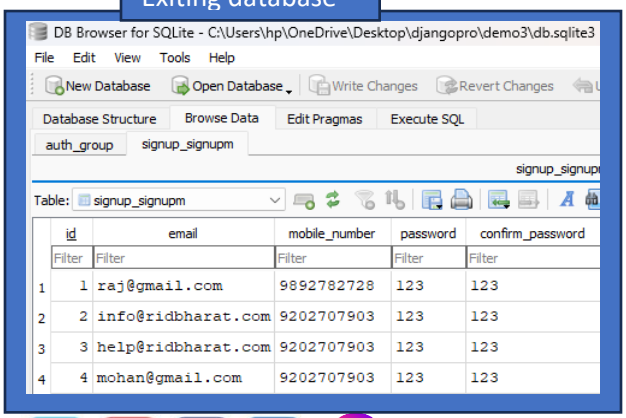
```
from django.db import models
class signupm(models.Model):
    email = models.EmailField(max_length=100)
    mobile_number = models.CharField(max_length=15)
    password = models.CharField(max_length=100)
    confirm_password = models.CharField(max_length=100)
```

updated models.py

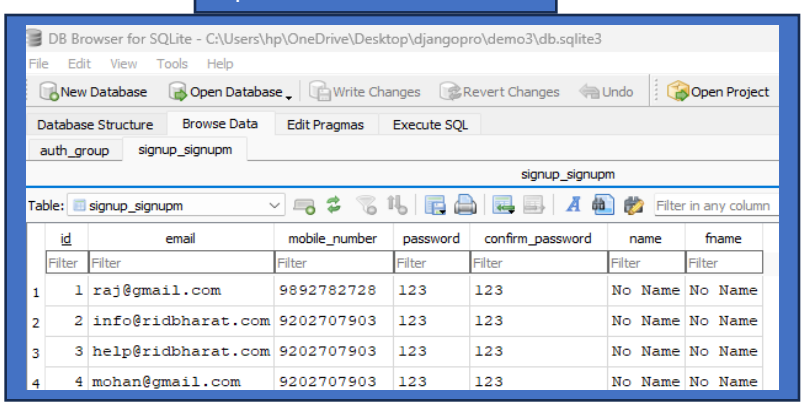
```
from django.db import models
class signupm(models.Model):
    #name=models.CharField(max_length=50) (Error )
    name = models.CharField(max_length=50, default="No Name")
    email = models.EmailField(max_length=100)
    mobile_number = models.CharField(max_length=15)
    password = models.CharField(max_length=100)
```



Exiting database



Updated database



How to change exit filed to new field

Step-by-Step: Update Field Type (e.g., CharField → TextField)

```
no3 > signup > models.py > ...
1  from django.db import models
2  class signupm(models.Model):
3      name = models.CharField(max_length=50, default="No Name")  # updated field
4      email = models.EmailField(max_length=100, unique=True)
5      mobile_number = models.CharField(max_length=15)
6      password = models.CharField(max_length=100)
7      confirm_password = models.CharField(max_length=100)
8      fname = models.CharField(max_length=50, default="No Name")  # updated field
9      # 🔽 Corrected & updated this line:
10     feedback = models.TextField(default="No Feedback")  # ✅ changed from CharField
11 # Change this:
12 # feedback = models.CharField(max_length=100, default="No Feedback")
13 # To this:
14 # feedback = models.TextField(default="No Feedback")
```

Then run:

- **python manage.py makemigrations**
- **python manage.py migrate**

- Can we change or update the field type (e.g., CharField → TextField) dynamically or conditionally without manually editing the migration files?"
- Then **Django's migration system is already dynamic** — it **auto-detects changes** in models.py and generates the correct SQL statements to update the database schema. Here's how you can dynamically manage or enhance this process:

2. Use null=True, blank=True for More Flexibility

- This makes the field optional in the database and forms:
- feedback = models.TextField(null=True, blank=True)
- This way, even if users leave it empty or you later change it, it won't break your app.

How to access the data from database to with different query

1. **Ordering Records**:
   - order_by("-email"): Sorts the records in **descending** order by the email field.
   - Use "email" (without -) to sort in **ascending** order.
2. **Limiting Results**:
   - [:3]: Applies a **limit** using Python's slice notation, equivalent to LIMIT 3 in SQL.
   - This means only the **top 3 records** (after ordering) are returned.
3. **Slicing Details**: Slicing in Django works just like Python lists. For example:
   - [0:2] → Returns the first 2 records.
   - [1:2] → Returns only the second record (index starts at 0).
   - **Important**: Django ORM **does not support negative indexing** (e.g., [-1]), unlike standard Python lists.
4. **Performance Consideration**:
   - Django translates this to SQL with a LIMIT clause, so it's efficient even with large datasets.
   - Always apply order_by() before slicing to ensure consistent results.
5. **Real-World Use**: Commonly used in dashboards, pagination, or when you only need the most recent or top-N entries.

Example:# Fetch the 3 latest users based on email (descending order)

```
signupdata = signupm.objects.all().order_by("-email")[:3]
```

How to access the data from database to with different query

Views.py

Models Name

```python
from django.shortcuts import render
from signup.models import signupm
def info(req):
    signupdata = signupm.objects.all().order_by("-email") #A. ordered by email in descending order
    signupdata = signupm.objects.all()[:2]                # B. Fetch the first 2 records using slicing (like SQL LIMIT 2)
    record_by_id = signupm.objects.get(id=1)              # C. Get the record with id = 1 (raises error if not found)
    first_record = signupm.objects.first()                # D. Get the first record (based on table order)
    last_record = signupm.objects.last()                  # E. Get the last record (based on table order)
    total_count = signupm.objects.count()                 # F. Count total number of records in the table
    has_records = signupm.objects.exists()                # G. Check if any record exists in the table
    data = {                                              # Prepare data dictionary to pass to template
        "alldata": signupdata,                            # First 2 records
        "record_by_id": record_by_id,                     # Record with id = 1
        "first": first_record,                            # First record
        "last": last_record,                              # Last record
        "count": total_count,                             # Total number of records
        "exists": has_records                             # True/False if data exists
    } # Render the template with the data
    return render(req, "home.html", data)
    # Fetch the latest 3 records from the signupm model, ordered by email in descending order.
    # Use "-email" for descending, "email" for ascending order.
    # [:3] applies a limit using Python's slice operator (like SQL LIMIT 3).
    # Note: Django ORM does not support negative indexing in slices.
    # signupdata = signupm.objects.all().order_by("-email")[:3]
    # signupdata = signupm.objects.all().order_by("-email")[1:2]
    #signupdata = signupm.objects.all()[:2]
    # Render the 'home.html' template and pass the data to it.
```

Result

Home.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home Page</title>
    <style>
        .c1 { width: 300px;height: auto;border: 2px solid black;
            padding: 10px;margin: 10px;}
    </style>
</head>
<body>
    <h1>Welcome to the Home Page</h1>
    <h2>First Record: {{ first.name }}</h2>
    <h2>Last Record: {{ last.name }}</h2>
    <h2>Record with ID 1: {{ record_by_id.name }}</h2>
    <h2>Total Count: {{ count }}</h2>
    <h2>Data Exists: {{ exists }}</h2>
    {% for i in alldata %}
    <div class="c1">
        <p><strong>Email ID:</strong> {{ i.email }}</p>
        <p><strong>Mobile No:</strong> {{ i.mobile_number }}</p>
    </div>
    {% endfor %}
</body></html>
```

Templates Filters in Django

- One of the most powerful features used to format and transform data inside Django templates.

Step 1: What Are Template Filters? Templates Filters in Django

- Template filters are used in **Django templates** to **modify variables** for display.
- They are written using the | (pipe) symbol.
- **Syntax:** {{ variable|filter_name }}
- **Example: {{ name|upper }}** If name = "rajesh", the output will be: RAJESH

Step 2: Basic Built-in Filters

❖ Text Transformation

Filter	Description	Example
upper	Converts to uppercase	{{ name | upper }}
lower	Converts to lowercase	{{ name | lower }}
title	Capitalizes each word	{{ name | title }}
capfirst	Capitalizes the first letter	{{ name | capfirst }}

❖ Handling Empty Values

Filter	Description	Example
default	Sets a fallback value if empty	{{ name | default:"Guest" }}

❖ String & List Operations

Filter	Description	Example
length	Returns length of string/list	{{ list | length }}
truncatechars	Cuts string after N characters	{{ text | truncatechars:50 }}
slice	Slices a list/string	{{ list | slice":3" }}

❖ Date Formatting

Filter	Description	Example
date	Formats a date object	{{ today | date:"Y-m-d" }}

❖ HTML & Line Breaks

Filter	Description	Example
safe	Renders HTML as markup	{{ html | safe }}
linebreaks	Converts \n to and <p>	{{ text | linebreaks }}
striptags	Removes HTML tags	{{ html | striptags }}

❖ Advanced Filters

Filter	Description	Example
filesizeformat	Formats file sizes (KB, MB, etc.)	{{ size | filesizeformat }}
pluralize	Adds "s" if quantity ≠ 1	{{ count | pluralize }}

Example: home.html

```
<h2>Uppercase: {{ name|upper }}</h2>
<h2>Lowercase: {{ name|lower }}</h2>
<h2>Title Case: {{ name|title }}</h2>
<h2>Length: {{ name|length }}</h2>
<h2>Default: {{ name|default:"Guest" }}</h2>
<h2>Formatted Date: {{ today|date:"F d, Y" }}</h2>
<h2>Truncated Message: {{ message|truncatechars:10 }}</h2>
<h2>Raw HTML: {{ html_content|safe }}</h2>
<h2>With Line Breaks: {{ content|linebreaks }}</h2>
```

Example

```
demo3 > templates > home.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Home Page</title>
6      <style>
7          .c1 {width: 300px;height: auto;
8              border: 2px solid black;
9              padding: 10px; margin: 10px;
10         }
11     </style>
12 </head>
13 <body>
14     <h1>Welcome to the Home Page</h1>
15     <h2>First Record: {{ first.name|upper }}</h2> <!-- Convert first record's name to uppercase using the 'upper' filte
16     <h2>Last Record: {{ last.name|upper }}</h2> <!-- Convert last record's name to uppercase -->
17     <h2>Record with ID 1: {{ record_by_id.name|upper }}</h2> <!-- Convert name of record with ID 1 to uppercase -->
18     <h2>Total Count: {{ count }}</h2> <!-- Show total number of records -->
19     <h2>Data Exists: {{ exists }}</h2><!-- Show True/False whether data exists -->
20     {% for i in alldata %}<!-- Loop through all records -->
21     <div class="c1">
22         <!-- Convert email to uppercase -->
23         <p><strong>Email ID:</strong> {{ i.email|upper }}</p>
24         <p><strong>Mobile No:</strong> {{ i.mobile_number }}</p>
25     </div>
26     {% endfor %}
27 </body>
28 </html>
```

How to add your own text editor in admin side for display contents on website

❖ django-tinymce

- **django-tinymce** is a Django application that contains a widget to render a form field as a TinyMCE editor.

Step-1:- install django-tinymce **Example: >** pip install django-tinymce

Step-2: - Add tinymce to INSTALLED_APPS in settings.py for your project:

Example: - INSTALLED_APPS = (
...
'tinymce',)

Step-3: Add tinymce.urls to urls.py for your project:

Example: from django.contrib import admin
from django.urls import path, include
urlpatterns = [
...
path('tinymce/', include('tinymce.urls')),
]

> ➢ **admin.site.register(...) in Django:**
> - **admin**: Django's admin module.
> - **site**: default admin interface. Or Default admin site instance
> - **register()**: Adds your model to the admin so you can manage it via /admin.

Step-4: Create the model

Example: - python manage.py startapp ridnews

Step-5: add model_name in INSTALLED_APPS in settings.py for your project.

Example: INSTALLED_APPS = (
...
'tinymce',
'ridnews')

Step-6: import the HTMLField from tinymce.models inside model and design the model

Example:- from tinymce.models import HTMLField

models.py

```
demo3 > ridnews > models.py > ...
1  from django.db import models
2  from tinymce.models import HTMLField
3  from django.utils import timezone  # ✅ Import timezone for current datetime
4  class Display_News(models.Model):
5      news_title = models.CharField(max_length=100)
6      date = models.DateField(default=timezone.now) # Automatically stores the current date when a record is created
7      current_day = models.DateTimeField(default=timezone.now) # Stores the current date and time (can be renamed to 'timestamp' or 'create
8      place = models.CharField(max_length=100)
9      news_desc = HTMLField()
10     def __str__(self):
11         return self.news_title
```

Step-6: write code in admin page for display the content in admin

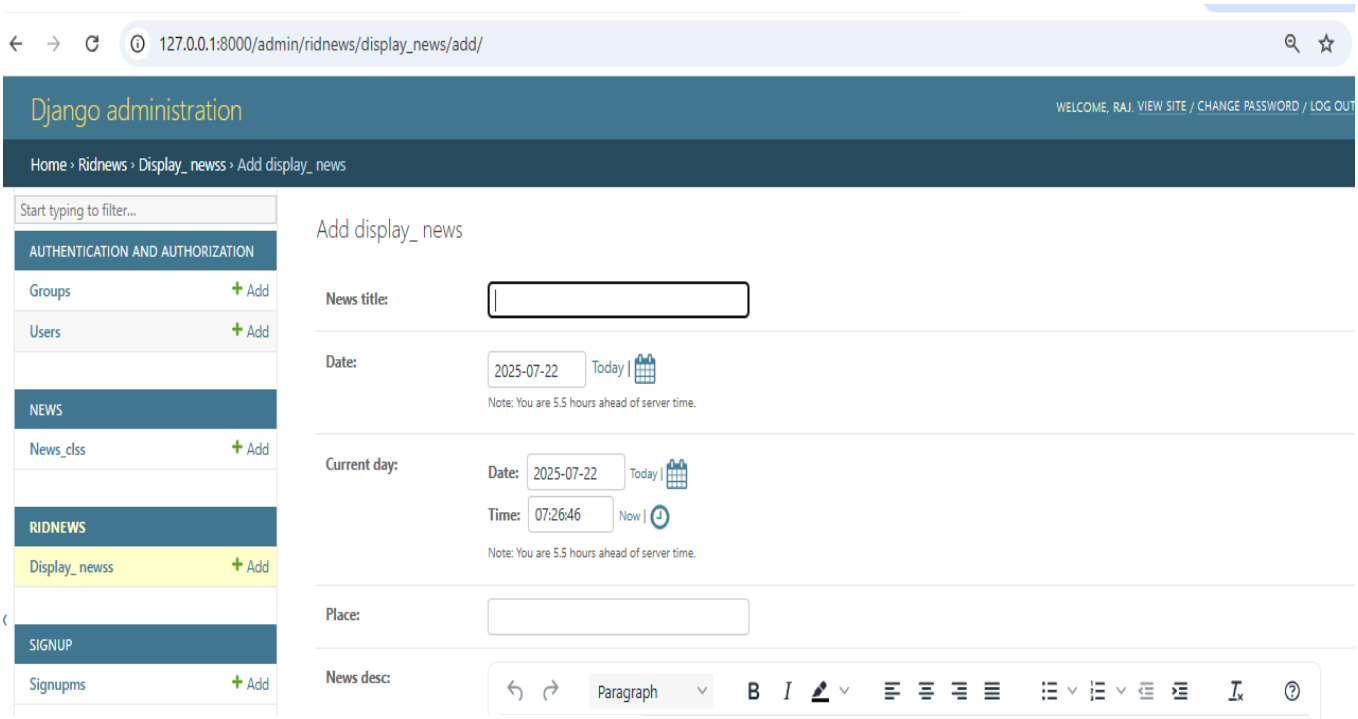
admin.py

```
demo3 > ridnews > admin.py > ...
1  from django.contrib import admin
2  from news.models import Display_News
3  class DisplayNewsAdmin(admin.ModelAdmin):
4      list_display = ('news_title', 'date', 'current_day', 'place')
5  admin.site.register(Display_News, DisplayNewsAdmin)
6
```

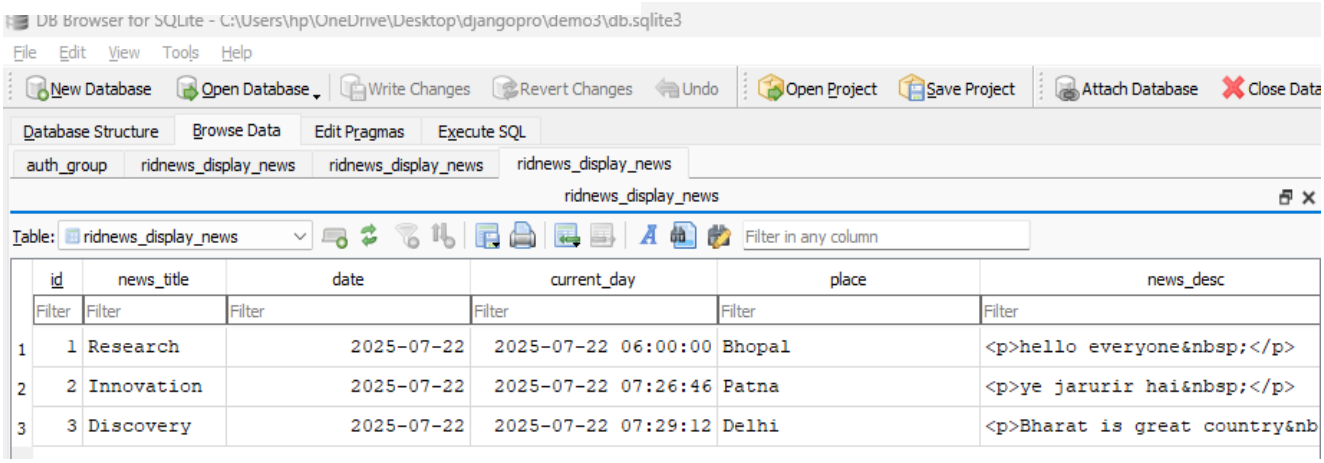
Step-7: run these two commands

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\hp\OneDrive\Desktop\djangopro> cd demo3
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py makemigrations
Migrations for 'ridnews':
  ridnews\migrations\0001_initial.py
    + Create model Display_News
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo3> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, news, ridnews, sessions, signup
Running migrations:
  Applying ridnews.0001_initial... OK
```

Step-8: visit admin and open ridnews model and fill the required data



Step-9: open data base show data



	id	news_title	date	current_day	place	news_desc
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Research	2025-07-22	2025-07-22 06:00:00	Bhopal	hello everyone
2	2	Innovation	2025-07-22	2025-07-22 07:26:46	Patna	ye jarurir hai
3	3	Discovery	2025-07-22	2025-07-22 07:29:12	Delhi	Bharat is great country&nb

Step-9: write code in project views.py to display the news and write ursl for news in urls.py

views.py

```python
from django.shortcuts import render
from ridnews.models import Display_News
def shownews(request):
    all_news = Display_News.objects.all().order_by('-date')
    # Get all news, latest first
    d={
        'news_list':all_news
        }
    return render(request, 'news.html',d)
```

urls.py

```python
"""
from django.contrib import admin
from django.urls import path, include  # ✅ incl
from demo3 import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('tinymce/', include('tinymce.urls')),
    path('info/', views.info),
    path("",views.shownews)
```

Step-10: write html code in news.html templates folder

news.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>News Page</title>
</head>
<body>
    <h1>Latest News</h1>
    {% for news in news_list %}
        <div style="border: 1px solid #ccc; padding: 15px; margin: 15px;">
            <h2>{{ news.news_title }}</h2>
            <p><strong>Date:</strong> {{ news.date }}</p>
            <p><strong>Day & Time:</strong> {{ news.current_day }}</p>
            <p><strong>Place:</strong> {{ news.place }}</p>
            <div>
                <strong>Description:</strong><br>
                {{ news.news_desc|safe }}
            </div>
        </div>
    {% empty %}
        <p>No news items available.</p>
    {% endfor %}
</body>
```

Step-11: visit on http://127.0.0.1:8000/ browser and see the result

Research

Date: July 22, 2025

Day & Time: July 22, 2025, 6 a.m.

Place: Bhopal

Description:

hello everyone

Innovation

Date: July 22, 2025

Template Tag Notes:

- {{ news.date }} → shows only the date.
- {{ news.current_day }} → shows full date and time.
- {{ news.news_desc|safe }} → renders HTML content from TinyMCE safely.

Django commands Name list

1. Project & App Setup

- django-admin startproject <projectname> # Create new Django project
- python manage.py startapp <appname> # Create new app within project
- python manage.py runserver # Start development server (default: 8000)
- python manage.py runserver 8080 # Run on a custom port (e.g., 8080)

2. Database & Migrations

- python manage.py makemigrations # Detect model changes and create migrations
- python manage.py migrate # Apply all pending migrations
- python manage.py migrate <appname> # Migrate a specific app
- python manage.py showmigrations # List all migrations (applied/unapplied)
- python manage.py sqlmigrate <app> <0001> # Show SQL for a specific migration

3. Admin & Authentication

- python manage.py createsuperuser # Create admin user for /admin
- python manage.py changepassword <user> # Change a user's password
- python manage.py shell # Django-aware Python shell
- python manage.py shell_plus # Enhanced shell (django-extensions)

4. Testing & Debugging

- python manage.py test # Run all tests
- python manage.py test <appname> # Test a specific app
- python manage.py check --deploy # Check production-ready settings
- python manage.py dbshell # Open database CLI (SQLite/PostgreSQL)

5. Static Files & Production

- python manage.py collectstatic # Gather static files for production
- python manage.py findstatic <file> # Locate static files

6. Third-Party Extensions (django-extensions)

- python manage.py graph_models -a > models.dot # Generate DB schema diagram
- python manage.py show_urls # Display all URL patterns
- python manage.py runscript <script> # Execute custom scripts

7. Utility Commands

- python manage.py help # List all available commands
- python manage.py version # Check Django version
- pip freeze > requirements.txt # Save dependencies (for deployment)

8. Reset & Maintenance

- python manage.py reset_db # Wipe and recreate DB (django-extensions)
- python manage.py clear_cache # Clear cache

Django imports list that is use during project development

1. Core Imports (Used in Almost Every Project)

- from django.shortcuts import render, redirect # Render templates / redirect URLs
- from django.http import HttpResponse, JsonResponse # Basic HTTP responses
- from django.urls import path, include # URL routing
- from django.db import models # Database models
- from django.contrib import admin # Admin panel

2. Authentication & Users

- from django.contrib.auth.models import User, Group # Built-in User model
- from django.contrib.auth import authenticate, login, logout # Auth functions
- from django.contrib.auth.forms import UserCreationForm # User registration

3. Forms

- from django import forms # Form base class
- from django.forms import ModelForm # Model-bound forms
- from django.core.exceptions import ValidationError # Form validation

4. Time & Utilities

- from django.utils import timezone # Timezone-aware datetimes
- from django.contrib import messages # Flash messages (alerts)

5. Development & Debugging

- from django.conf import settings # Access project settings
- from django.conf.urls.static import static # Serve media files in dev
- from django.core.management.base import BaseCommand # Custom commands

6. Advanced (Common in Real Projects)

- from django.db.models import Q, F # Complex DB queries
- from django.db.models.signals import post_save # Model signals
- from django.dispatch import receiver # Signal handlers
- from django.core.cache import cache # Caching
- from django.core.mail import send_mail # Email sending

Note: render() is used in 90% of views (replaces `HttpResponse` + templates).

Important Methods Name list in Django

1. Core View Methods

- **render(request, 'template.html', context={}) :-** Renders a template with context (used in 90% of views)
- **redirect('url-name-or-path') :-**Redirects to another URL (for POST-redirect-GET pattern)
- **HttpResponse('text') / JsonResponse({'data': 1}) :-** Returns basic HTTP/JSON responses

2. Database Methods

- **Model.objects.all() :-** Gets all records (common in list views)
- **Model.objects.get(id=1) :-**Gets single record (throws DoesNotExist if missing)
- **Model.objects.filter(condition).first()** :-Safe lookup (returns None if not found)
- **Model.save()** :-Saves model instance (insert/update)

3. Auth Methods

- **login(request, user) / logout(request**) :- Handles user sessions
- **authenticate(username='x', password='y') :-** Verifies credentials (returns user or None)

4. Form Methods

- **form.is_valid():-** Validates form data (required before saving)
- **form.save() :-** Saves ModelForm data to database
- **form.errors** :- Displays validation errors in templates

5. Utility Methods

- **messages.success(request, 'Done!')** :- Shows one-time notifications
- **timezone.now()** :- Gets current time (timezone-aware)
- reverse('url-name') :- Converts URL name to absolute path

Note:

- These 15 methods cover 80% of Django development.
- Master render(), filter(), is_valid(), and @login_required first - they're in nearly every project.

How does filter work in Django

Step-1: Create a New Django App

- django-admin startproject studentproject
- cd studentproject
- python manage.py startapp **collgedat**a

Step-2: Add model name ' collgedata ' to INSTALLED_APPS in settings.py.

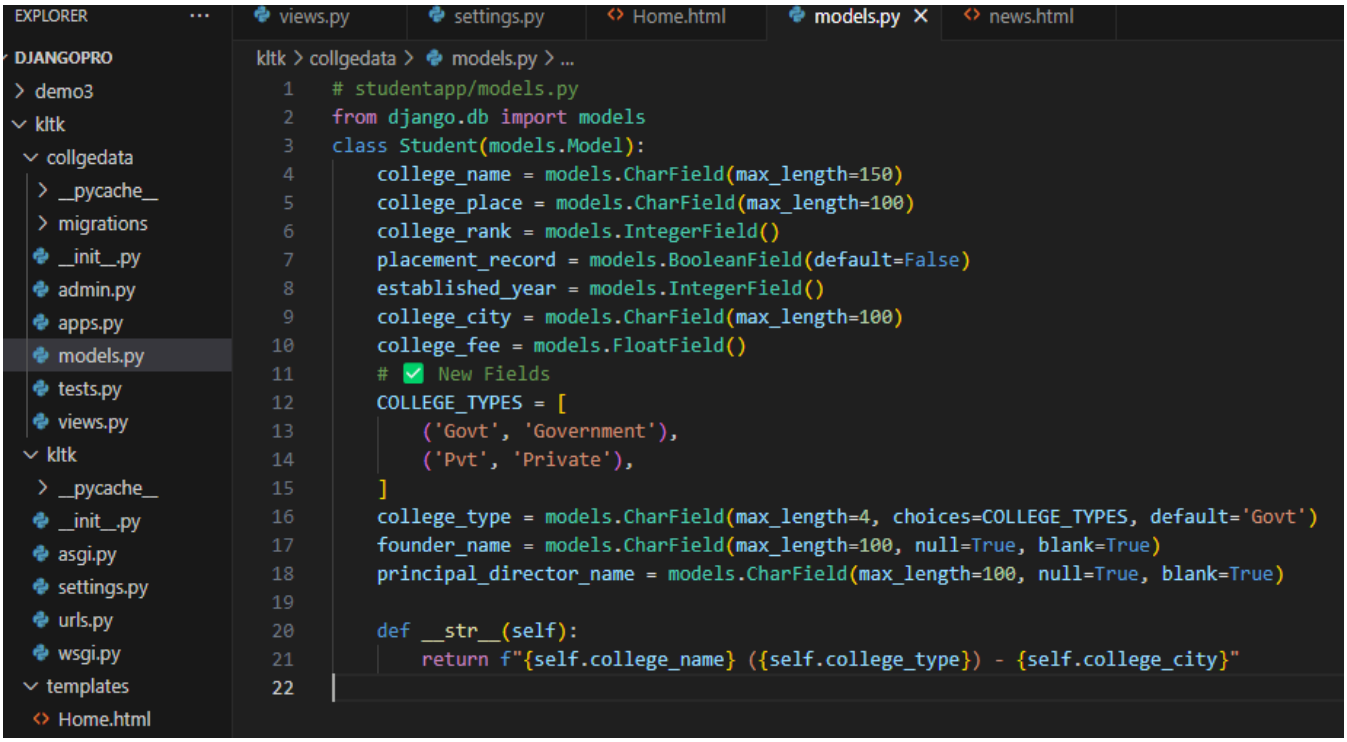
Example: INSTALLED_APPS = [

.....

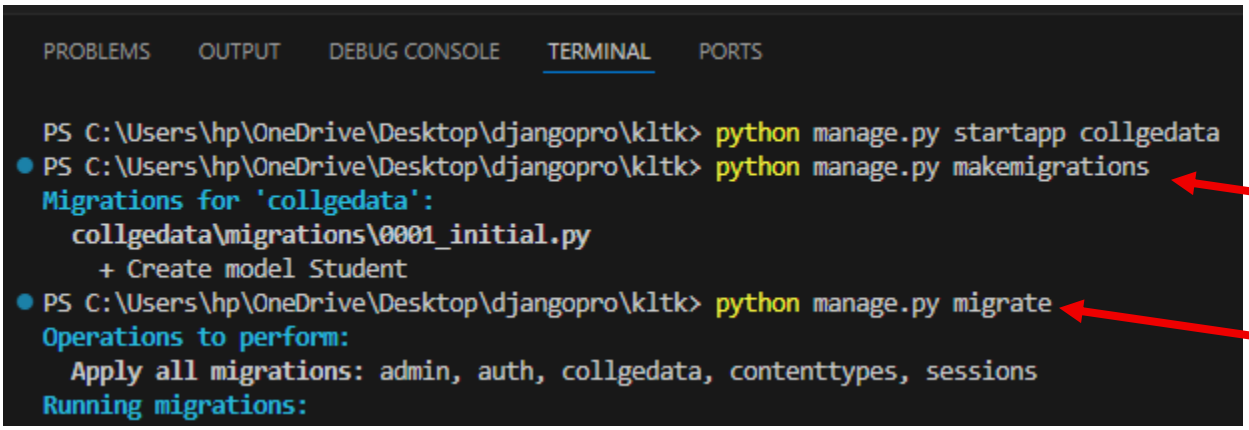
'collgedata'

]

Step-3: Define the Model according to your project



Step-4: run this two command makemigration and migrate



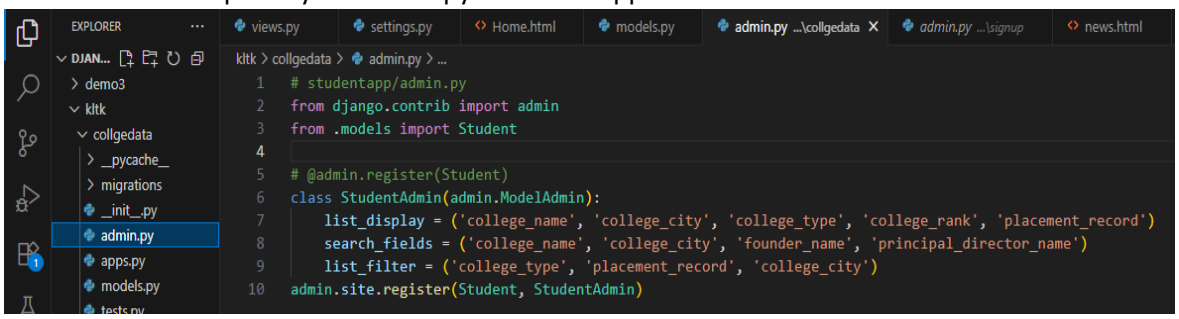
Step-4: Add Some Data in database in these two ways

- You can add sample data using the **admin panel** or **Django shell**:

1. Using Django Admin Panel

Step 1: Register your model in admin

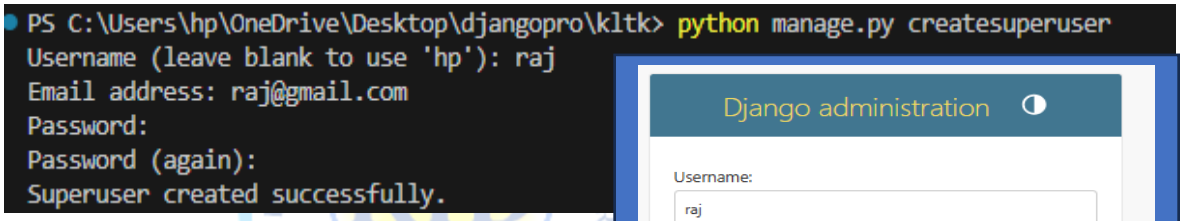
Create or update your admin.py in studentapp:

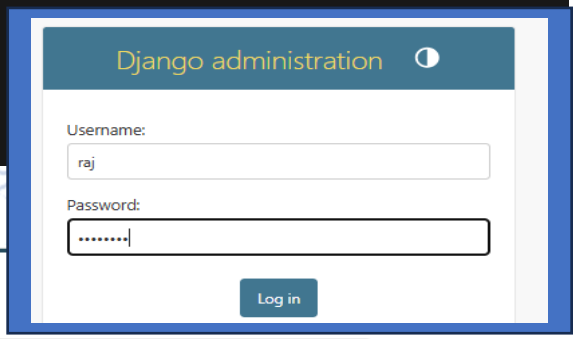


Step-2: create the superuser for add the Manul data through admin panel and form

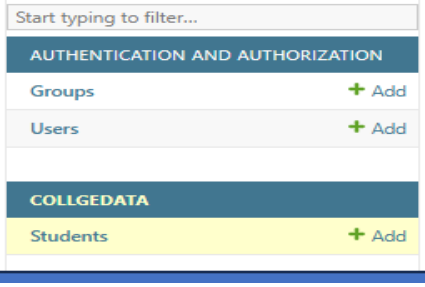
Run this command

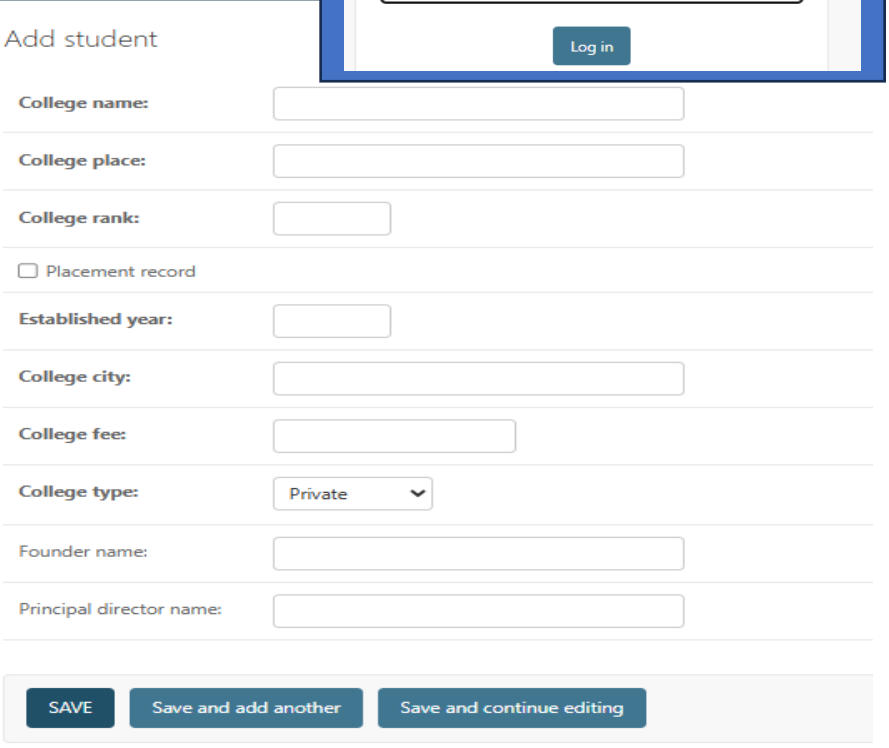
Example: python manage.py createsuperuser

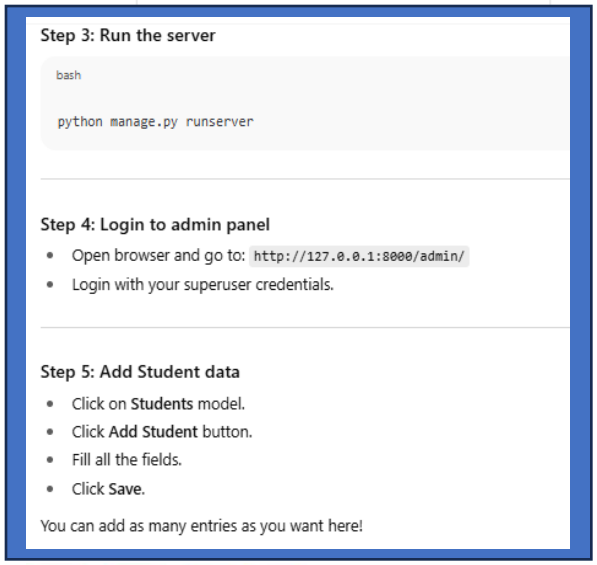




Step-3: open the admin & add data







2. Using Django shell

Step-1: Open Django shell for this runs this command

Example: python manage.py shell

Step 2: Import model and create data

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\kltk> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from collgedata.models import Student
>>> Student.objects.create(
...     college_name="IIT Kharagpur",
...     college_place="Kharagpur",
...     college_rank=1,
...     placement_record=True,
...     established_year=1951,
...     college_city="Kharagpur",
...     college_fee=200000.00,
...     college_type='Govt',
...     principal_director_name="Prof. V.K. Tewari",
...     founder_name="Jawaharlal Nehru"
... )
<Student: IIT Kharagpur (Govt) - Kharagpur>
>>>
>>> Student.objects.create(
...     college_name="BITS Pilani",
...     college_place="Pilani",
...     college_rank=6,
...     placement_record=True,
...     established_year=1964,
...     college_city="Pilani",
...     college_fee=300000.00,
...     college_type='Pvt',
...     principal_director_name="Prof. G. Sundar",
...     founder_name="G.D. Birla"
```

Step 3: Exit shell Example: exit ()

How to see the data are store in data base or not in Django shell

Step-1: Verify your data You can check if the data is saved by opening the shell again:

- **First run this command**: - python manage.py shell the write this code

  >> from collgedata.models import Student

  >> students = Student.objects.all()

  >> for s in students:

  … print(s)

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\kltk> python manage.py shell
7 objects imported automatically (use -v 2 for details).

Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from collgedata.models import Student
>>> students = Student.objects.all()
>>> for s in students:
...     print(s)
...
IIT Kharagpur (Govt) - Kharagpur
BITS Pilani (Pvt) - Pilani
IIT Madras (Govt) - Chennai
IIIT Hyderabad (Pvt) - Hyderabad
IISc Bangalore (Govt) - Bangalore
SRM Institute of Science and Technology (Pvt) - Chennai
IIT Kharagpur (Govt) - Kharagpur
BITS Pilani (Pvt) - Pilani
IIT Madras (Govt) - Chennai
IIIT Hyderabad (Pvt) - Hyderabad
IISc Bangalore (Govt) - Bangalore
SRM Institute of Science and Technology (Pvt) - Chennai
>>>
```

Step-5: Write the views for filter the data and map urls.py

views.py

```python
# collgedata/views.py
from django.shortcuts import render
from collgedata.models import Student
def home(request):
    # Store all records (in case we want to show total count or compare)
    allrecord = Student.objects.all()
    # Start filtered data as all records initially
    filtered_data = allrecord
    # Get search parameters from the URL
    city = request.GET.get('city')
    rank = request.GET.get('rank')
    college_type = request.GET.get('college_type')
    # Apply filters one by one
    if city:
        filtered_data = filtered_data.filter(college_city__icontains=city)
    if rank:
        filtered_data = filtered_data.filter(college_rank=rank)
    if college_type:
        filtered_data = filtered_data.filter(college_type=college_type)
    if rank:
        filtered_data = filtered_data.filter(college_rank=rank)
    context = {
        'allrecord': allrecord,         # total records (optional)
        'students': filtered_data,      # filtered results
    }
    return render(request, 'home.html', context)
```

urls.py

```python
"""
from django.contrib import admin
from django.urls import path
from kltk import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path("",views.home, name="home")
]

```

Search Colleges

City: BHOPAL Rank: e.g. 10 Type: Private Search

Filtered Results (12)

College Name	City	Place	Rank	Type	Fee	Founder	Principal/Director	Placement	Established
IIT Kharagpur	Kharagpur	Kharagpur	1	Government	₹200000.0	Jawaharlal Nehru	Prof. V.K. Tewari	Yes	1951
BITS Pilani	Pilani	Pilani	6	Private	₹300000.0	G.D. Birla	Prof. G. Sundar	Yes	1964
IIT Madras	Chennai	Adyar	2	Government	₹210000.0	Jawaharlal Nehru	Prof. V. Kamakoti	Yes	1959
IIIT Hyderabad	Hyderabad	Gachibowli	8	Private	₹280000.0	N. Chandrababu Naidu	Prof. P.J. Narayanan	Yes	1998
IISc Bangalore	Bangalore	Malleshwaram	3	Government	₹250000.0	Jamshedji Tata	Prof. Govindan Rangarajan	Yes	1909
SRM Institute of Science and Technology	Chennai	Kattankulathur	45	Private	₹320000.0	T.R. Pachamuthu	Dr. R. Shivakumar	Yes	1985

```
<!-- templates/home.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>College Search</title>
</head> <body>
  <h1>Search Colleges</h1>
  <!-- Search Filter Form -->
  <form method="get">
    City:<input type="text" name="city" placeholder="e.g. Delhi">
    Rank:<input type="number" name="rank" placeholder="e.g. 10">
    Type:<select name="college_type">
      <option value="">--Select--</option>
      <option value="Govt">Government</option>
      <option value="Pvt">Private</option>
       </select>
    <button type="submit">Search</button>
  </form><hr>
  <!-- Filtered Results -->
  <h2>Filtered Results ({{ students.count }})</h2>
  <table>
    <tr>
      <th>College Name</th>
      <th>City</th>
      <th>Place</th>
      <th>Rank</th>
      <th>Type</th>
      <th>Fee</th>
      <th>Founder</th>
      <th>Principal/Director</th>
      <th>Placement</th>
      <th>Established</th> </tr>
    {% for s in students %}
    <tr>
      <td>{{ s.college_name }}</td>
      <td>{{ s.college_city }}</td>
      <td>{{ s.college_place }}</td>
      <td>{{ s.college_rank }}</td>
      <td>{{ s.get_college_type_display }}</td>
      <td>₹{{ s.college_fee }}</td>
      <td>{{ s.founder_name }}</td>
      <td>{{ s.principal_director_name }}</td>
      <td>{{ s.placement_record|yesno:"Yes,No" }}</td>
      <td>{{ s.established_year }}</td> </tr>
    {% empty %}
    <tr><td colspan="10">No colleges found for the given filters.</td> </tr>
    {% endfor %}
  </table>
  <p><strong>Total Colleges in Database:</strong> {{ allrecord.count }}</p>
</body> </html>
```

CSS

```
<style>
    body {
      font-family: Arial, sans-serif;
      padding: 20px;
    }
    table {
      width: 100%;
      border-collapse: collapse;
    }
    th, td {
      padding: 8px;
      border: 1px solid #ccc;
      text-align: left;
    }
    th {background-color: #f2f2f2; }
    input, select, button {
      margin: 5px;
      padding: 5px;
    }
    form { margin-bottom: 20px;  }
```

Explanation:

- The <form> takes input for **city**, **rank**, and **college type** from the user and sends it to the Django view using a **GET request**.
- The Django view uses these inputs to **filter** the Student model records.
- {{ students.count }} shows how many results matched the filter.
- {% for s in students %} loops through filtered colleges and displays each one in the table.
- {% empty %} runs if no records match.
- get_college_type_display shows the full name ("Government"/"Private") from the choice field.
- yesno:"Yes,No" displays the boolean placement field in readable text.
- {{ allrecord.count }} shows the total number of colleges in the database (before filtering).

Example-2:

Step 1: Define the Model # models.py

```
from django.db import models
class Student(models.Model):
    name = models.CharField(max_length=100)
    marks = models.IntegerField()
```

Step 2: Add Data (via admin panel or shell # Django shell

```
python manage.py shell
from yourapp.models import Student
Student.objects.create(name="Rahul", marks=85)
Student.objects.create(name="Anjali", marks=76)
Student.objects.create(name="Riya", marks=90)
```

Step 3: Filter Data in Views # views.py

```
from django.shortcuts import render
from .models import Student
def show_top_students(request):
    top_students = Student.objects.filter(marks__gt=80)  # Filter students with marks > 80
    return render(request, 'students.html', {'students': top_students})
```

Step 4: Create URL Patter # urls.py

```
from django.urls import path
from . import views
urlpatterns = [
    path('top-students/', views.show_top_students, name='top_students'),
]
```

Step 5: Display Results in Template

```
<!-- students.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Top Students</title>
</head>
<body>
    <h1>Students with Marks > 80</h1>
    <ul>
        {% for student in students %}
            <li>{{ student.name }} - {{ student.marks }}</li>
        {% empty %}
            <li>No student found.</li>
        {% endfor %}
    </ul></body></html>
```

Step 6: Run Server and Visit Page

python manage.py runserver

Open your browser and go to:

http://127.0.0.1:8000/top-students/

1. icontains Filter (Case-insensitive contains)

- Use when you want to filter text fields using partial and case-insensitive matching.
- **Example:** Search for titles containing "news"
  - Display_News.objects.filter(news_title__icontains="news")

2. Date Filter

- You can filter based on exact date, year, month, day, etc.
- **Filter by specific date**
  - Display_News.objects.filter(date="2025-07-25")
- **Filter by year**
  - Display_News.objects.filter(date__year=2025)
- **Filter by month**
  - Display_News.objects.filter(date__month=7)
- **Filter by day**
  - Display_News.objects.filter(date__day=25)
- Filter by greater than (after a date)
  - **from datetime import date**
- Display_News.objects.filter(date__gt=date(2025, 1, 1))

3. Foreign Key Filters

- If your model has a foreign key, you can filter using related model fields.
- Assume this:

```
class Category(models.Model):
    name = models.CharField(max_length=100)
class Display_News(models.Model):
    news_title = models.CharField(max_length=100)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

Now you can filter like:

- **Filter news by category name**
  - Display_News.objects.filter(category__name="Sports")
- **Filter news by category id**
  - Display_News.objects.filter(category_id=1)

Example:

models.py

```
from django.db import models
from django.utils import timezone
class Category(models.Model):
    name = models.CharField(max_length=100)
class Display_News(models.Model):
    news_title = models.CharField(max_length=100)
    date = models.DateField(default=timezone.now)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
```

1. Filter by icontains (title search)

views.py

```
from django.shortcuts import render
from .models import Display_News
def filter_by_title(request):
    query = request.GET.get("q", "")
    news_list = Display_News.objects.filter(news_title__icontains=query)
    return render(request, "news.html", {"news_list": news_list})
```

2. Filter by date or parts of date

views.py

```
from datetime import datetime
from .models import Display_News
def filter_by_date(request):
    date_str = request.GET.get("date")  # Format: YYYY-MM-DD
    news_list = []
    if date_str:
        try:
            date_obj = datetime.strptime(date_str, "%Y-%m-%d").date()
            news_list = Display_News.objects.filter(date=date_obj)
        except ValueError:
            news_list = []
    return render(request, "news.html", {"news_list": news_list})
```

Or if you want year/month/day as separate inputs:

```
def filter_by_year_month_day(request):
    year = request.GET.get("year")
    month = request.GET.get("month")
    day = request.GET.get("day")
    filters = {}
    if year:
        filters["date__year"] = year
    if month:
        filters["date__month"] = month
    if day:
        filters["date__day"] = day
    news_list = Display_News.objects.filter(**filters)
    return render(request, "news.html", {"news_list": news_list})
```

3. Filter by foreign key (category)

```
from .models import Display_News
def filter_by_category(request):
    category_name = request.GET.get("category", "")
    news_list = Display_News.objects.filter(category__name__iexact=category_name)
    return render(request, "news.html", {"news_list": news_list})
```

4. Combined Filters (title + date + category)

```
def filter_combined(request):
    title = request.GET.get("title", "")
    date_str = request.GET.get("date", "")
    category = request.GET.get("category", "")
    filters = {}
    if title:
        filters["news_title__icontains"] = title
    if category:
        filters["category__name__iexact"] = category
    if date_str:
        try:
            date_obj = datetime.strptime(date_str, "%Y-%m-%d").date()
            filters["date"] = date_obj
        except ValueError:
            pass
    news_list = Display_News.objects.filter(**filters)
    return render(request, "news.html", {"news_list": news_list})
```

HTML Form Example for Combined Filter

```
<form method="get" action="{% url 'filter_combined' %}">
    Title: <input type="text" name="title">
    Date: <input type="date" name="date">
    Category: <input type="text" name="category">
    <input type="submit" value="Filter">
</form>
```

How to use the Django AutoSlugField

What is a Slug (Short Version)

- **Slug** is a short, URL-friendly version of a string — usually used to represent the **title** of a page or article in the URL.
- A **slug** is a clean, URL-friendly version of a title.
- Used in URLs to make them readable and SEO-friendly.

Example:

Title: India Wins World Cup 2025

Slug: india-wins-world-cup-2025

URL: example.com/news/india-wins-world-cup-2025

❖ What is AutoSlugField?

- AutoSlugField (from the django-autoslug package) automatically generates a slug from another field like a title. No need to manually set slug!

Basic Command for this project

- django-admin startproject demo6
- cd demo6
- mkdir templates
- pip install django-autoslug
- python manage.py startapp newdata
- python mangae.py runserver
- python manage.py makemigrations
- python manage.py migrate
- python manage.py createsuperuser

Example: - Use slug field in Django

Step 1: Install the required package for AutoSlugField

Example: pip install django-autoslug

Step 2: Add it to your models.py

Example: admin.py

admin.py

```
demo6 > newdata > admin.py > ...
1  # admin.py
2  from django.contrib import admin
3  from newdata.models import News
4  # @admin.register(News)
5  class NewsAdmin(admin.ModelAdmin):
6      list_display = ('title', 'slug',
       'date')
7  admin.site.register(News,NewsAdmin)
8
```

Step 3: Register the model in admin.py

Example: models.py

models.py

```
demo6 > newdata > models.py > ...
1  # models.py
2  from django.db import models
3  from autoslug import AutoSlugField
4  class News(models.Model):
5      title = models.CharField(max_length=200)
6      slug = AutoSlugField(populate_from='title', unique=True)
7      content = models.TextField()
8      date = models.DateField(auto_now_add=True)
9      def __str__(self):
10         return self.title
11
```

Step 4: Create and apply migrations

- python manage.py makemigrations
- python manage.py migrate

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo6> python manage.py makemigrations
Migrations for 'newdata':
  newdata\migrations\0001_initial.py
    + Create model News
PS C:\Users\hp\OneDrive\Desktop\djangopro\demo6> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, newdata, sessions
Running migrations:
```

Add models inside setting.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth'
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'newdata'
]
```

Step 5: Create sample data using admin or shell and open database to see data

- python manage.py createsuperuser

Step 6: Create a view to fetch the slug-based detail

```python
# views.py
from django.shortcuts import render
from newdata.models import News

def news_list(request):
    news_list = News.objects.all()
    d={'news_list': news_list}
    return render(request, 'news_list.html', d)
```

```python
"""
# urls.py
from django.contrib import admin
from django.urls import path
from demo6 import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('news/', views.news_list, name='news_list'),  # Shows a
    path('news/<slug:slug>/', views.news_list, name='new_slug'),
]
```

This slug type

This slug come from model field

Database data

Table: newdata_news

Filter in any column

	id	title	slug	content	date
	Filter	Filter	Filter	Filter	Filter
1	1	aaj tk	aaj-tk	this is new content	2025-07-25
2	2	Programing…	programing-langauge-coding-…	Hello everyone	2025-07-25
3	3	this is no…	this-is-not-a-math-class	kya aap thik hai	2025-07-25

Output:

127.0.0.1:8000/news/

All News Articles

- aaj tk (Slug: aaj-tk)
- Programing langauge coding lover (Slug: programing-langauge-coding-lover)
- this is not a math class (Slug: this-is-not-a-math-class)

How to add pagination in Django project

- **Pagination** means **dividing a large amount of data into smaller parts (pages)**
- Imagine you have **100 students** in a list. Showing all 100 on one page is too much.
  So, with pagination:
  - Page 1 shows 10 students
  - Page 2 shows the next 10 students
  - ... and so on up to Page 10

You can move between pages using **Next, Previous, or Page Numbers**.

Project Overview

We will create a Django project called pagiproject with an app college. It will show a list of colleges with pagination (5 per page). A styled **"Apply"** button will be present with each college, and the pagination buttons will look like boxes and change appearance when active.

Step 1: Create Django Project and App

- django-admin startproject pagiproject
- cd pagiproject
- python manage.py startapp college

Step 2: Register the App

- In pagiproject/settings.py, add 'college', to INSTALLED_APPS:

```
INSTALLED_APPS = [
    ...
    'college',
]
```

```
PS C:\Users\hp\OneDrive\Desktop\djangopro> django-admin startproject pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro> cd pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro\pagiproject> python manage.py startapp college
PS C:\Users\hp\OneDrive\Desktop\djangopro\pagiproject>
PS C:\Users\hp\OneDrive\Desktop\djangopro> django-admin startproject pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro> cd pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro> django-admin startproject pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro> django-admin startproject pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro> cd pagiproject
PS C:\Users\hp\OneDrive\Desktop\djangopro\pagiproject> python manage.py startapp college
PS C:\Users\hp\OneDrive\Desktop\djangopro\pagiproject> python manage.py makemigrations
Migrations for 'college':
  college\migrations\0001_initial.py
    + Create model College
PS C:\Users\hp\OneDrive\Desktop\djangopro\pagiproject> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, college, contenttypes, sessions
Running migrations:
```

Step 3: Create Model and Step 4: Create Dummy Data

models.py

```python
from django.db import models
class College(models.Model):
    name = models.CharField(max_length=100)
    city = models.CharField(max_length=50)
    rank = models.IntegerField()

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

admin.py

```python
from django.contrib import admin
from college.models import College
# @admin.register(College)
class CollegeAdmin(admin.ModelAdmin):
    list_display = ('name', 'city', 'rank')
admin.site.register(College, CollegeAdmin)
```

Then run this command and create superuser password and enter the college data

- python manage.py createsuperuser

Add Dummy Data via Python Shell

1. Open Django Shell

Run this in your terminal: > python manage.py shell

2. Run These Commands in the Shell

```
from college.models import College
College.objects.create(name="IIT Delhi", city="Delhi", rank=1)
College.objects.create(name="IIT Bombay", city="Mumbai", rank=2)
College.objects.create(name="IIT Kanpur", city="Kanpur", rank=3)
College.objects.create(name="IIT Madras", city="Chennai", rank=4)
College.objects.create(name="NIT Trichy", city="Trichy", rank=5)
College.objects.create(name="BITS Pilani", city="Pilani", rank=6)
College.objects.create(name="VIT Vellore", city="Vellore", rank=7)
College.objects.create(name="IIIT Hyderabad", city="Hyderabad", rank=8)
College.objects.create(name="DTU", city="Delhi", rank=9)
College.objects.create(name="NSUT", city="Delhi", rank=10)
```

3. Exit Shell exit()

Open database & see

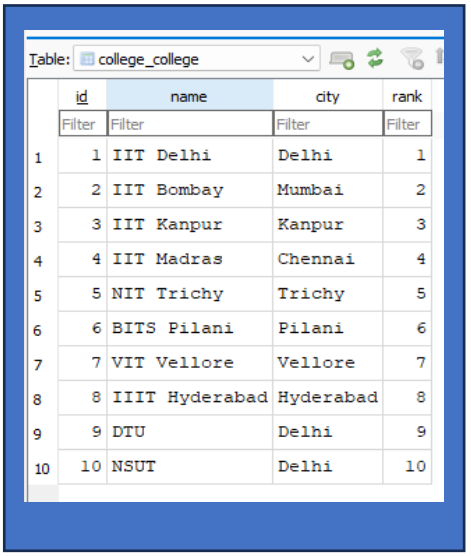


Table: college_college

	id	name	city	rank
	Filter	Filter	Filter	Filter
1	1	IIT Delhi	Delhi	1
2	2	IIT Bombay	Mumbai	2
3	3	IIT Kanpur	Kanpur	3
4	4	IIT Madras	Chennai	4
5	5	NIT Trichy	Trichy	5
6	6	BITS Pilani	Pilani	6
7	7	VIT Vellore	Vellore	7
8	8	IIIT Hyderabad	Hyderabad	8
9	9	DTU	Delhi	9
10	10	NSUT	Delhi	10

Step 5: Create Views with Pagination

```python
# Import required modules
from django.shortcuts import render  # For rendering templates
from college.models import College          # College model
from django.core.paginator import Paginator  # For pagination
def college_list(request):
    colleges = College.objects.all()              # Fetch all records
    paginator = Paginator(colleges, 5)            # 5 per page
    page_number = request.GET.get("page")         # Get page from URL
    page_obj = paginator.get_page(page_number)    # Get current page data
    d={'page_obj': page_obj}
    return render(request, 'clg.html',d)  # Send to template
```

Step 6: URL Configuration

In `college/urls.py` (create this file):

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

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

In `pagiproject/urls.py`:

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

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

clg.html(templates)

```
<!DOCTYPE html>
<html>
<head>
  <title>College List with Pagination</title>
</head>
<body>
  <h2 style="text-align: center;">College List</h2>
  <!-- Looping through paginated college records passed as 'page_obj' from Django view -->
  {% for college in page_obj %}
    <div class="college-box">
      <!-- Apply button for each college -->
      <button class="apply-btn">Apply</button>
      <!-- Display college name -->
      <h3>{{ college.name }}</h3>
      <!-- Display college city -->
      <p>City: {{ college.city }}</p>
      <!-- Display college rank -->
      <p>Rank: {{ college.rank }}</p>
    </div>
  {% endfor %}
  <!-- Pagination section -->
  <div class="pagination">
    <!-- Show "First" and "Prev" only if previous pages exist -->
    {% if page_obj.has_previous %}
      <a href="?page=1">First</a>
      <a href="?page={{ page_obj.previous_page_number }}">Prev</a>
    {% endif %}
    <!-- Loop through all page numbers -->
    {% for num in page_obj.paginator.page_range %}
      {% if page_obj.number == num %}
        <!-- Highlight current page -->
        <a class="active">{{ num }}</a>
      {% else %}
        <!-- Link to other pages -->
        <a href="?page={{ num }}">{{ num }}</a>
      {% endif %}
    {% endfor %}
    <!-- Show "Next" and "Last" only if more pages exist -->
    {% if page_obj.has_next %}
      <a href="?page={{ page_obj.next_page_number }}">Next</a>
      <a href="?page={{ page_obj.paginator.num_pages }}">Last</a>
    {% endif %}
  </div>
</body>
</html>
```

```
<style>
    .college-box {
      border: 1px solid #ccc;
      padding: 15px;
      margin: 10px;
      border-radius: 8px;
      box-shadow: 2px 2px 10px
rgba(0,0,0,0.1);
    }
    .apply-btn {
      background-color: #3498db;
      color: white;
      padding: 8px 15px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      margin-bottom: 10px; /* space
below button */
    }
    .apply-btn:hover {
      background-color: #2980b9;
    }
    .pagination {
      display: flex;
      justify-content: center;
      margin-top: 20px;
    }
    .pagination a {
      margin: 0 5px;
      padding: 10px 15px;
      border: 1px solid #ddd;
      text-decoration: none;
      color: black;
      border-radius: 5px;
      background-color: #f4f4f4;
    }
    .pagination .active {
      background-color: #3498db;
      color: white;
      font-weight: bold;
    }
  </style>
```

Apply

IIT Madras

City: Chennai

Rank: 4

Apply

NIT Trichy

City: Trichy

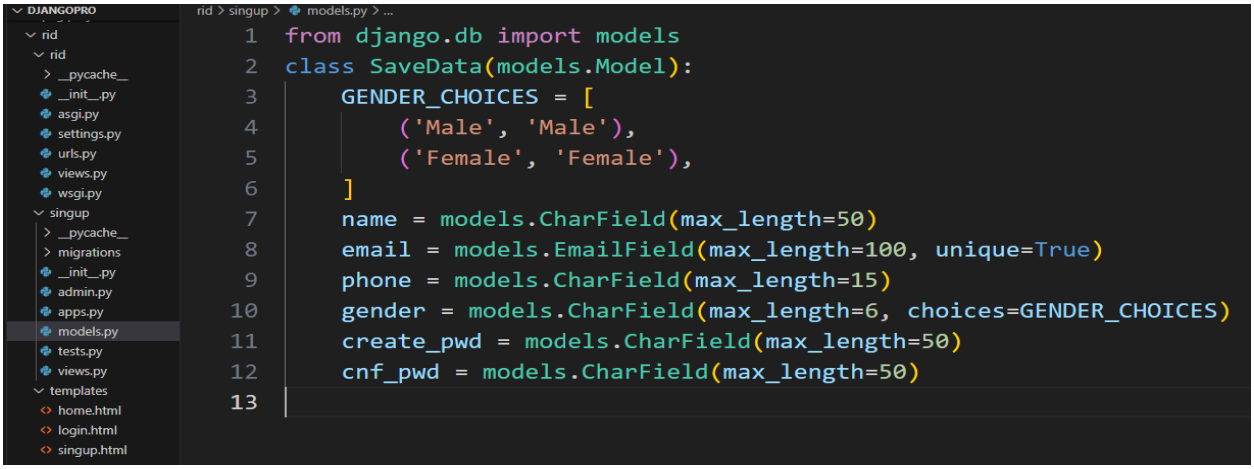
Rank: 5

1 2 Next Last

How to add data from (Front End) HTML form to database

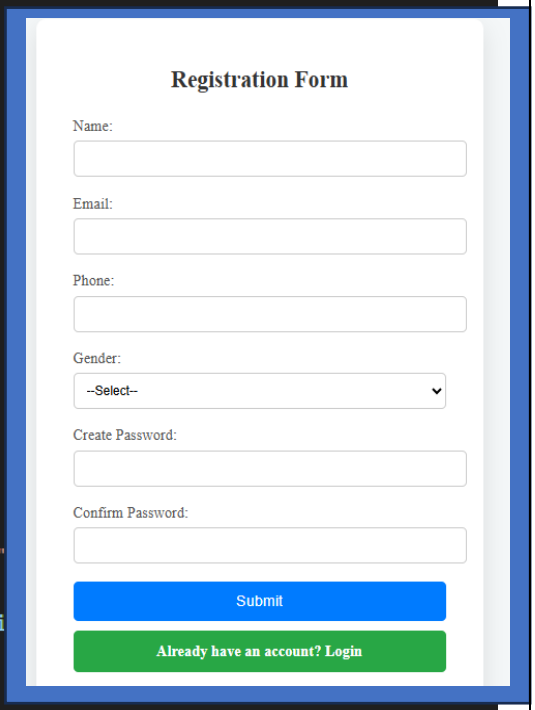
Step-1: create the model use this command > python manage.py startapp app_name

Example:> python manage.py startapp singup



step- open database and see the table and row and create the signup.html (templates)

```
<!DOCTYPE html>
<html>
<head> <title>Save Data Form</title> </head>
<body> <form method="POST" action="">
        {% if error %}
    <p style="color: red; text-align:center;">{{ error }}</p>
    <p style="text-align:center;"> Already registered? <a href="{% url 'login' %}">Login here</a>
    </p>
    {% endif %}
        <h2>Registration Form</h2>
        {% csrf_token %}
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <label for="phone">Phone:</label>
        <input type="text" id="phone" name="phone" required>
        <label for="gender">Gender:</label>
        <select id="gender" name="gender" required>
            <option value="">--Select--</option>
            <option value="Male">Male</option>
            <option value="Female">Female</option>
        </select>
        <label for="create_pwd">Create Password:</label>
        <input type="password" id="create_pwd" name="create_pwd"
        <label for="cnf_pwd">Confirm Password:</label>
        <input type="password" id="cnf_pwd" name="cnf_pwd" requi
        <button type="submit">Submit</button>
        <!-- Login Button/Link -->
        <a href="{% url 'login' %}" class="login-btn">Already have an account? Login</a>
    </form>
</body></html>
```

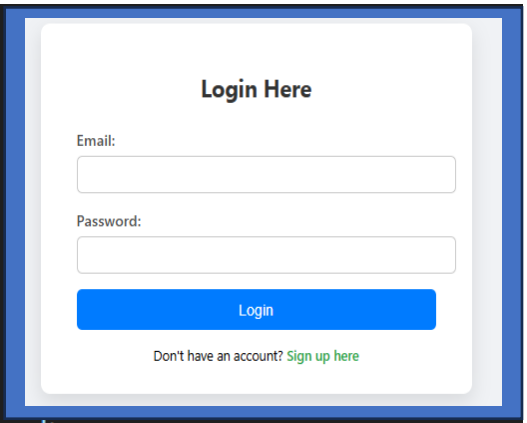


Singup.html(templates)

```
<!DOCTYPE html>
<html>
<head>
  <title>Save Data Form</title>
</head>
<body>
  <form method="POST" action="">
    {% if error %}
  <p style="color: red; text-align:center;">{{ error }}</p>
  <p style="text-align:center;">
    Already registered? <a href="{% url 'login' %}">Login
here</a>
  </p>
  {% endif %}
    <h2>Registration Form</h2>
    {% csrf_token %}
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>
    <label for="phone">Phone:</label>
    <input type="text" id="phone" name="phone" required>
    <label for="gender">Gender:</label>
    <select id="gender" name="gender" required>
      <option value="">--Select--</option>
      <option value="Male">Male</option>
      <option value="Female">Female</option>
    </select>
    <label for="create_pwd">Create Password:</label>
    <input type="password" id="create_pwd"
name="create_pwd" required>
    <label for="cnf_pwd">Confirm Password:</label>
    <input type="password" id="cnf_pwd" name="cnf_pwd"
required>
    <button type="submit">Submit</button>
    <!-- Login Button/Link -->
    <a href="{% url 'login' %}" class="login-btn">Already have
an account? Login</a>
  </form>
</body>
</html>
```

```
<style>
    body {
      background: #f4f7f8;
      display: flex;
      justify-content: center;
      align-items: center; height: 100vh;}
    form {
      background: white;
      padding: 30px 40px;
      border-radius: 10px;
      box-shadow: 0 10px 25px rgba(0,0,0,0.1);
      width: 100%;
      max-width: 400px;  }
    h2 {
      text-align: center;
      margin-bottom: 25px;
      color: #333  }
    label {
      display: block;
      margin-bottom: 6px;
      color: #555;
      font-weight: 500; }
    input, select {
      width: 100%; padding: 10px;
      margin-bottom: 18px;
      border: 1px solid #ccc; border-radius: 5px;
      transition: border 0.3s ease;
    } input:focus, select:focus {
      border-color: #007bff; outline: none; }
    button {
      width: 100%;padding: 12px; background: #007bff;
      border: none; color: white;
      border-radius: 5px;
      cursor: pointer;
      transition: background 0.3s ease;
      margin-bottom: 10px;
    }button:hover {background: #0056b3;}
    .login-btn {
      background: #28a745; /* Green */
      text-align: center;
      text-decoration: none; display: block; padding: 12px;
      border-radius: 5px;color: white;
      font-weight: 600; }
    .login-btn:hover {
      background: #218838;
    }
  </style>
```

```html
<!-- templates/login.html -->
<!DOCTYPE html>
<html>
<head>
    <title>Login Page</title>
</head>
<body>
    <div class="login-container">
        <h2>Login Here</h2>
        <form method="POST">
            {% csrf_token %}
            <label for="email">Email:</label>
            <input type="email" name="email" required>
            <label for="password">Password:</label>
            <input type="password" name="password" required>
            <button type="submit">Login</button>
        </form>
        <div class="signup-link">
            Don't have an account? <a href="{% url 'singup' %}">Sign up here</a>
        </div></div>
</body></html>
```




```css
<style>
    body {
      background-color: #f0f2f5;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }
    .login-container {
      background-color: white;
      padding: 30px 40px;
      border-radius: 10px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
      max-width: 400px;
      width: 100%;
    }
    h2 {
      text-align: center;
      color: #333;
      margin-bottom: 25px;
    }
    label {
      font-weight: 500;
      color: #555;
    }
```

```css
    input {
      width: 100%;
      padding: 10px;
      margin-top: 6px;
      margin-bottom: 15px;
      border: 1px solid #ccc;
      border-radius: 6px;
    }
    button {
      width: 100%;
      padding: 12px;
      background-color: #007bff;
      color: white;
      border: none;
      font-size: 16px;
      border-radius: 6px;
      cursor: pointer;}
    button:hover {background-color: #0056b3;}
    .signup-link {
      text-align: center;
      margin-top: 15px;
      font-size: 14px;}
    .signup-link a {
      color: #28a745;
      text-decoration: none;
      font-weight: 600;  }
    .signup-link a:hover {
      text-decoration: underline;
    } </style>
```

Views.py (signup method)

```python
from django.shortcuts import render, redirect
from singup.models import SaveData
def singup(req):
    error = ""
    if req.method == "POST":
        name = req.POST.get("name")
        email = req.POST.get("email")
        phone = req.POST.get("phone")
        gender = req.POST.get("gender")
        create_pwd = req.POST.get("create_pwd")
        cnf_pwd = req.POST.get("cnf_pwd")
        if create_pwd != cnf_pwd:
            error = "Passwords do not match."
        else:
            try:
                SaveData.objects.create(
                    name=name,
                    email=email,
                    phone=phone,
                    gender=gender,
                    create_pwd=create_pwd,
                    cnf_pwd=cnf_pwd
                )
                return redirect('login')
            except Exception as e:
                error = "Email already registered or error saving data."
    return render(req, "singup.html", {"error": error})
```

Same

Views.py (signup method)

```python
def login(req):
    error = ""
    if req.method == "POST":
        loginemail = req.POST.get("email")
        loginpassword = req.POST.get("password")
        try:
            user = SaveData.objects.get(email=loginemail, create_pwd=loginpassword)
            # If match found, redirect to home
            return redirect('home')
        except SaveData.DoesNotExist:
            error = "Invalid email or password."
    return render(req, "login.html", {"error": error})
def home(req):
    return render(req, "home.html")
```

How to upload file in Django in admin panel

```
Length Name
------ ----
       fileup
       imageup
       media
       static
       templates
135168 db.sqlite3
   684 manage.py
```

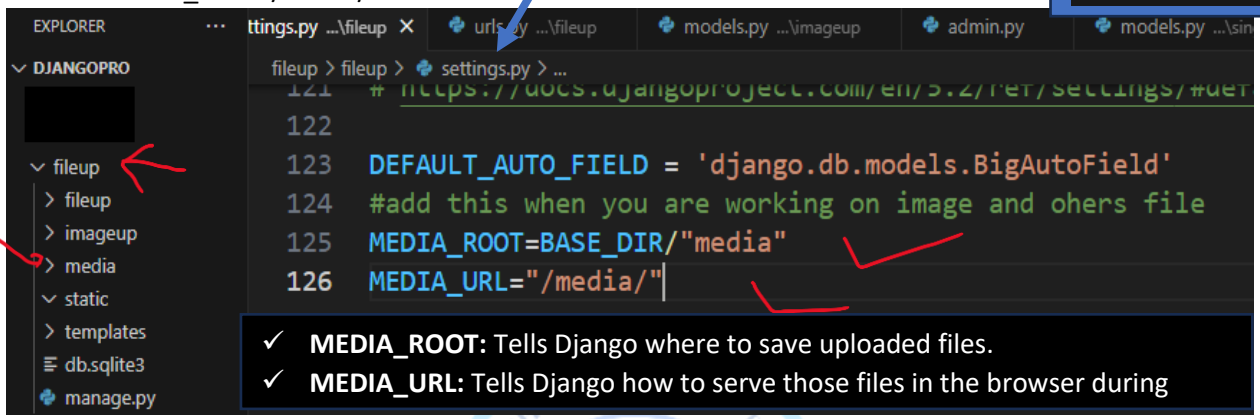
Step-1: - create media folder inside the main project folder

Step-2: add the media root in setting.py

#add this when you are working on image and ohers file

MEDIA_ROOT=BASE_DIR/"media"

MEDIA_URL="/media/"



❖ **Note:**

- Needed when your app allows users to **upload files**.
- Without this, Django won't know **where to store** or **how to serve** uploaded files.

✓ **Must add when working with image upload or file fields like FileField, ImageField.**

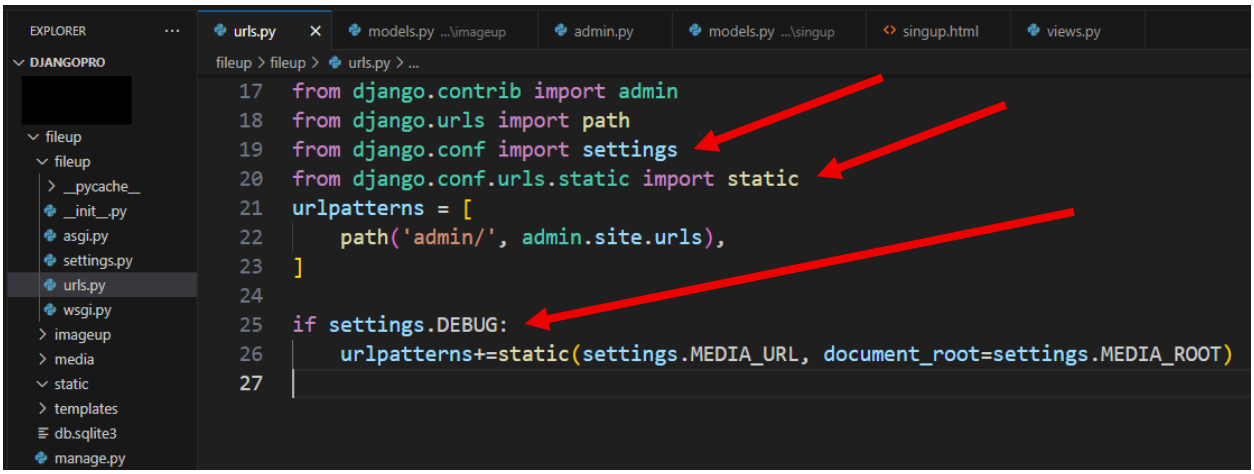
❖ Why we need medial folder separate

- We create the media folder so Django knows **where to save** and **how to serve** uploaded files during development. It must match the **MEDIA_ROOT** path in your settings.py.
- We create the **media folder** to store uploaded files like images and documents.

Step-3: import this module inside the main project urls.py

✓ **from django.conf import settings**

✓ **from django.conf.urls.static import static**



1. **from django.conf import settings** → To access project settings like MEDIA_URL and MEDIA_ROOT.
2. **from django.conf.urls.static import static** → To enable serving of media files (images, docs, etc.) during development (DEBUG=True).
   it works for **any uploaded files**, like:
   - Images (.jpg, .png)
   - Documents (.pdf, .docx)
   - Videos (.mp4) and Any other file types

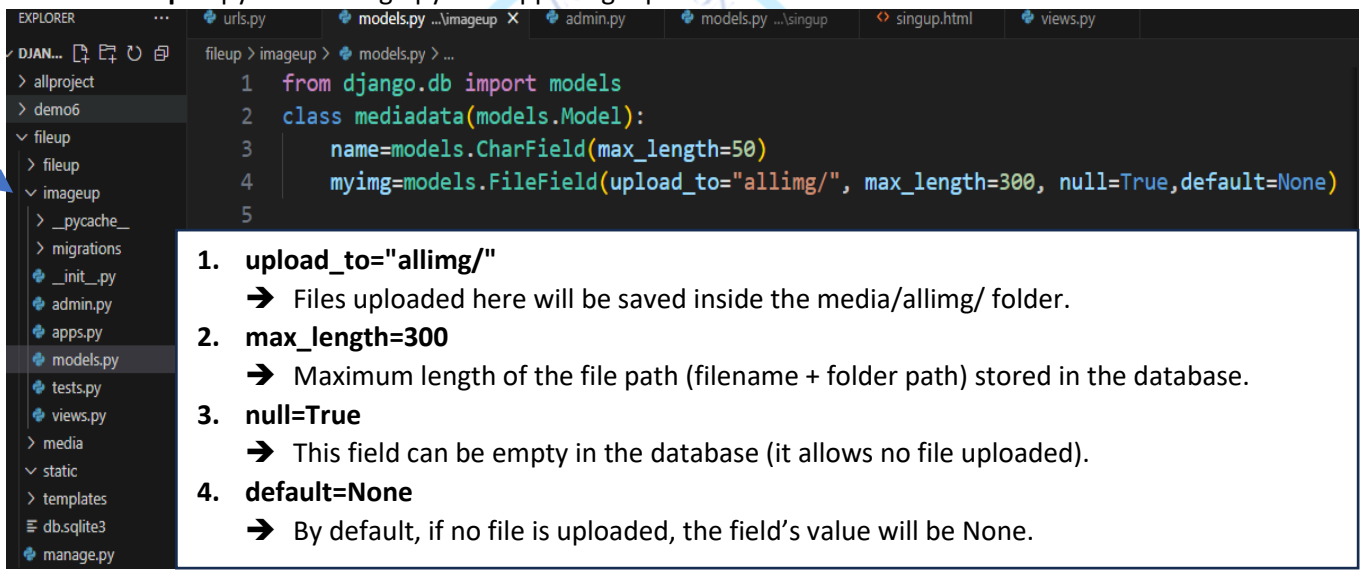
   ➔ **It handles all media files uploaded using FileField or ImageField.**

```
if settings.DEBUG:
urlpatterns+=static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

➢ We use this to **serve uploaded media files (images, documents, etc.) only during development**.
- **if settings.DEBUG:** ensures it runs **only when debugging is on** (not in production).
- It adds URL patterns to serve files from **MEDIA_ROOT** at **MEDIA_URL** so you can access them in the browser while developing.

Step-4: Create models and desing model for upload the documents

- **Syntax:** python manage.py startapp aapname
- **Example:** python manage.py startapp imageup



1. **upload_to="allimg/"**
   ➔ Files uploaded here will be saved inside the media/allimg/ folder.
2. **max_length=300**
   ➔ Maximum length of the file path (filename + folder path) stored in the database.
3. **null=True**
   ➔ This field can be empty in the database (it allows no file uploaded).
4. **default=None**
   ➔ By default, if no file is uploaded, the field's value will be None.

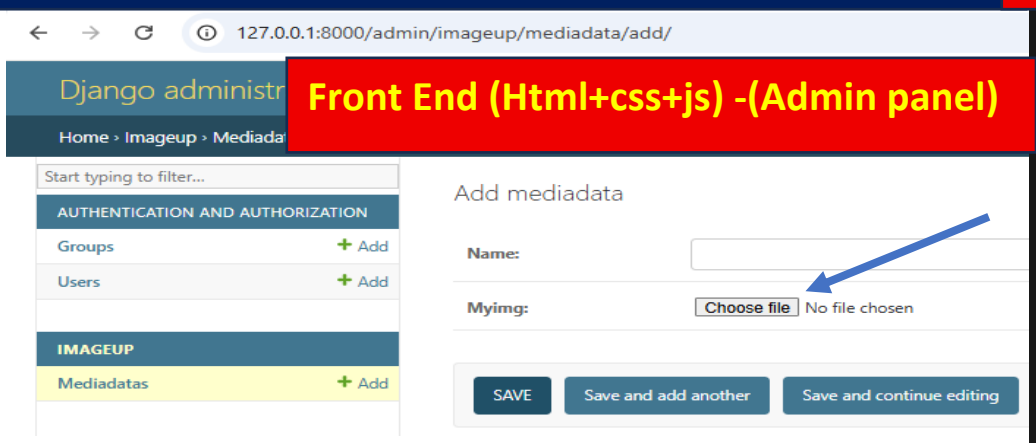
Step-5: Admin.py write the code for show the field in admin side

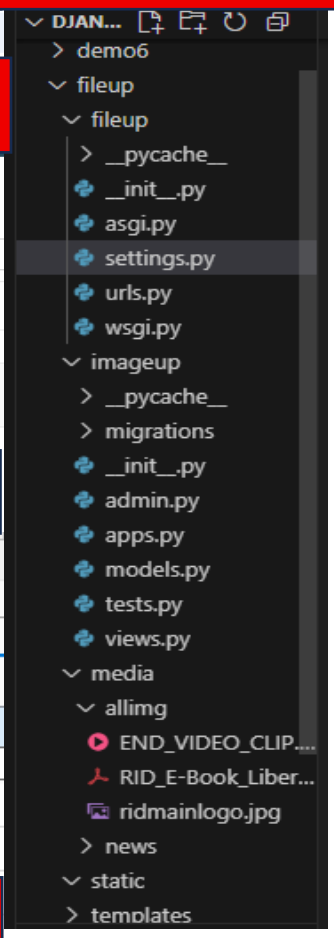
Step-6: Run these three commands
1. Python manage.py makemigrations
2. Python manage.py migrate
3. Python manage.py createsuperuser

Step-7: open admin panel & upload file
➔ http://127.0.0.1:8000/

How to upload file in Django through admin panel

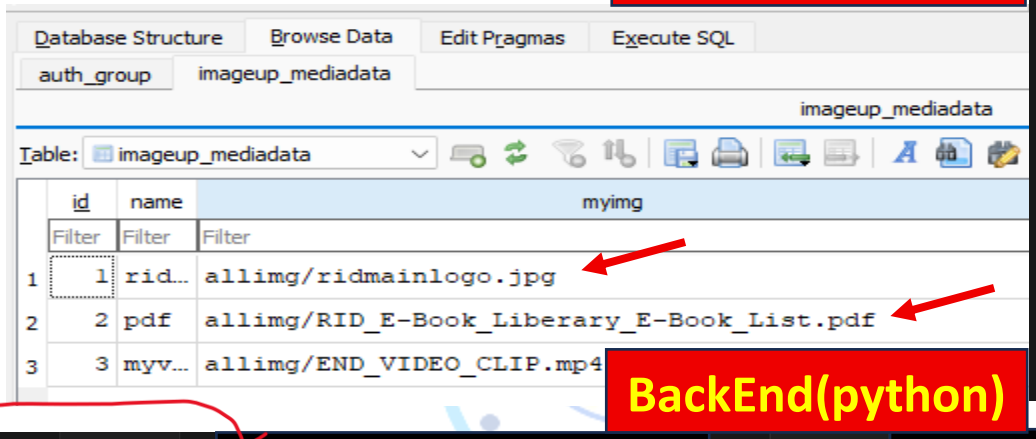
Project (Folder Structure)





Step-8: Open database & see the data

Database (MySql)



BackEnd(python)

admin.py(model folder)

```python
from django.contrib import admin
from imageup.models import mediadata
class mediadataAdmin(admin.ModelAdmin):
    list_display = ('id', 'name', 'myimg')  # Disp
admin.site.register(mediadata, mediadataAdmin)
```

models.py (app folder)

```python
from django.db import models
class mediadata(models.Model):
    name=models.CharField(max_length=50)
    myimg=models.FileField(
                    upload_to="allimg/",
                    max_length=300,
                    null=True,
                    default=None
                    )
```

Urls.py (main project)

```python
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
]
if settings.DEBUG:
    urlpatterns+=static(settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT)
```

setting.py (main project)

```python
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
#add this when you are working on image and ohers file
MEDIA_ROOT=BASE_DIR/"media"
MEDIA_URL="/media/"
```

How to Display the uploaded image in Django

Step-6: Write views.py method set urls.py path to access image from database

views.py

```python
from django.shortcuts import render
from imageup.models import mediadata
def home(request):
    data = mediadata.objects.first()  # or filter/query as needed
    return render(request, 'home.html', {'mediadata': data})
# Get first record          mediadata.objects.first()
# Get 10th record           mediadata.objects.all()[9]
# Get record with id=5      mediadata.objects.get(id=5)
# Safe get with id=5        mediadata.objects.filter(id=5).first()
# Get all data (to loop)    mediadata.objects.all()
```

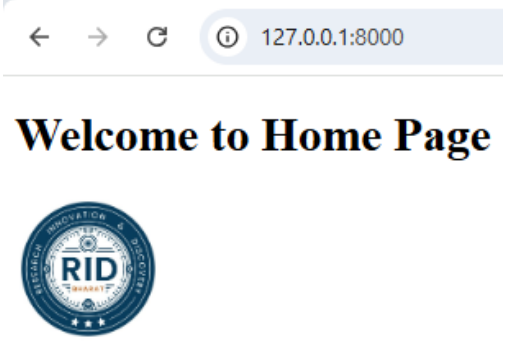
urls.py

```python
# from fileup import views
from . import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.home)
]
if settings.DEBUG:
    urlpatterns+=static(settings.MEDIA_URL
        document_root=settings.MEDIA_ROOT)

```

Home.html(templates)

home.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>
<body>
    <h1>Welcome to Home Page</h1>
    <div>
        {% if mediadata.myimg %}
            <img src="{{ mediadata.myimg.url }}" width="100px" height="100px">
        {% else %}
            <p>No image uploaded.</p>
        {% endif %}
    </div>
</body>
</html>
```



Before Learning Database Connectivity following keep in your Django

Add this in setting.py for static file, media file, templates, and models(app)

```
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA  URL = '/media/'
```

```
TEMPLATES = [ {
  .....
 'DIRS': [BASE_DIR, "templates"],
'APP_DIRS': True,
....
}]
```

```
INSTALLED_APPS = [
'django.contrib.admin',
   ..............
   'appname'
]
```

Add this in setting.py for connect with MongoDB (only python 3.12 and below support)

```
DATABASES = {
            'default': {
            'ENGINE': 'djongo',
            'NAME': 'mymongodb',          # Write here your MongoDB database name
            'ENFORCE_SCHEMA': False,      # Optional
            'CLIENT': {
            'host': 'mongodb://localhost:27017',} }}
```

Install these Required Packages for connecting MongoDB

- pip install djongo **:- Use Django models with MongoDB**
- pip install pymongo **:- Connect Python/Djongo to MongoDB**

Run these two commands after database add and model change

1. python manage.py makemigrations
2. python manage.py migrate

Add this in settings.py to connect with MySQL

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'yourdbname',              # Write here your MySQL database name
      'USER': 'yourusername',            # Your MySQL username
      'PASSWORD': 'yourpassword',        # Your MySQL password
      'HOST': 'localhost',               # Usually 'localhost' if using local MySQL
      'PORT': '3306',                    # Default MySQL port
   } }
```

❖ **Install these Required Packages for connecting MySQL**

- pip install mysqlclient **:- Enables Django to work with MySQL through the ORM Official MySQL database adapter for Django (if error)**
- pip install pymysql **:- Pure-Python MySQL client (alternative to mysqlclient) Easier to install but slightly slower; also requires extra setup below For PyMySQL users, add this to the top of your settings.py**
- import pymysql
- pymysql.install_as_MySQLdb()

Access or serves media files like images, PDFs, videos, add this in urls.py(main project)

```
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
   path('admin/', admin.site.urls),
]# This serves media files like images, PDFs, videos in development
if settings.DEBUG:
   urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Install this for model file/image uploads

- pip install Pillow
- **Pillow** is required for Django's ImageField to work. model uses file/image uploads

How to save the data from (Front End) to Any database (Main)

Step-1: create the Django project and app (for models)

Run this command 1. Django-admin startproject project_name 2. Python manage.py startapp app_name

Add these all things in setting.py →

settings.py | views.py | urls.py | form

abt > abt > settings.py > ...

```
118 STATIC_URL = 'static/'
119 import os
120 # Static files (CSS, JS)
121 STATIC_URL = '/static/'
122 STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
123 # Default primary key field type
124 # https://docs.djangoproject.com/en/5.2/ref/settings/#de
125 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
126 #add this when you are working on image and ohers file
127 MEDIA_ROOT=BASE_DIR/"media"
128 MEDIA_URL="/media/"
```

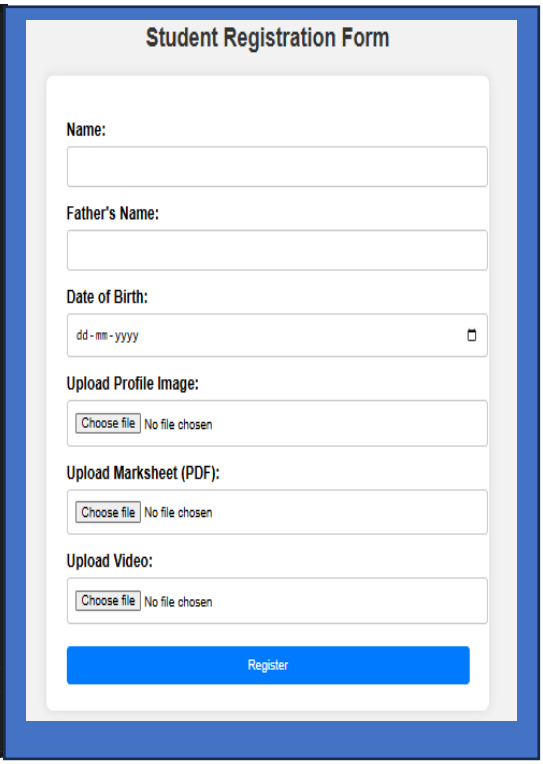
```
INSTALLED_APPS = [
    .....
    'formdata'
]
```

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

Step-2: create from.html inside templates

templates > form.html > html > body > form

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Student Registration</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <h2>Student Registration Form</h2>
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        <label for="id_name">Name:</label>
        <input type="text" name="name" id="id_name" required>
        <label for="id_father_name">Father's Name:</label>
        <input type="text" name="father_name" id="id_father_name" required>
        <label for="id_dob">Date of Birth:</label>
        <input type="date" name="dob" id="id_dob" required>
        <label for="id_profile_image">Upload Profile Image:</label>
        <input type="file" name="profile_image" id="id_profile_image" accept="image/*" required>
        <label for="id_marksheet_pdf">Upload Marksheet (PDF):</label>
        <input type="file" name="marksheet_pdf" id="id_marksheet_pdf" accept="application/pdf" required>
        <label for="id_video_file">Upload Video:</label>
        <input type="file" name="video_file" id="id_video_file" accept="video/*" required>
        <button type="submit">Register</button>
    </form>
</body>
</html>
```



Step-3: Create the model and design the model for the store the data

models.py

abt > formdata > models.py > Student > __str__

```python
from django.db import models
class Student(models.Model):
    name = models.CharField(max_length=100)
    father_name = models.CharField(max_length=100)
    dob = models.DateField()
    # Profile image file, saved to 'media/profile_images folder/'
    profile_image = models.ImageField(upload_to='profile_images/')
    marksheet_pdf = models.FileField(upload_to='marksheets/')
    # Uploaded video file, saved to 'media/videos/
    video_file = models.FileField(upload_to='videos/')
```

Run this command → Python manage.py makemigrations and python manage.py migrate

Without Pillow, Django can't handle uploaded image files.

Why Pillow is Required? ImageField (used in profile_image = models.ImageField(...)) needs Pillow to:

- Validate image format, Handle image dimensions and file headers, Save and process images

Solution: Install Pillow Run this command in your terminal or command prompt:

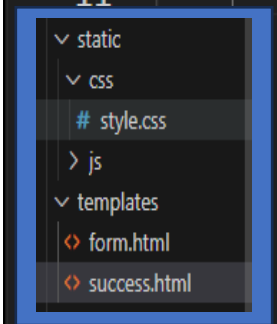
- **python -m pip install Pillow**:- This will install the required image processing library that Django uses internally for ImageField.

Step-4: write the views for access the data from the form and store in database

views.py

abt > abt > views.py > home

```python
from django.shortcuts import render
from formdata.models import Student
def home(request):
    if request.method == 'POST':
        name_1 = request.POST.get('name')
        father_name_1 = request.POST.get('father_name')
        dob_1 = request.POST.get('dob')
        profile_image_1 = request.FILES.get('profile_image')
        marksheet_pdf_1 = request.FILES.get('marksheet_pdf')
        video_file_1 = request.FILES.get('video_file')
        # Save to database
        Student.objects.create(
            name=name_1,
            father_name=father_name_1,
            dob=dob_1,
            profile_image=profile_image_1,
            marksheet_pdf=marksheet_pdf_1,
            video_file=video_file_1
        )
        return render(request, 'success.html')
    return render(request, 'form.html')
```



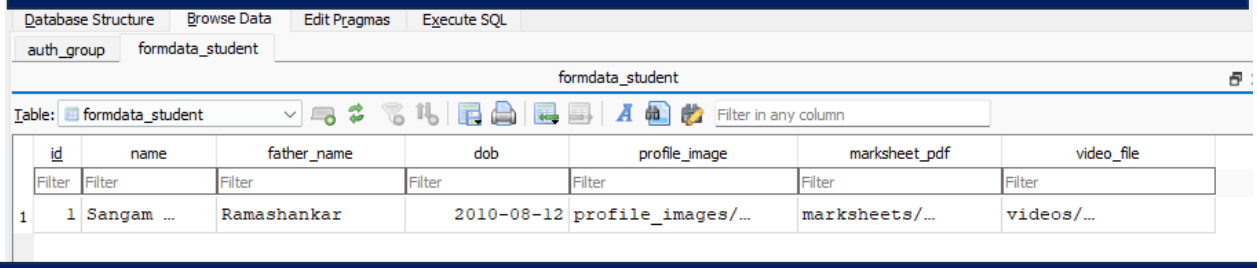
Urls.py

urls.py

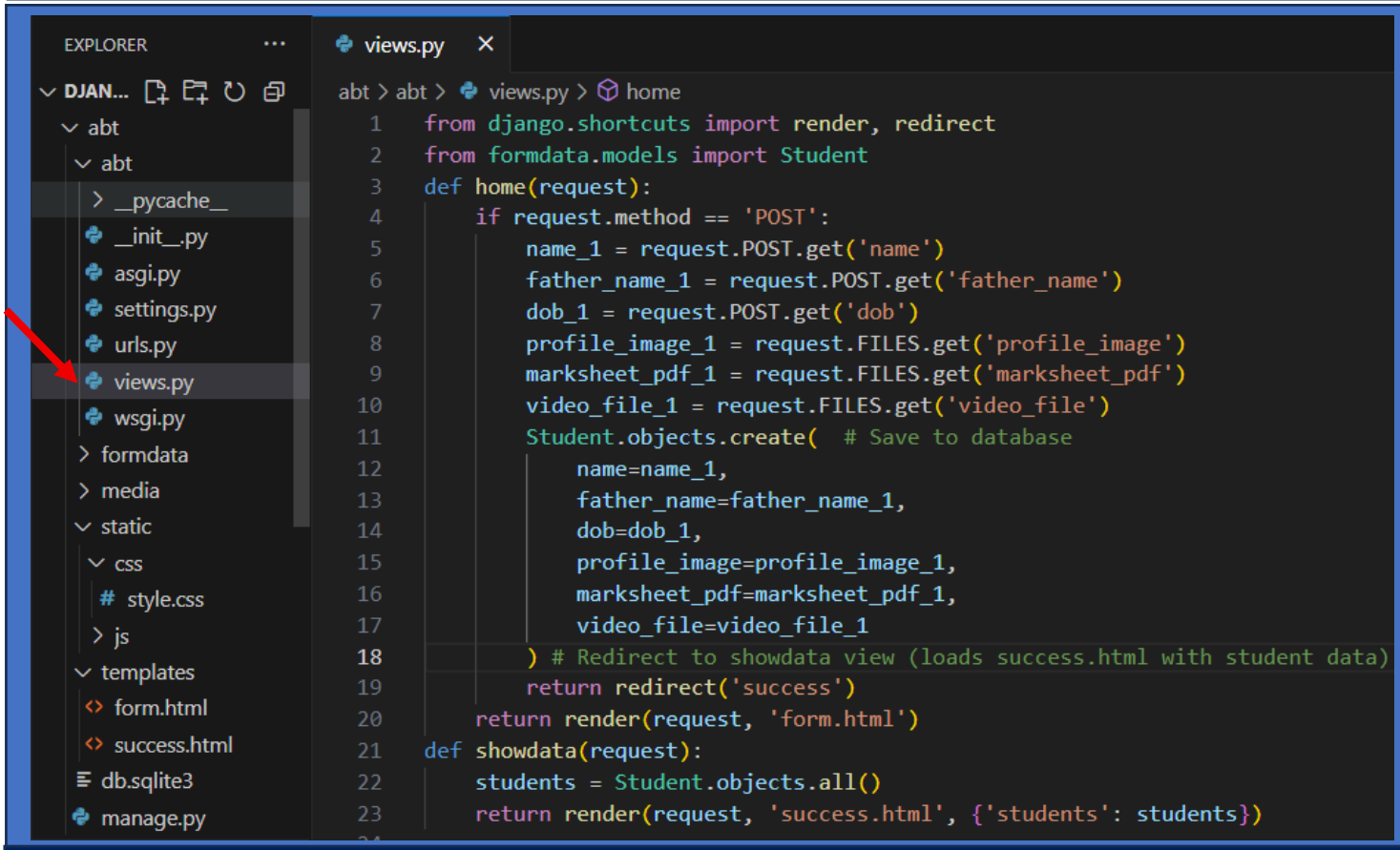
abt > abt > urls.py > ...

```python
from django.contrib import admin
from django.urls import path
from .import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.home)
]
```

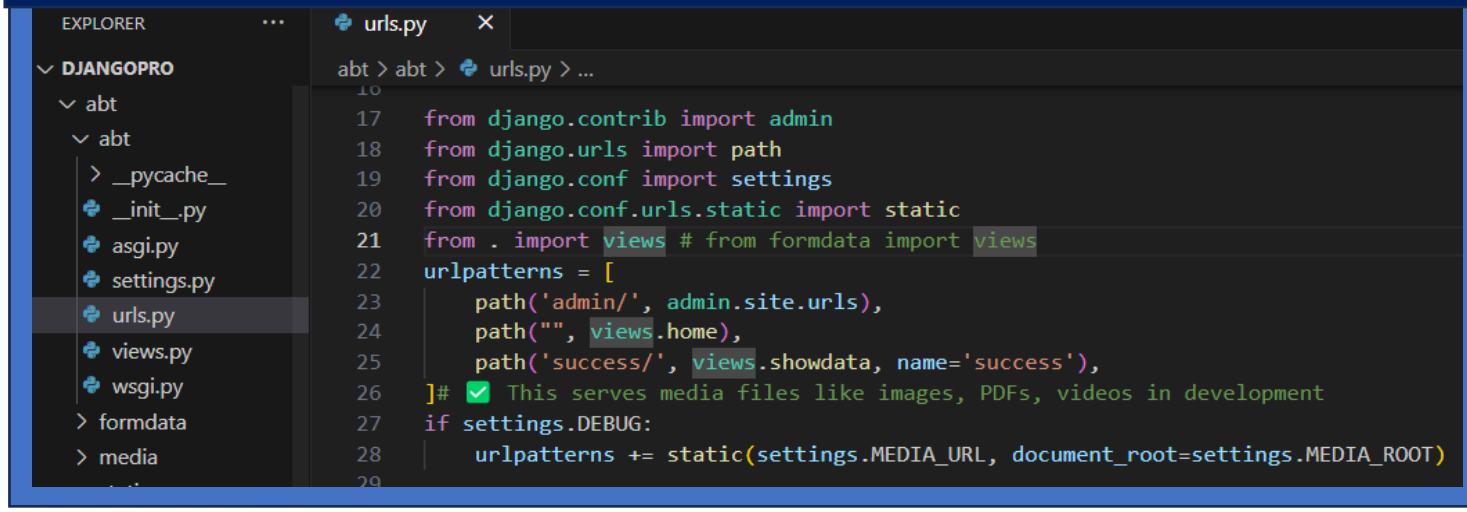
Step-5: Open database and see the data



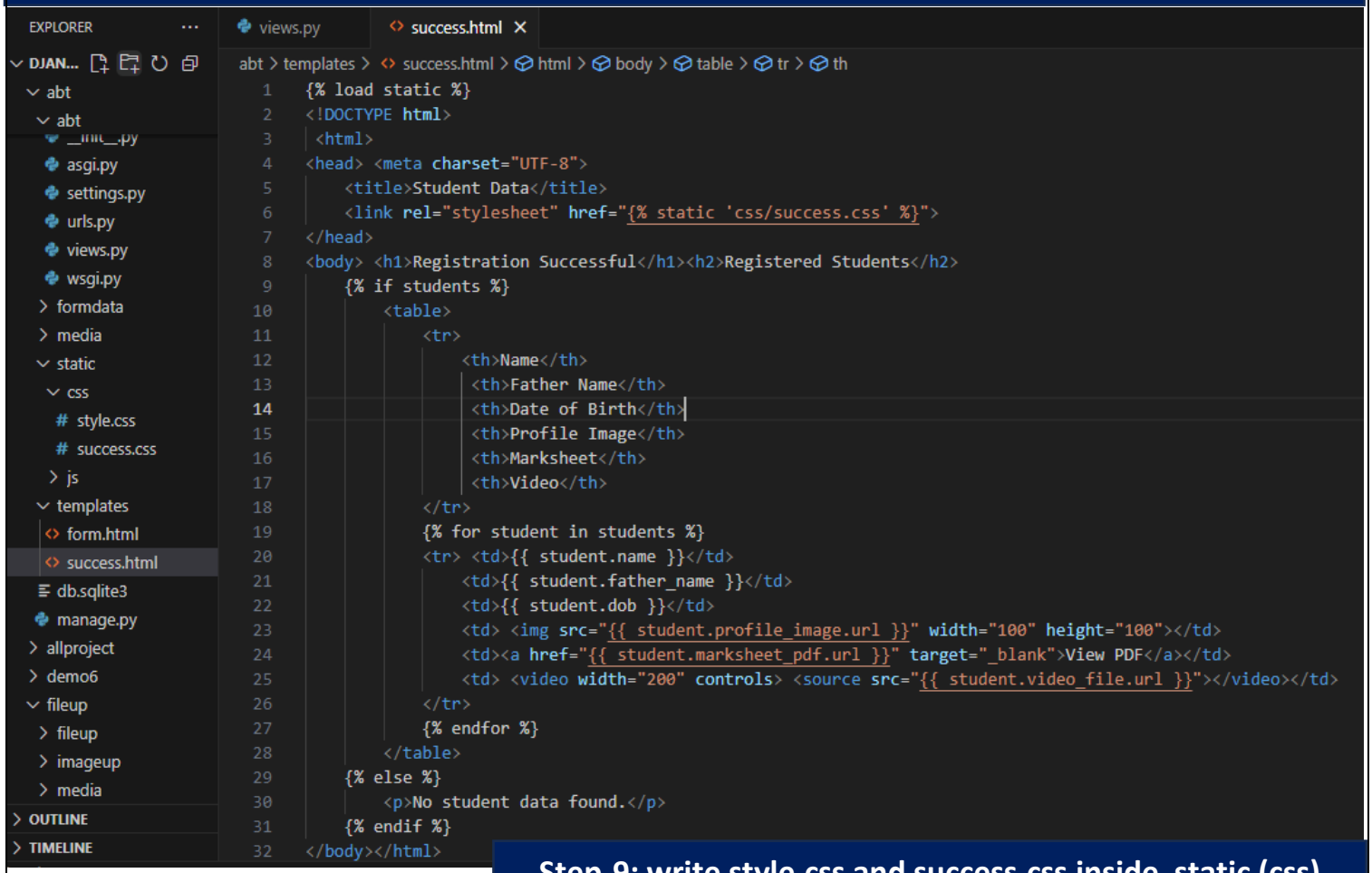
Step-6: How to data from database and show success.html



Step-7: write urls.py to redirect to the page

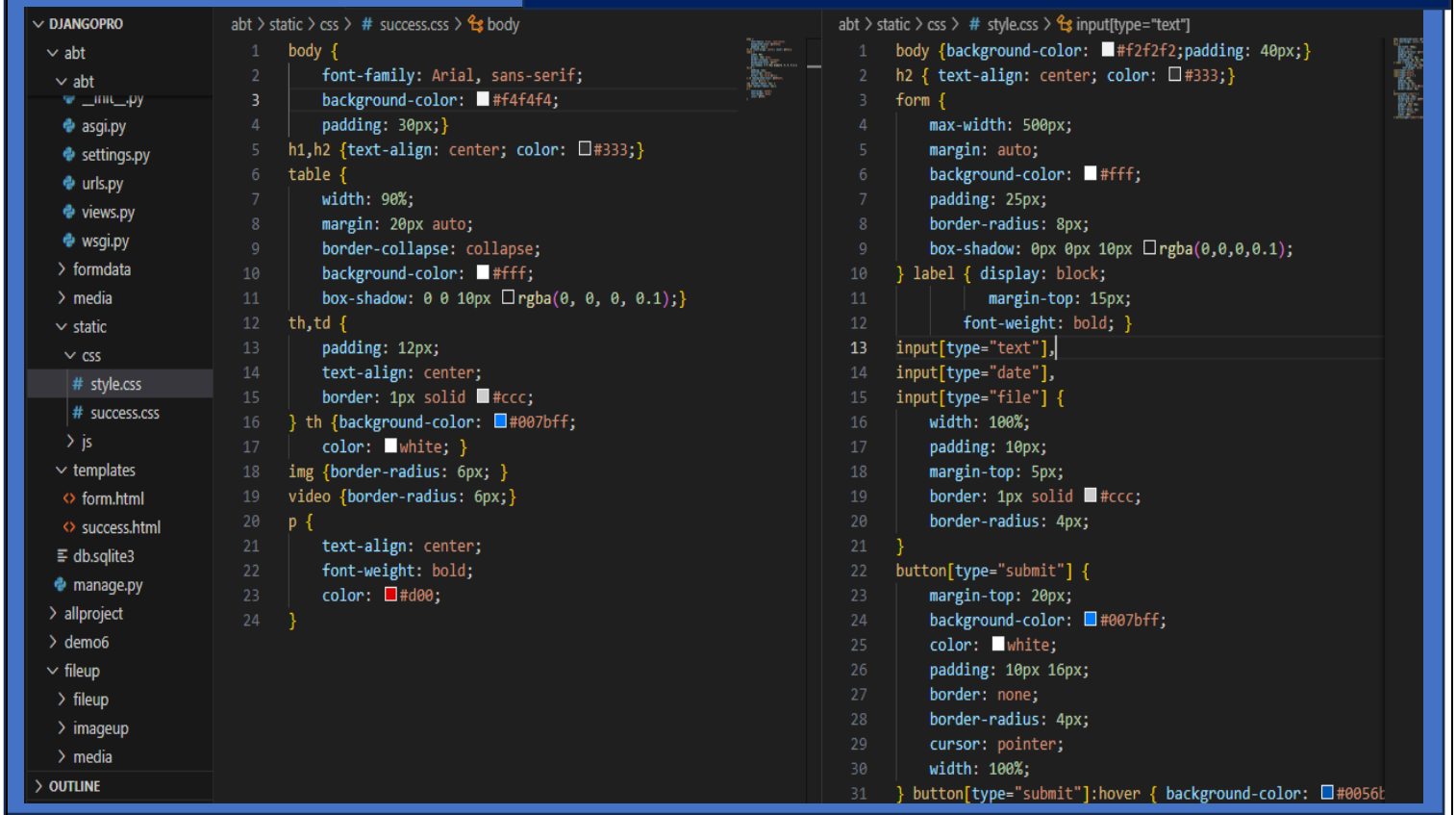


Step-8: write success.htm inside templates



```
{% load static %}
<!DOCTYPE html>
<html>
<head> <meta charset="UTF-8">
    <title>Student Data</title>
    <link rel="stylesheet" href="{% static 'css/success.css' %}">
</head>
<body> <h1>Registration Successful</h1><h2>Registered Students</h2>
    {% if students %}
        <table>
            <tr>
                <th>Name</th>
                <th>Father Name</th>
                <th>Date of Birth</th>
                <th>Profile Image</th>
                <th>Marksheet</th>
                <th>Video</th>
            </tr>
            {% for student in students %}
            <tr> <td>{{ student.name }}</td>
                <td>{{ student.father_name }}</td>
                <td>{{ student.dob }}</td>
                <td> <img src="{{ student.profile_image.url }}" width="100" height="100"></td>
                <td><a href="{{ student.marksheet_pdf.url }}" target="_blank">View PDF</a></td>
                <td> <video width="200" controls> <source src="{{ student.video_file.url }}"></video></td>
            </tr>
            {% endfor %}
        </table>
    {% else %}
        <p>No student data found.</p>
    {% endif %}
</body></html>
```

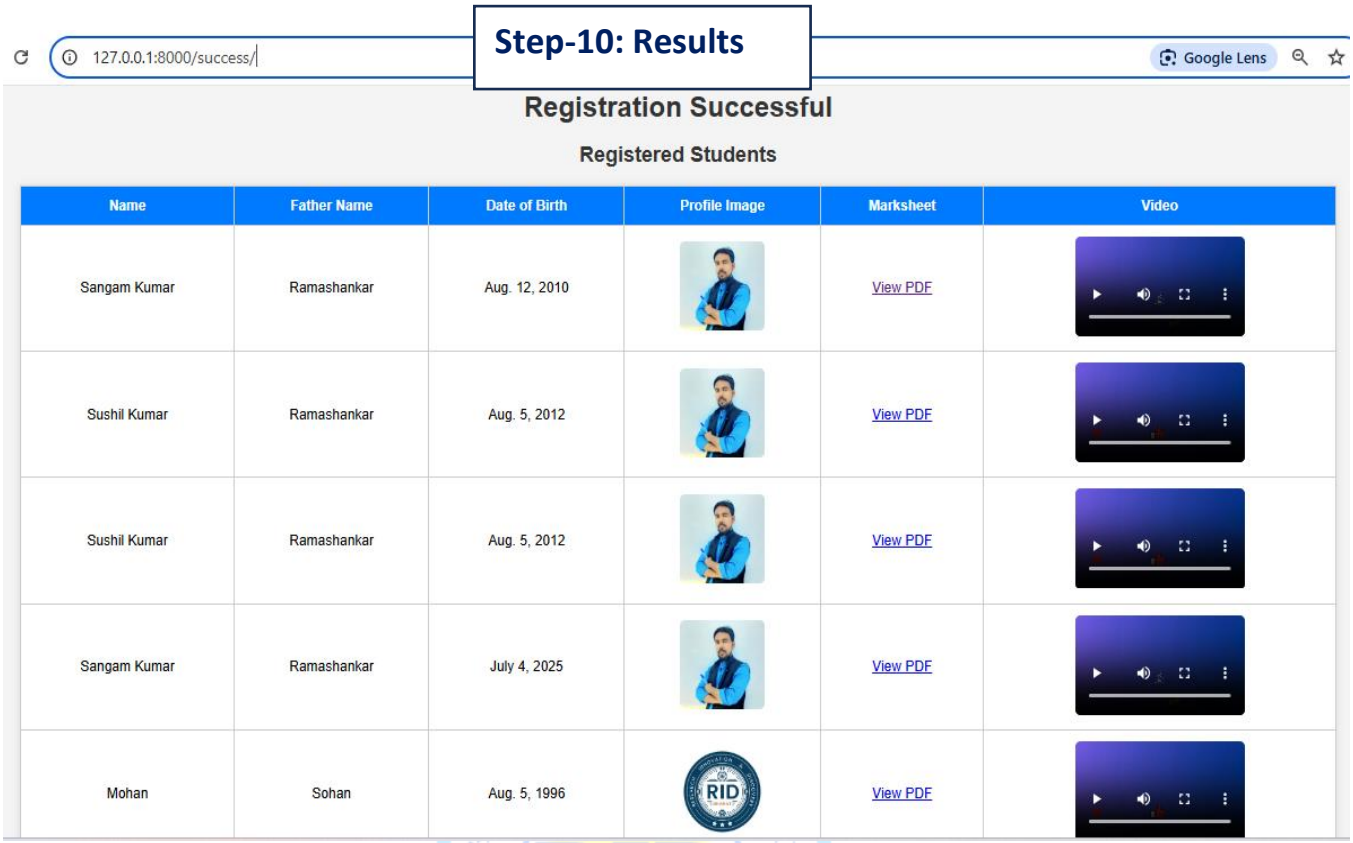
Step-9: write style.css and success.css inside static (css)



```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    padding: 30px;}
h1,h2 {text-align: center; color: #333;}
table {
    width: 90%;
    margin: 20px auto;
    border-collapse: collapse;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);}
th,td {
    padding: 12px;
    text-align: center;
    border: 1px solid #ccc;
} th {background-color: #007bff;
    color: white; }
img {border-radius: 6px; }
video {border-radius: 6px;}
p {
    text-align: center;
    font-weight: bold;
    color: #d00;
}
```

```
body {background-color: #f2f2f2;padding: 40px;}
h2 { text-align: center; color: #333;}
form {
    max-width: 500px;
    margin: auto;
    background-color: #fff;
    padding: 25px;
    border-radius: 8px;
    box-shadow: 0px 0px 10px rgba(0,0,0,0.1);
} label { display: block;
          margin-top: 15px;
        font-weight: bold; }
input[type="text"],
input[type="date"],
input[type="file"] {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
button[type="submit"] {
    margin-top: 20px;
    background-color: #007bff;
    color: white;
    padding: 10px 16px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
} button[type="submit"]:hover { background-color: #0056b
```

Step-10: Results



Default database are connected

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Change database with MySQL

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'riddb', # your MySQL DB name
      'USER': 'root',   # your MySQL user
      'PASSWORD': 'raj12345', # your MySQL password
      'HOST': 'localhost',
      'PORT': '3306',
   }
```

Django project command for connecting "Frontend+Backend+Database" performing all operation

1. **Install Django**→ pip install django
2. **Create Django Project** → django-admin startproject myproject then cd myproject
3. **Run Server (to test)** → python manage.py runserver
4. **Create Django App** → python manage.py startapp myapp
5. **Add App in settings.py** → INSTALLED_APPS = [
   ...,
   'myapp']
6. **Create templates in main project folder add in setting.py**
   TEMPLATES = [
   { ....,
   'DIRS': [BASE_DIR, "templates"]
7. **Create static folder in main project for write css and js code and add this below in setting.py**
   import os
   STATIC_URL = '/static/'
   STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
8. **Create media folder inside main project and add these all below inside setting.py and urls.py**

   Setting.py

   MEDIA_ROOT=BASE_DIR/"media"
   MEDIA_URL="/media/ **or**
   MEDIA_URL = '/media/'
   MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

   Urls.py

```
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
    #or
MEDIA_ROOT=BASE_DIR/"media"
MEDIA_URL="/media/"
```

```
from django.contrib import admin
from django.urls import path
from . import views # from formdata import views
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),

]# ✅ This serves media files like images, PDFs, videos in development
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

9. **Design model and install Pillow for FileField and ImageField** → pip install Pillow
   After designing the model, run these two commands to create the database table:
   - python manage.py makemigrations, > python manage.py migrate
10. **Import the model and write the views.py for different method**
    Ex: - from formdata.models import Student
    def home(request):
11. **Write the front-end code inside templates show the results**
    Write this top in html code {% load static %}
    {% for i in model_class_name %}
    {{ i.key_name }}
    <img src="{{ i.photo.url }}" width="150">
    {% endfor %}
12. **admin.py** → register your model (for admin panel)

Django + MySQL + Admin Panel Data Insertion

Step-1: - Create the Django Project → Ex: Django-admin startproject project_name

Step-2: - Install Required Packages→ pip install mysqlclient If error occurs run this → pip install pymysql

STEP 3: Configure settings.py for MySQL → download mysql in your system

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'testdb',                    # your MySQL DB name
    'USER': 'root',                      # your MySQL user
    'PASSWORD': 'yourpassword',          # your MySQL password
    'HOST': 'localhost',
    'PORT': '3306', }}
```

Note: If using pymysql, add this in:

- File: myproject/__init__.py
  - import pymysql
  - pymysql.install_as_MySQLdb()

STEP 4: Create the MySQL Database

- Login to MySQL and run:
- CREATE DATABASE testdb;

STEP 5: Create App for model and database

- python manage.py startapp myapp
- Add 'myapp' in INSTALLED_APPS inside settings.py

STEP 6: Create Model

- **File: myapp/models.py**

```
from django.db import models
class Student(models.Model):
  name = models.CharField(max_length=100)
  email = models.EmailField()
```

STEP 7: Run this two Migrations command for change databased structure

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

STEP 8: Register Model in Admin

- **File: myapp/admin.py**

```
from django.contrib import admin
from .models import Student
class StudentAdmin(admin.ModelAdmin):
  list_display = ('id', 'name', 'email')  # Columns to display in admin list view
admin.site.register(Student, StudentAdmin)
```

STEP 9: Create Admin User → python manage.py createsuperuser

- Follow the prompt to enter username, email, and password.

STEP 10: Run Server → python manage.py runserver

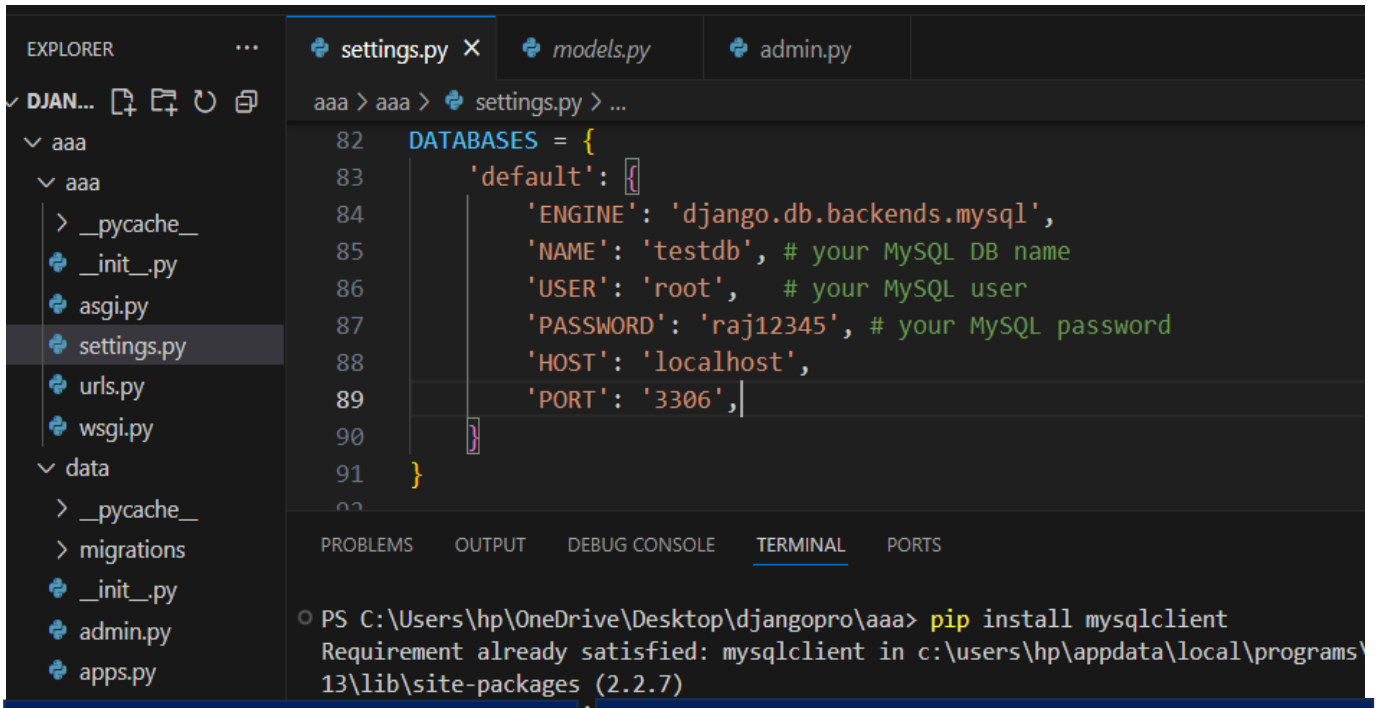
STEP 11: Open Admin Panel → http://127.0.0.1:8000/admin/ → Login with the superuser credentials.

STEP 12: Add Data via Admin → Click on **Students → Click Add Student → Click Save**

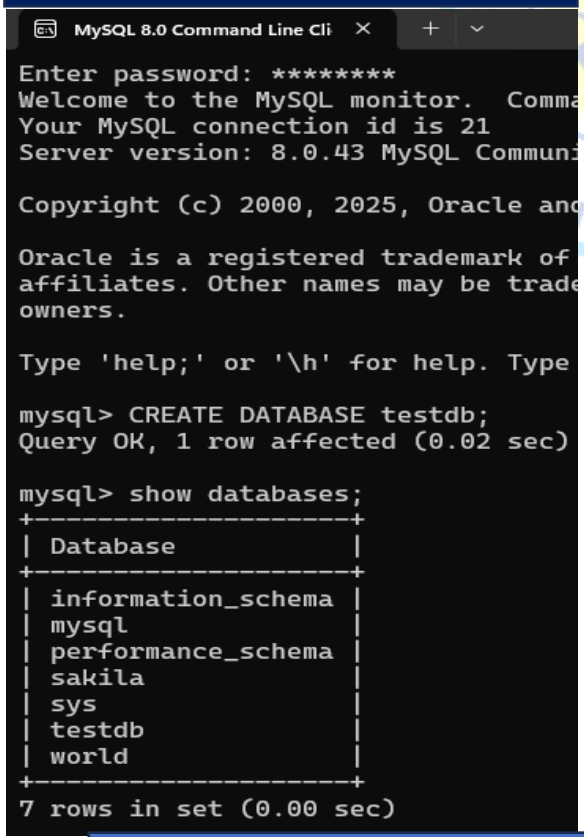
Step 13: Data will now be stored directly in the **MySQL database → open mysql database and see data.**

Django + MySQL + Admin Panel Data Connection

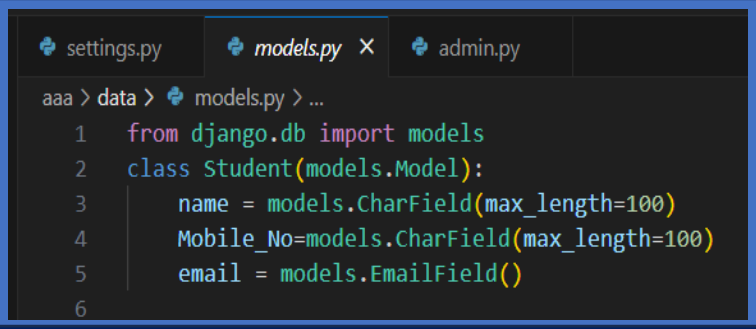
Example-1: Step-1,2,3



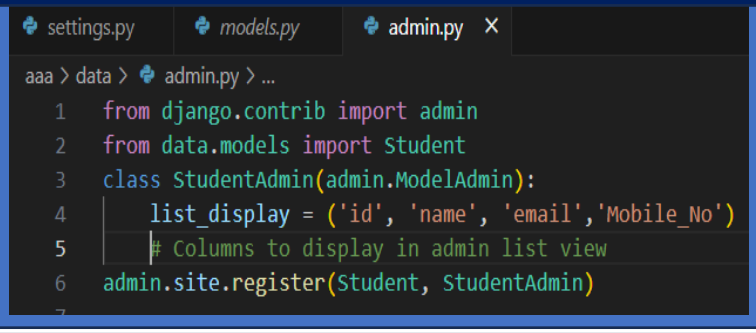
Step-4: create database in mysql



Step-5,6,7: Create App for model and run migrations



Step-8: Register Model in Admin for add data



STEP 9: Create Admin User → python manage.py createsuperuser

- Follow the prompt to enter username, email, and password.

STEP 10: Run Server → python manage.py runserver
STEP 11: Open Admin Panel → http://127.0.0.1:8000/admin/ → Login with the superuser credentials.
STEP 12: Add Data via Admin → Click on **Students** → **Click Add Student** → **Click Save**
Step 13: Data will now be stored directly in the **MySQL database** → **open mysql database and see data.**

Step-13: Show data in myself shell

```
mysql> use testdb;
Database changed
mysql> show tables;
+----------------------------+
| Tables_in_testdb           |
+----------------------------+
| auth_group                 |
| auth_group_permissions     |
| auth_permission            |
| auth_user                  |
| auth_user_groups           |
| auth_user_user_permissions |
| data_student               |
| django_admin_log           |
| django_content_type        |
| django_migrations          |
| django_session             |
+----------------------------+
11 rows in set (0.00 sec)

mysql> SELECT *FROM data_student;
+----+---------------+------------+-----------------+
| id | name          | Mobile_No  | email           |
+----+---------------+------------+-----------------+
|  1 | Sangam Kumar  | 9892782728 | raj@gmail.com   |
|  2 | Rajesh Prasad | 9892782728 | raja@gmail.com  |
+----+---------------+------------+-----------------+
2 rows in
```

View Django Data in MySQL Workbench (Short Steps)

- **1. Open MySQL Workbench**

Launch it from your system.

- **2. Connect to Server**

Click your connection (e.g., Local instance MySQL80)

Enter your MySQL username and password.

- **3. Find Django Database**

Check the DB name in `settings.py` → `'NAME': 'testdb'`

Find `testdb` under Schemas in Workbench.

- **4. Expand and Open Tables**

Click ➕ next to `testdb` → Click Tables

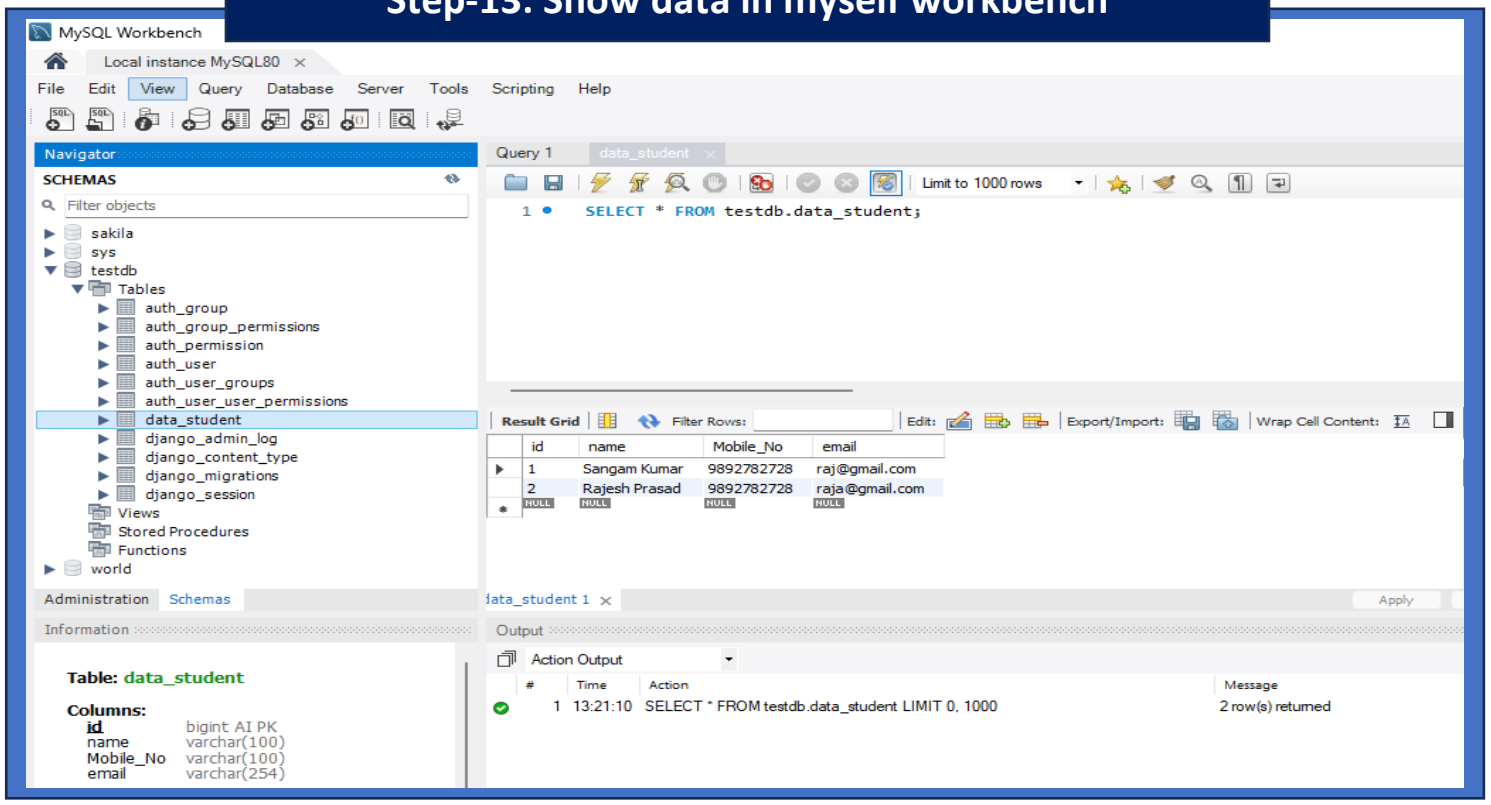
You'll see tables like `myapp_student`, `auth_user`, etc.

- **5. View Table Data**

Right-click a table (e.g., `myapp_student`) → Select "Select Rows - Limit 1000"

✔ See your Django data stored in MySQL.

Step-13: Show data in myself workbench



How to save data from Registration form to MySQL Database

Step-1: Create Django project→ django-admin startproject mission
cd mission

Step-2: Create App for database→ python manage.py startapp alldata

Step-3: create three folders in main project→ mkdir templates, mkdir static, media

```
alldata
media
mission
static
templates
manage.py
```

Step-4: add app and templates in setting.py

```
INSTALLED_APPS = [
        ......
        "alldata"
        ]
```

```
TEMPLATES = [
        BACKEND':
        ...,
    'DIRS': [BASE_DIR, "templates"]
```

Step-5: add static and media folder inside setting.py

```
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
    #or
# MEDIA_ROOT=BASE_DIR/"media"
# MEDIA_URL="/media/
```

Change database in setting.py

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'riddb', # your MySQL DB name
      'USER': 'root',   # your MySQL user
      'PASSWORD': 'raj12345', # your MySQL password
      'HOST': 'localhost',
      'PORT': '3306',
    }
}
```

```
INSTALLED_APPS = [
    ..........
    'formdata'
]
```

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

settings.py × views.py urls.py

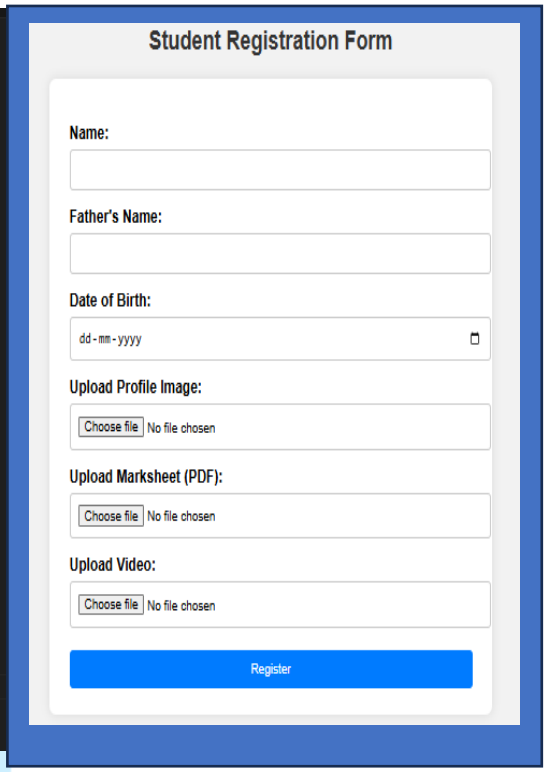
abt > abt > settings.py > ...

```
118 STATIC_URL = 'static/'
119 import os
120 # Static files (CSS, JS)
121 STATIC_URL = '/static/'
122 STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
123 # Default primary key field type
124 # https://docs.djangoproject.com/en/5.2/ref/settings/#de
125 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
126 #add this when you are working on image and ohers file
127 MEDIA_ROOT=BASE_DIR/"media"
128 MEDIA_URL="/media/"
```

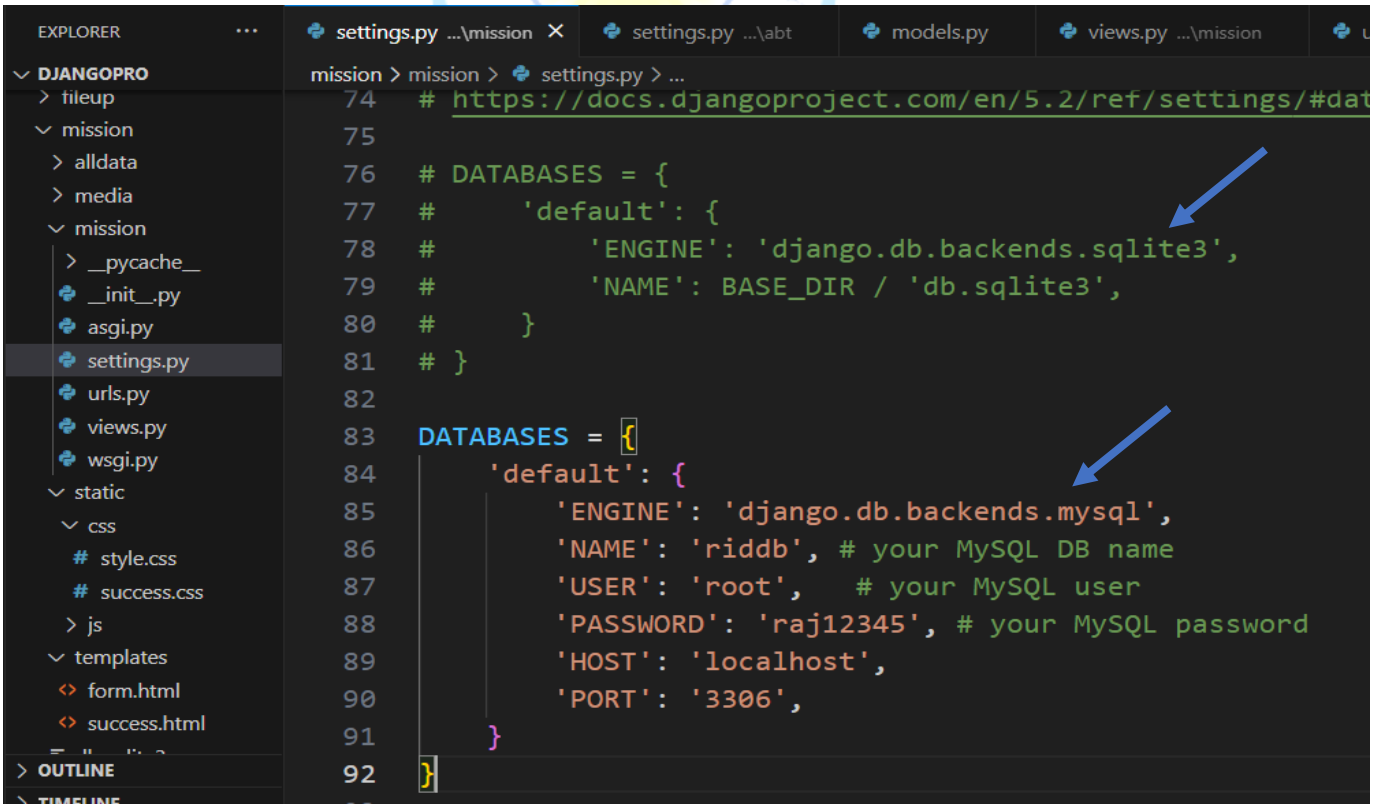
Step-6: Install Required Packages→ pip install mysqlclient If error occurs run this → pip install pymysql

Step-7: create from.html inside template

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Student Registration</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <h2>Student Registration Form</h2>
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        <label for="id_name">Name:</label>
        <input type="text" name="name" id="id_name" required>
        <label for="id_father_name">Father's Name:</label>
        <input type="text" name="father_name" id="id_father_name" required>
        <label for="id_dob">Date of Birth:</label>
        <input type="date" name="dob" id="id_dob" required>
        <label for="id_profile_image">Upload Profile Image:</label>
        <input type="file" name="profile_image" id="id_profile_image" accept="image/*" required>
        <label for="id_marksheet_pdf">Upload Marksheet (PDF):</label>
        <input type="file" name="marksheet_pdf" id="id_marksheet_pdf" accept="application/pdf" required>
        <label for="id_video_file">Upload Video:</label>
        <input type="file" name="video_file" id="id_video_file" accept="video/*" required>
        <button type="submit">Register</button>
    </form>
</body>
</html>
```



Step-8: Change database



Step-8: Create the model and design the model for the store the data

models.py

abt > formdata > models.py > Student > __str__

```python
from django.db import models
class Student(models.Model):
    name = models.CharField(max_length=100)
    father_name = models.CharField(max_length=100)
    dob = models.DateField()
    # Profile image file, saved to 'media/profile_images folder/'
    profile_image = models.ImageField(upload_to='profile_images/')
    marksheet_pdf = models.FileField(upload_to='marksheets/')
    # Uploaded video file, saved to 'media/videos/
    video_file = models.FileField(upload_to='videos/')
```

Run this command → Python manage.py makemigrations and python manage.py migrate

Without Pillow, Django can't handle uploaded image files.

Why Pillow is Required? ImageField (used in profile_image = models.ImageField(...)) needs Pillow to:

- Validate image format, Handle image dimensions and file headers, Save and process images

Solution: Install Pillow Run this command in your terminal or command prompt:

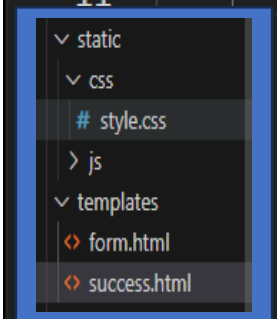
- **python -m pip install Pillow**:- This will install the required image processing library that Django uses internally for ImageField.

Step-9: write the views for access the data from the form and store in database

views.py

abt > abt > views.py > home

```python
from django.shortcuts import render
from formdata.models import Student
def home(request):
    if request.method == 'POST':
        name_1 = request.POST.get('name')
        father_name_1 = request.POST.get('father_name')
        dob_1 = request.POST.get('dob')
        profile_image_1 = request.FILES.get('profile_image')
        marksheet_pdf_1 = request.FILES.get('marksheet_pdf')
        video_file_1 = request.FILES.get('video_file')
        # Save to database
        Student.objects.create(
            name=name_1,
            father_name=father_name_1,
            dob=dob_1,
            profile_image=profile_image_1,
            marksheet_pdf=marksheet_pdf_1,
            video_file=video_file_1
        )
        return render(request, 'success.html')
    return render(request, 'form.html')
```



Urls.py

urls.py

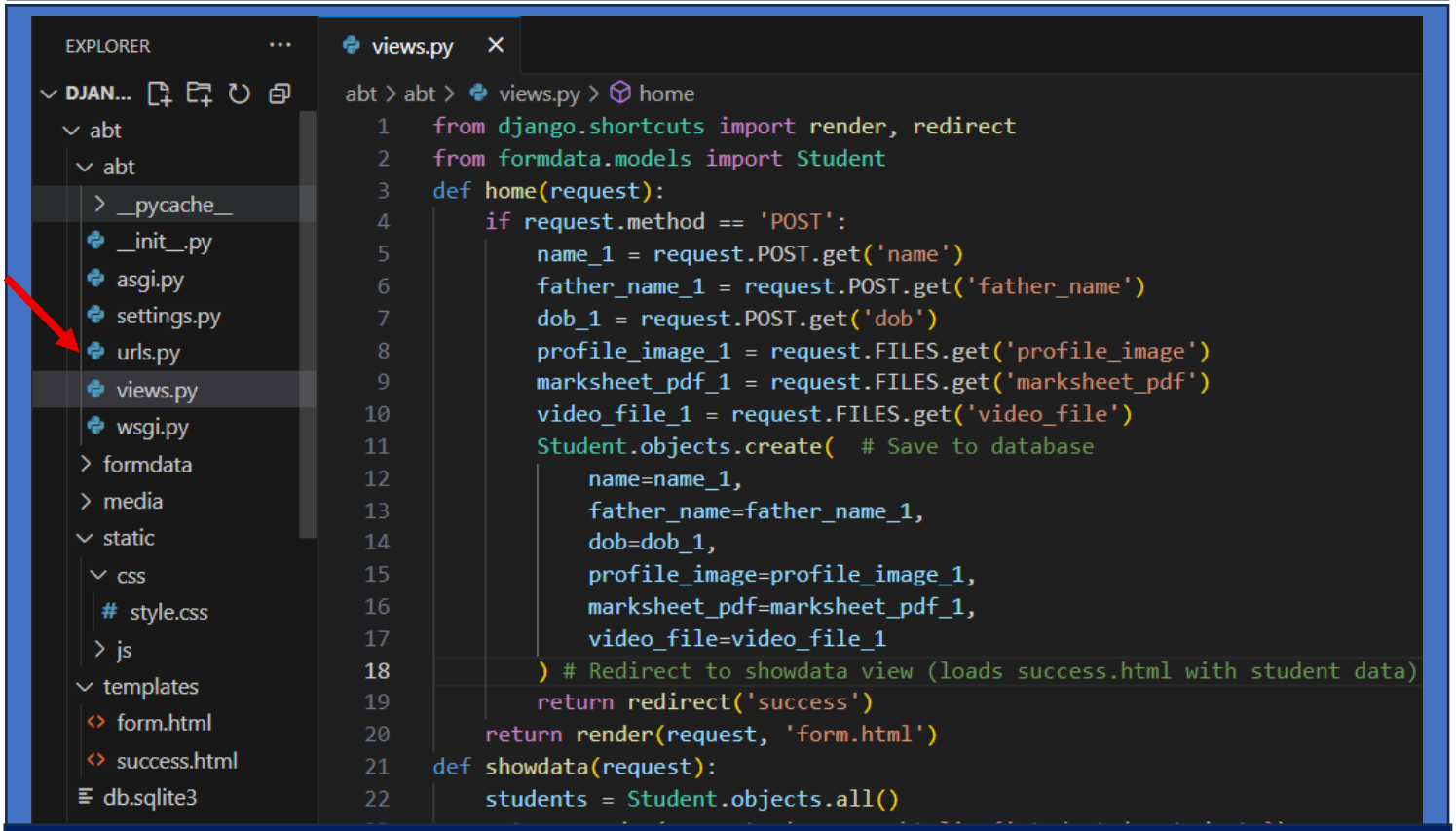
abt > abt > urls.py > ...

```python
from django.contrib import admin
from django.urls import path
from .import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.home)
]
```

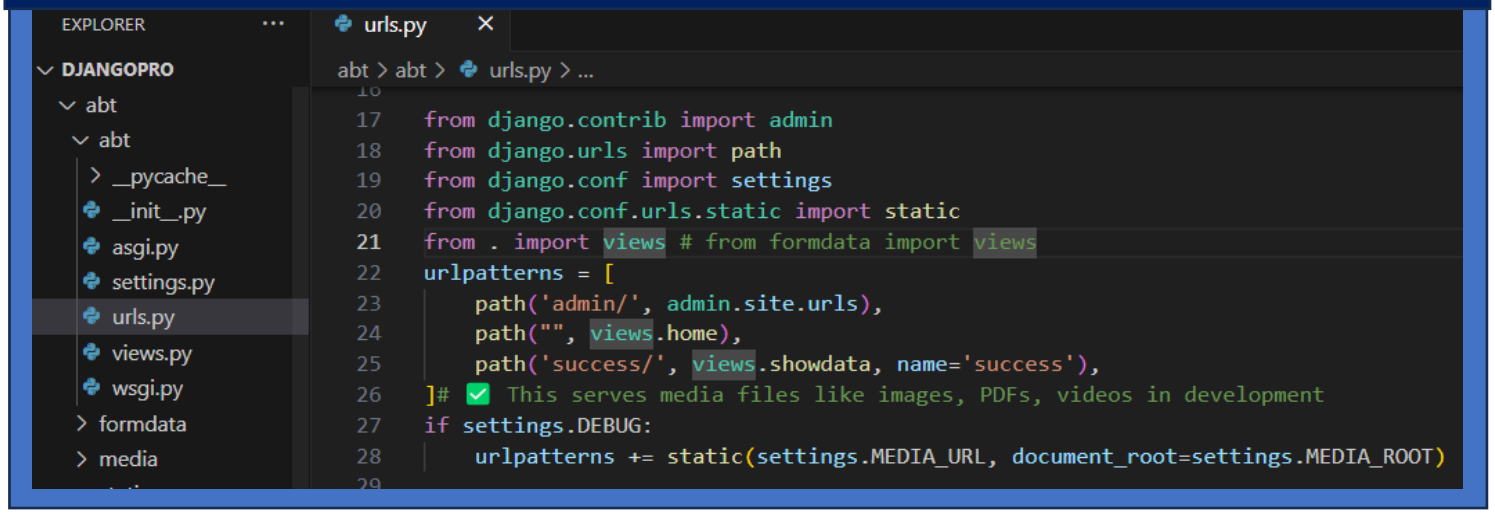
Step-5: Open Mysql database and see the data

```
mysql> SELECT *FROM  alldata_student;
+----+--------------+------------+------------+---------------------+-----------------------------+-----------------------------+
| id | name         | father_name | dob        | profile_image       | marksheet_pdf               | video_file                  |
+----+--------------+------------+------------+---------------------+-----------------------------+-----------------------------+
|  1 | Sangam Kumar | Ramashankar | 2025-07-29 | profile_images/rpsir.jpg | marksheets/t3_duniya__UtlynuO.pdf | videos/END_VIDEO_CLIP_J9qNDLl.mp4 |
+----+--------------+------------+------------+---------------------+-----------------------------+-----------------------------+
1 row in set (0.00 sec)
```

Step-6: How to access data from database and show success.html



Step-7: write urls.py to redirect to the page



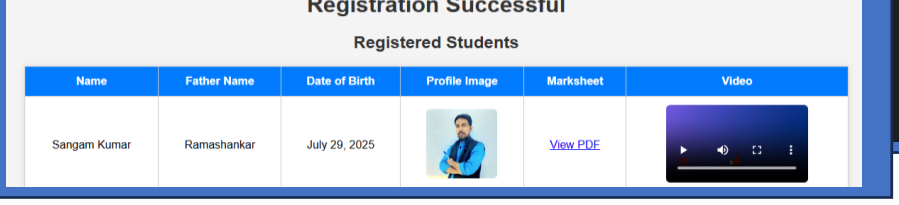
Step-8: write success.htm inside templates

```
{% load static %}
<!DOCTYPE html>
<html>
<head> <meta charset="UTF-8">
    <title>Student Data</title>
    <link rel="stylesheet" href="{% static 'css/success.css' %}">
</head>
<body> <h1>Registration Successful</h1><h2>Registered Students</h2>
    {% if students %}
        <table>
            <tr>
                <th>Name</th>
                <th>Father Name</th>
                <th>Date of Birth</th>
                <th>Profile Image</th>
                <th>Marksheet</th>
                <th>Video</th>
            </tr>
            {% for student in students %}
            <tr> <td>{{ student.name }}</td>
                <td>{{ student.father_name }}</td>
                <td>{{ student.dob }}</td>
                <td> <img src="{{ student.profile_image.url }}" width="100" height="100"></td>
                <td><a href="{{ student.marksheet_pdf.url }}" target="_blank">View PDF</a></td>
                <td> <video width="200" controls> <source src="{{ student.video_file.url }}"></video></td>
            </tr>
            {% endfor %}
        </table>
    {% else %}
        <p>No student data found.</p>
    {% endif %}
</body></html>
```

Step-9: write style.css and success.css inside static (css)

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    padding: 30px;}
h1,h2 {text-align: center; color: #333;}
table {
    width: 90%;
    margin: 20px auto;
    border-collapse: collapse;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);}
th,td {
    padding: 12px;
    text-align: center;
    border: 1px solid #ccc;
} th {background-color: #007bff;
    color: white; }
img {border-radius: 6px; }
video {border-radius: 6px;}
p {
    text-align: center;
    font-weight: bold;
    color: #d00;
}
```

```
body {background-color: #f2f2f2;padding: 40px;}
h2 { text-align: center; color: #333;}
form {
    max-width: 500px;
    margin: auto;
    background-color: #fff;
    padding: 25px;
    border-radius: 8px;
    box-shadow: 0px 0px 10px rgba(0,0,0,0.1);
} label { display: block;
          margin-top: 15px;
        font-weight: bold; }
input[type="text"],
input[type="date"],
input[type="file"] {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
button[type="submit"] {
    margin-top: 20px;
    background-color: #007bff;
    color: white;
    padding: 10px 16px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
} button[type="submit"]:hover { background-color: #0056b
```



How to connect the MongoDB with Django from admin panel

- To connect **MongoDB** with **Django** and add data from the **admin panel**, follow this **step-by-step guide**. Since Django officially supports SQL databases (like MySQL, PostgreSQL), we'll use **djongo** or **mongoengine** to connect with MongoDB.
  - We'll use **djongo** here because it integrates MongoDB with Django's ORM and admin panel.
  - pip install --upgrade Django
  - pip install django==4.2.4

Step-by-Step: Connect MongoDB with Django Using djongo

1. Install Required Packages

- pip install djongo
- pip install pymongo

How to uninstall any package
➔ pip uninstall pymysql

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\raj> pip list
Package          Version
---------------- -------
asgiref          3.8.1
Django           5.2.4
django-autoslug  1.9.9
django-tinymce   4.1.0
djongo           1.3.7
mysqlclient      2.2.7
pillow           11.3.0
pip              25.1.1
pymongo          3.11.4
pytz             2025.2
setuptools       80.9.0
sqlparse         0.5.3
tzdata           2025.2
PS C:\Users\hp\OneDrive\Desktop\djangopro\raj>
```

2. Create MongoDB Database

- Open **MongoDB Compass** or use Mongo shell:
- use mymongodb
- No need to manually create collections; Django will do it.

3. Create Django Project and App

- django-admin startproject myproject
- cd myproject
- python manage.py startapp myapp

4. Configure MongoDB in settings.py

myproject/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'mymongodb',          # your MongoDB database name
        'ENFORCE_SCHEMA': False,      # Optional
        'CLIENT': {
            'host': 'mongodb://localhost:27017',
        }
    }}
```

For connecting MongoDb python does not support 3.13 version so use 3.12

Use djongo + MongoDB Atlas + Django 3.2 + Python 3.10/3.11

5. Add App in INSTALLED_APPS

```
INSTALLED_APPS = [
    ...,
    'myapp',
]
```

6. Create Model

myapp/models.py

```
from django.contrib import admin
from data.models import regdata
class regdataAdmin(admin.ModelAdmin):
    list_display = ('name', 'mobile_No', 'email')
    # Show these fields in the admin list view
admin.site.register(regdata, regdataAdmin)
```

models.py

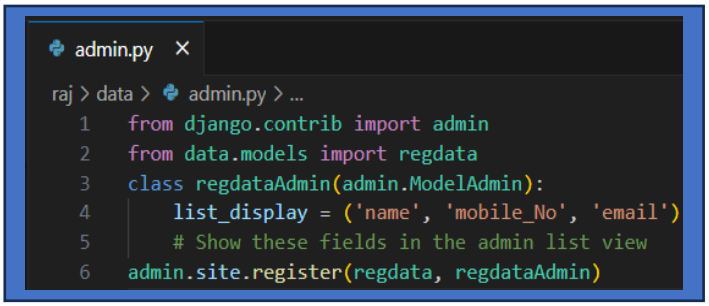
raj > data > models.py > ...

```
1  from django.db import models
2  class regdata(models.Model):
3      name = models.CharField(max_length=100)
4      mobile_No=models.CharField(max_length=100)
5      email = models.EmailField()
6
```

7. Register Model in Admin

myapp/admin.py

```
from django.contrib import admin
from data.models import regdata
class regdataAdmin(admin.ModelAdmin):
   list_display = ('name', 'mobile_No', 'email')
   # Show these fields in the admin list view
admin.site.register(regdata, regdataAdmin)
```



8. Make Migrations and Migrate

- python manage.py makemigrations
- python manage.py migrate
- djongo translates migrations to MongoDB collections.

9. Create Superuser: → python manage.py createsuperuser

10. Run Server and Open Admin Panel

- python manage.py runserver
- Go to http://127.0.0.1:8000/admin/
- Login with your superuser
- Click on **Students** → **Add Student**
- Fill and Save — Data goes to **MongoDB**

11. View Data in MongoDB

- Open **MongoDB Compass** or shell
- Check database: mymongodb
- Collection: myapp_student

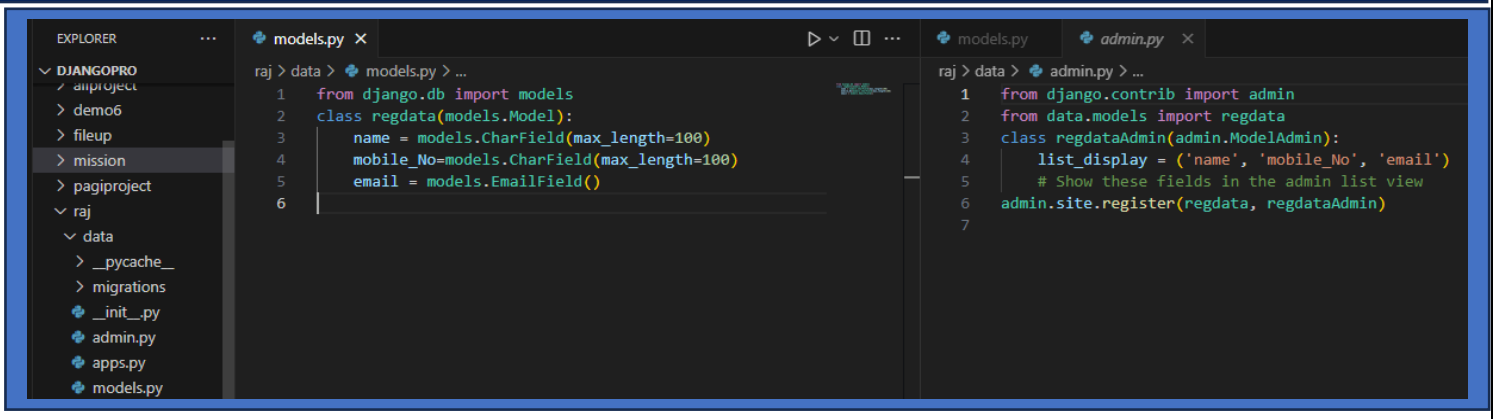
Summary

Step	Command/Action
Install	pip install djongo pymongo
DB Config	settings.py → DATABASES
Model	Define in models.py
Admin	Register in admin.py
Migrate	makemigrations, migrate
Add Data	Through Django admin panel
View Data	In MongoDB Compass or shell

Basic MongoDB Command

Task	Command
Run MongoDB shell	mongo (or mongosh for latest)
Show all databases	show databases
Use a database (switch or create)	use database_name e.g., use riddb
Show all collections	show collections
Show all data in a collection	db.collection_name.find()
Pretty print output	db.collection_name.find().pretty()
Limit number of results	db.collection_name.find().limit(5)

Example: Django connecting with MongoDB database

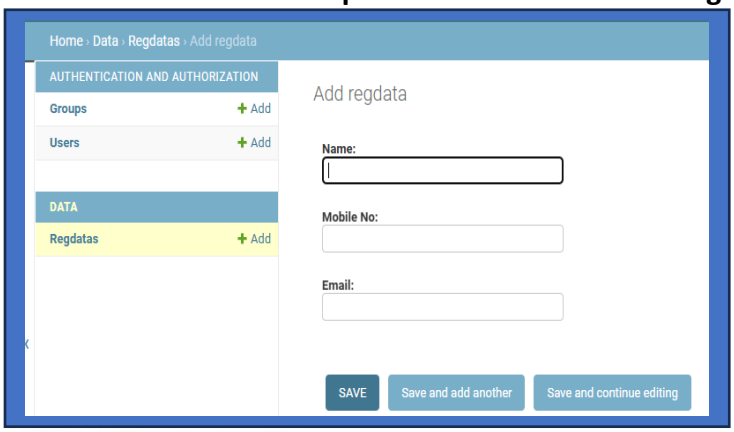


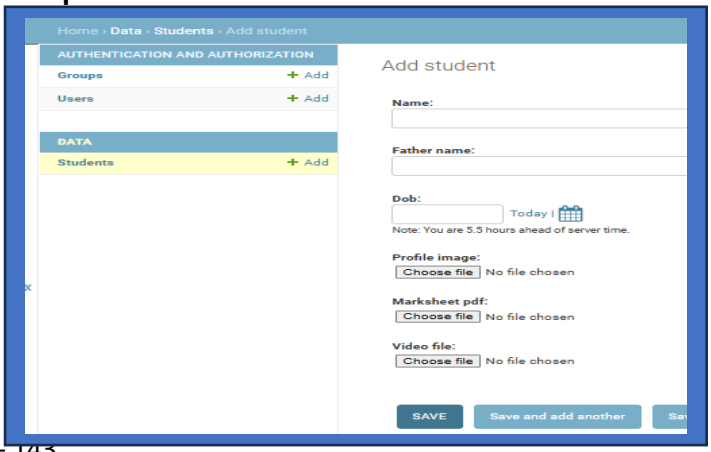
Setting.py change the database

Connecting Django with MongoDB Atlas

```
import mongoengine
mongoengine.connect(
   db='your_database_name',
host='mongodb+srv://<username>:<password>@cluster0.mongodb.net/your_database_name?retryWrites=true&w=majority'
)
```

➔ **Create superuser and add data through admin panel**





Database-1 without image and file

```
test> use riddb;
switched to db riddb
riddb> show collections;
__schema__
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
data_regdata
django_admin_log
django_content_type
django_migrations
django_session
riddb> db.data_regdata.find().pretty()
[
  {
    _id: ObjectId('6888f76f3dd989ad44b9a29a'),
    id: 1,
    name: 'Sangam Kumar',
    mobile_No: '9892782728',
    email: 'sangam339@gmail.com'
  },
  {
    _id: ObjectId('6888f77b3dd989ad44b9a29c'),
    id: 2,
    name: 'Rajesh Prasad',
    mobile_No: '9892782728',
    email: 'raj@gmail.com'
  }
]
riddb>
```

Database-2 with image and file

```
test> show databases;
admin     40.00 KiB
alldata  312.00 KiB
config   108.00 KiB
local     72.00 KiB
riddb    672.00 KiB
riddb1   488.00 KiB
todaydb  984.00 KiB
test> use todaydb;
switched to db todaydb
todaydb> show collections;
__schema__
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
data_student
django_admin_log
django_content_type
django_migrations
django_session
todaydb> db.data_student.find().pretty()
[
  {
    _id: ObjectId('6889c288bb2b3575c4192308'),
    id: 1,
    name: 'Sangam Kumar',
    father_name: 'Ramashankar',
    dob: ISODate('2000-05-08T00:00:00.000Z'),
    profile_image: 'profile_images/rpsir.jpg',
    marksheet_pdf: 'marksheets/t3_duniya_.pdf',
    video_file: 'videos/END_VIDEO_CLIP.mp4'
  }
]
todaydb>
```

Model.py (app name data)

models.py

nand > data > models.py > Student

```python
from django.db import models
class Student(models.Model):
    name = models.CharField(max_length=100)
    father_name = models.CharField(max_length=100)
    dob = models.DateField()
    # Profile image file, saved to 'media/profile_images folder/'
    profile_image = models.ImageField(upload_to='profile_images/')
    marksheet_pdf = models.FileField(upload_to='marksheets/')
    # Uploaded video file, saved to 'media/videos/
    video_file = models.FileField(upload_to='videos/')
```

Admin.py (inside app)

models.py | admin.py

nand > data > admin.py > ...

```python
from django.contrib import admin
from .models import Student  # adjust the imp

class StudentAdmin(admin.ModelAdmin):
    list_display = (
        'name',
        'father_name',
        'dob',
        'profile_image',
        'marksheet_pdf',
        'video_file',
    )
    search_fields = ('name', 'father_name')
    list_filter = ('dob',)

admin.site.register(Student, StudentAdmin)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\hp\OneDrive\Desktop\djangopro\nand> python manage.py makemigrations
Migrations for 'data':
  data\migrations\0001_initial.py
    - Create model Student
PS C:\Users\hp\OneDrive\Desktop\djangopro\nand> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, data, sessions
Running migrations:
  Applying data.0001_initial...This version of djongo does not support "NULL, NOT
ongo/support/
 OK
PS C:\Users\hp\OneDrive\Desktop\djangopro\nand> python manage.py createsuperuser
Username (leave blank to use 'hp'): raj
Email address: raj@gmail.com
Password:
Password (again):
Superuser created successfully.
PS C:\Users\hp\OneDrive\Desktop\djangopro\nand> python manage.py migrate
```

settings.py

nand > nand > settings.py > ...

```python
DATABASES = {
    'default': {
        'ENGINE': 'djongo',
        'NAME': 'todaydb',
        'ENFORCE_SCHEMA': False,
        'CLIENT': {
            'host': 'mongodb://localhost:27017',
        }
    }
}
```

settings.py

nand > nand > settings.py > ...

```python
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

How to save & display data from Registration form to MongoDB

Step-1: Create Django project→ django-admin startproject RegFrom → cd mission
Step-2: Create App for database→ python manage.py startapp alldata
Step-3: create three folders in main project→ mkdir templates, mkdir static, media
Step-4: add app and templates in setting.py

```
INSTALLED_APPS = [
        ......
       "alldata"
       ]
```

```
alldata
media
mission
static
templates
manage.py
```

```
TEMPLATES = [
        BACKEND':
          ...,
    'DIRS': [BASE_DIR, "templates"]
```

Step-5: add static and media folder inside setting.py

```
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
    #or
# MEDIA_ROOT=BASE_DIR/"media"
# MEDIA_URL="/media/
```

Change database in setting.py

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'riddb', # your MySQL DB name
    'USER': 'root',   # your MySQL user
    'PASSWORD': 'raj12345', # your MySQL password
    'HOST': 'localhost',
    'PORT': '3306',
  }
}
```

settings.py | views.py | urls.py | form.html | models.py | success.html | wsgi.py

```
118 STATIC_URL = 'static/'
119 import os
120 # Static files (CSS, JS)
121 STATIC_URL = '/static/'
122 STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
123 # Default primary key field type
124 # https://docs.djangoproject.com/en/5.2/ref/settings/#de
125 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
126 #add this when you are working on image and ohers file
127 MEDIA_ROOT=BASE_DIR/"media"
128 MEDIA_URL="/media/"
```

```
INSTALLED_APPS = [
  ..........
  'formdata'
]
```

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

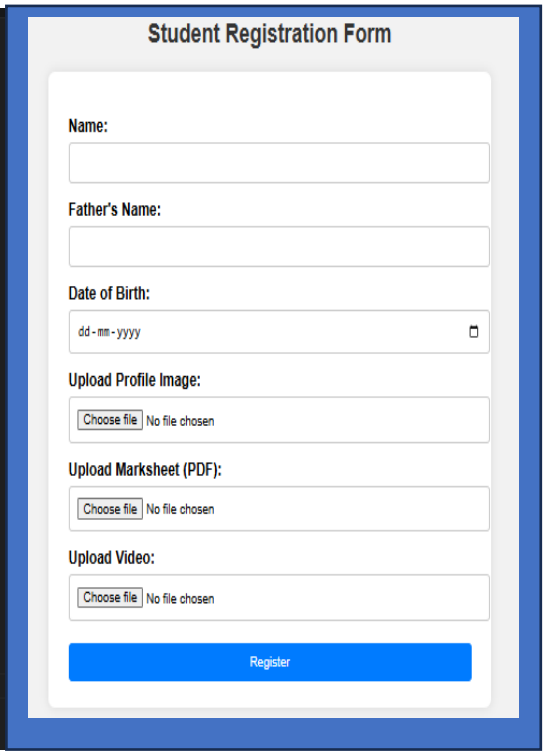
Step-6 Install these Required Packages connecting MongoDB and add file field in mode

- pip install djongo :- **Use Django models with MongoDB**
- pip install pymongo :- **Connect Python/Djongo to MongoDB**
- pip install Pillow :- **Pillow** is required for Django's ImageField to work. model uses file/image uploads

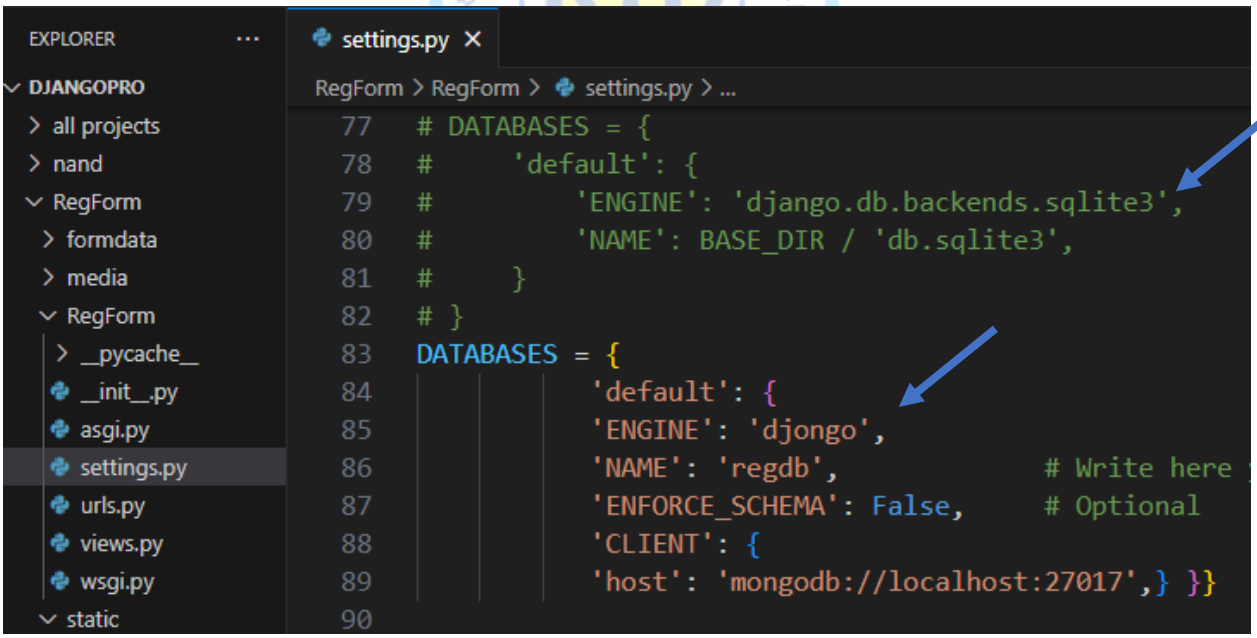
```
PS C:\Users\hp\OneDrive\Desktop\djangopro\RegForm> pip list
Package     Version
----------- -------
asgiref     3.9.1
Django      3.1.12
djongo      1.3.7
mysqlclient 2.2.7
pillow      11.3.0
pip         24.0
pymongo     3.11.4
pytz        2025.2
setuptools  65.5.0
sqlparse    0.2.4
tzdata      2025.2
```

Step-7: create from.html inside template

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Student Registration</title>
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <h2>Student Registration Form</h2>
    <form method="POST" enctype="multipart/form-data">
        {% csrf_token %}
        <label for="id_name">Name:</label>
        <input type="text" name="name" id="id_name" required>
        <label for="id_father_name">Father's Name:</label>
        <input type="text" name="father_name" id="id_father_name" required>
        <label for="id_dob">Date of Birth:</label>
        <input type="date" name="dob" id="id_dob" required>
        <label for="id_profile_image">Upload Profile Image:</label>
        <input type="file" name="profile_image" id="id_profile_image" accept="image/*" required>
        <label for="id_marksheet_pdf">Upload Marksheet (PDF):</label>
        <input type="file" name="marksheet_pdf" id="id_marksheet_pdf" accept="application/pdf" required>
        <label for="id_video_file">Upload Video:</label>
        <input type="file" name="video_file" id="id_video_file" accept="video/*" required>
        <button type="submit">Register</button>
    </form>
</body>
</html>
```



Step-8: Change database SQLite to MongoDB



Step-8: Create the model and design the model for the store the data

models.py

abt > formdata > models.py > Student > __str__

```python
from django.db import models
class Student(models.Model):
    name = models.CharField(max_length=100)
    father_name = models.CharField(max_length=100)
    dob = models.DateField()
    # Profile image file, saved to 'media/profile_images folder/'
    profile_image = models.ImageField(upload_to='profile_images/')
    marksheet_pdf = models.FileField(upload_to='marksheets/')
    # Uploaded video file, saved to 'media/videos/
    video_file = models.FileField(upload_to='videos/')
```

Run this command → Python manage.py makemigrations and python manage.py migrate

Without Pillow, Django can't handle uploaded image files.

Why Pillow is Required? ImageField (used in profile_image = models.ImageField(...)) needs Pillow to:

- Validate image format, Handle image dimensions and file headers, Save and process images

Solution: Install Pillow Run this command in your terminal or command prompt:

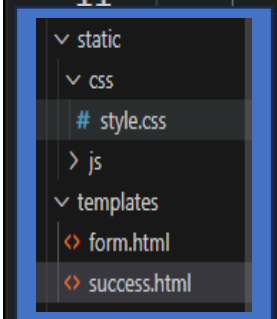
- **python -m pip install Pillow**:- This will install the required image processing library that Django uses internally for ImageField.

Step-9: write the views for access the data from the form and store in database

views.py

abt > abt > views.py > home

```python
from django.shortcuts import render
from formdata.models import Student
def home(request):
    if request.method == 'POST':
        name_1 = request.POST.get('name')
        father_name_1 = request.POST.get('father_name')
        dob_1 = request.POST.get('dob')
        profile_image_1 = request.FILES.get('profile_image')
        marksheet_pdf_1 = request.FILES.get('marksheet_pdf')
        video_file_1 = request.FILES.get('video_file')
        # Save to database
        Student.objects.create(
            name=name_1,
            father_name=father_name_1,
            dob=dob_1,
            profile_image=profile_image_1,
            marksheet_pdf=marksheet_pdf_1,
            video_file=video_file_1
        )
        return render(request, 'success.html')
    return render(request, 'form.html')
```



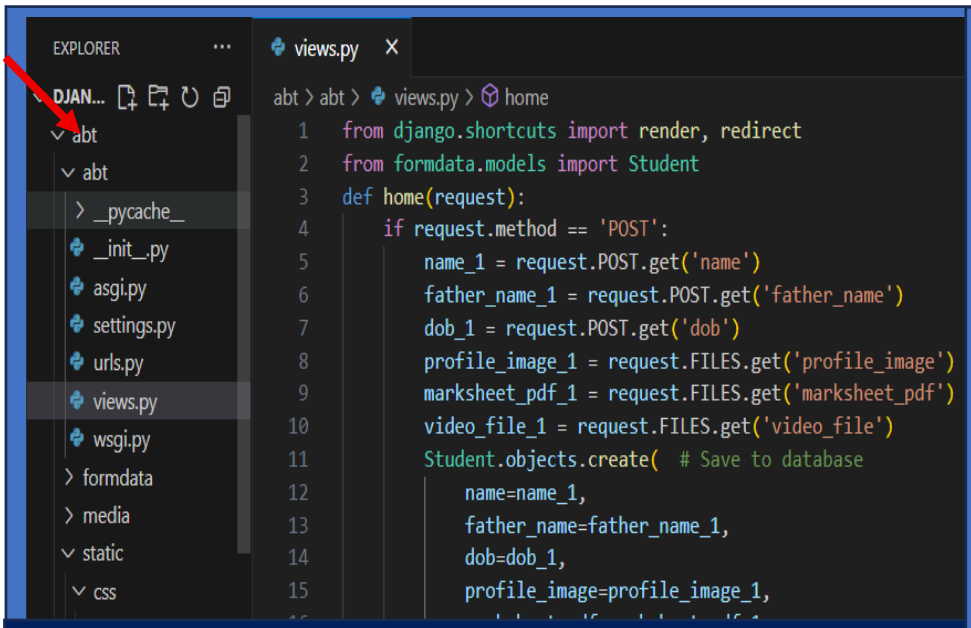
Urls.py

urls.py

abt > abt > urls.py > ...

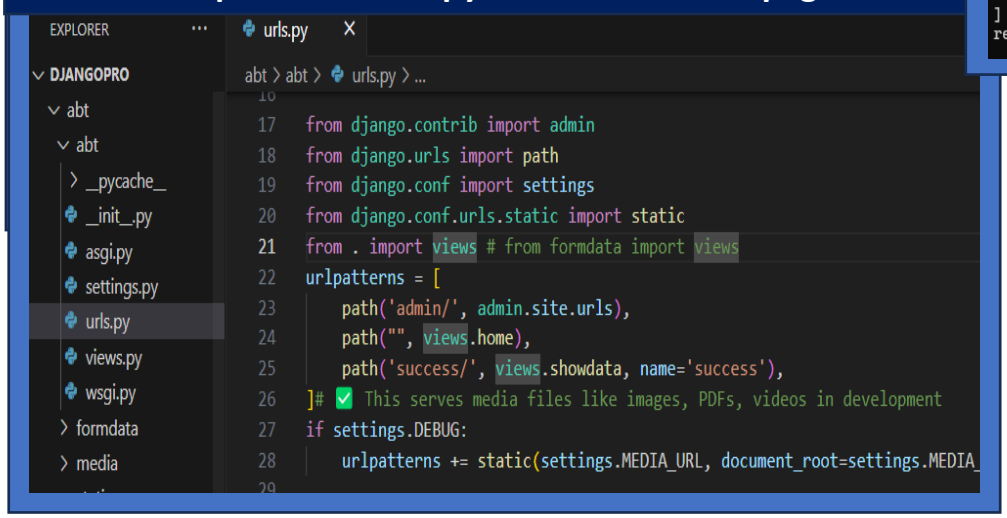
```python
from django.contrib import admin
from django.urls import path
from .import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.home)
]
```

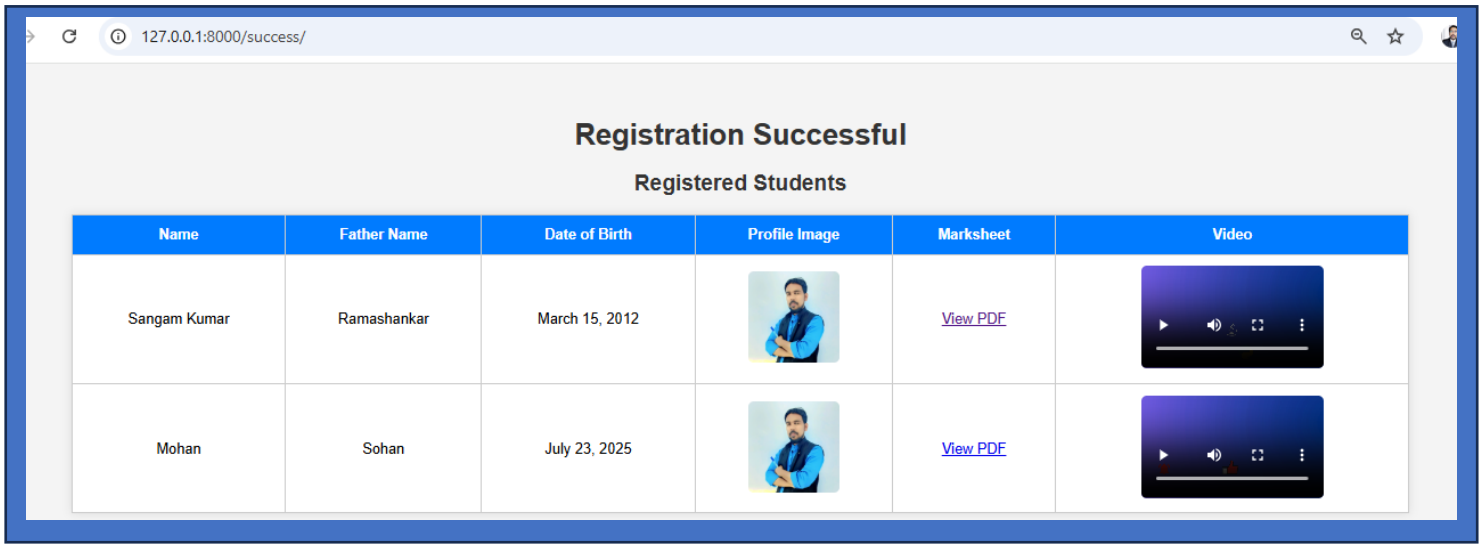
Step-5,6: How to access data from database and show success.html, Open MongoDB database and see the data



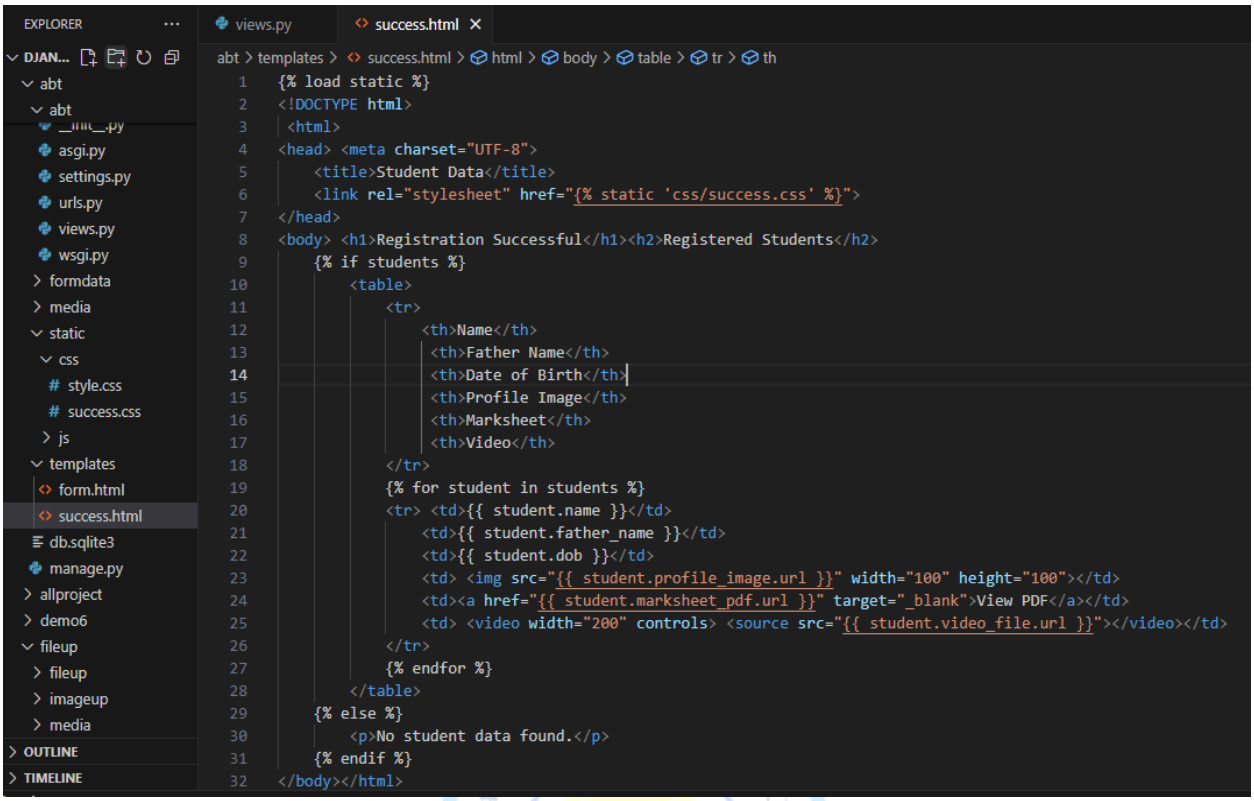

```
regdb> show collections;
__schema__
auth_group
auth_group_permissions
auth_permission
auth_user
auth_user_groups
auth_user_user_permissions
django_admin_log
django_content_type
django_migrations
django_session
formdata_student
regdb> db.formdata_student.find().pretty()
[
  {
    _id: ObjectId('6889e047df4b3f14fe2cd18a'),
    id: 1,
    name: 'Sangam Kumar',
    father_name: 'Ramashankar',
    dob: ISODate('2012-03-15T00:00:00.000Z'),
    profile_image: 'profile_images/rpsir.jpg',
    marksheet_pdf: 'marksheets/t3_duniya_.pdf',
    video_file: 'videos/START_VIDEO_CLIP.mp4'
  },
  {
    _id: ObjectId('6889e083df4b3f14fe2cd18b'),
    id: 2,
    name: 'Mohan ',
    father_name: 'Sohan',
    dob: ISODate('2025-07-23T00:00:00.000Z'),
    profile_image: 'profile_images/rpsir_1defnKF.jpg',
    marksheet_pdf: 'marksheets/t3_duniya__YZRMTN4.pdf',
    video_file: 'videos/END_VIDEO_CLIP.mp4'
  }
]
regdb>
```

Step-7: write urls.py to redirect to the page



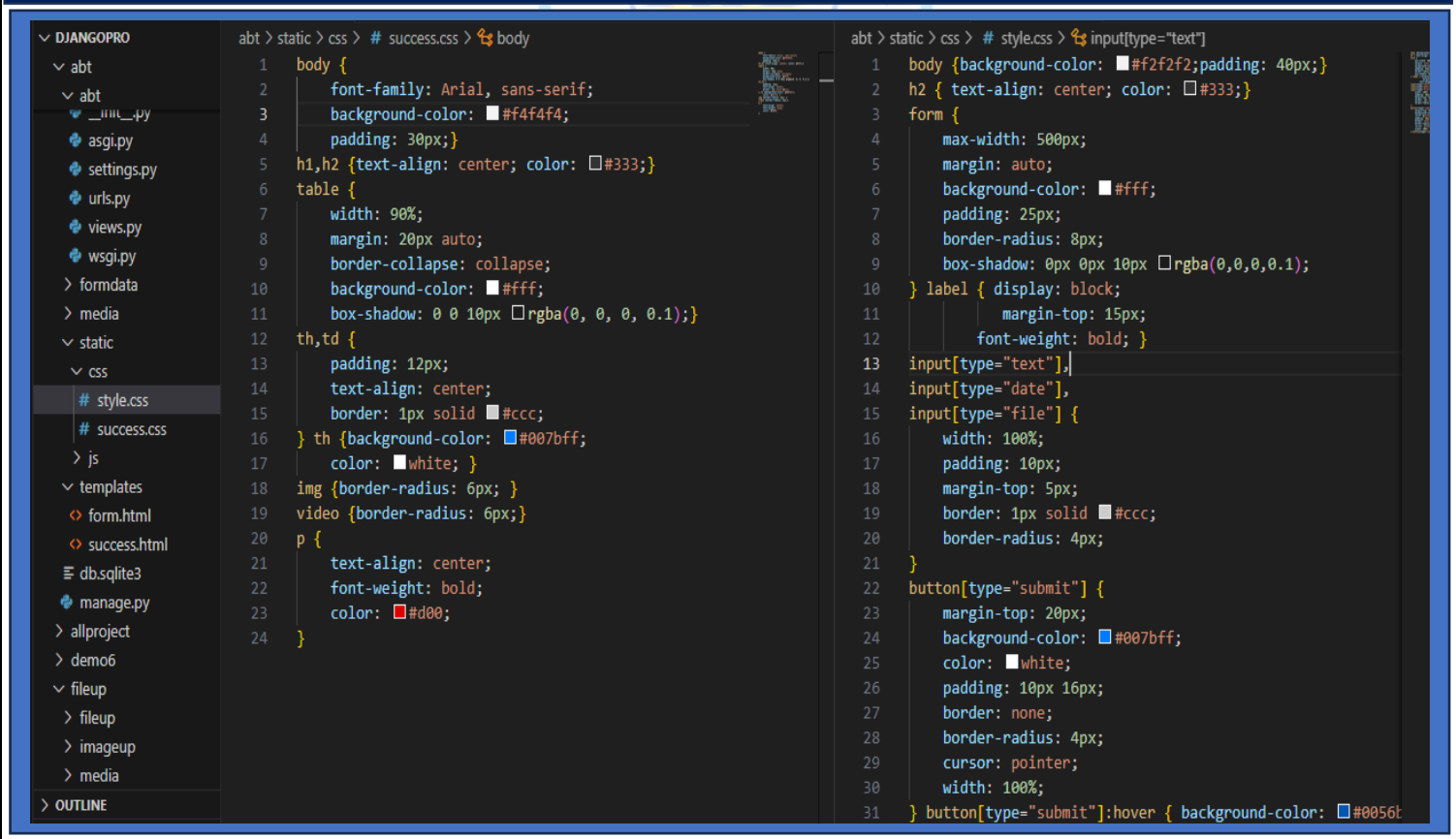


Step-8: write success.htm inside templates



```
{% load static %}
<!DOCTYPE html>
<html>
<head> <meta charset="UTF-8">
    <title>Student Data</title>
    <link rel="stylesheet" href="{% static 'css/success.css' %}">
</head>
<body> <h1>Registration Successful</h1><h2>Registered Students</h2>
    {% if students %}
        <table>
            <tr>
                <th>Name</th>
                <th>Father Name</th>
                <th>Date of Birth</th>
                <th>Profile Image</th>
                <th>Marksheet</th>
                <th>Video</th>
            </tr>
            {% for student in students %}
            <tr> <td>{{ student.name }}</td>
                <td>{{ student.father_name }}</td>
                <td>{{ student.dob }}</td>
                <td> <img src="{{ student.profile_image.url }}" width="100" height="100"></td>
                <td><a href="{{ student.marksheet_pdf.url }}" target="_blank">View PDF</a></td>
                <td> <video width="200" controls> <source src="{{ student.video_file.url }}"></video></td>
            </tr>
            {% endfor %}
        </table>
    {% else %}
        <p>No student data found.</p>
    {% endif %}
</body></html>
```

Step-9: write style.css and success.css inside static (css)



```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    padding: 30px;}
h1,h2 {text-align: center; color: #333;}
table {
    width: 90%;
    margin: 20px auto;
    border-collapse: collapse;
    background-color: #fff;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);}
th,td {
    padding: 12px;
    text-align: center;
    border: 1px solid #ccc;
} th {background-color: #007bff;
    color: white; }
img {border-radius: 6px; }
video {border-radius: 6px;}
p {
    text-align: center;
    font-weight: bold;
    color: #d00;
}
```

```
body {background-color: #f2f2f2;padding: 40px;}
h2 { text-align: center; color: #333;}
form {
    max-width: 500px;
    margin: auto;
    background-color: #fff;
    padding: 25px;
    border-radius: 8px;
    box-shadow: 0px 0px 10px rgba(0,0,0,0.1);
} label { display: block;
            margin-top: 15px;
        font-weight: bold; }
input[type="text"],
input[type="date"],
input[type="file"] {
    width: 100%;
    padding: 10px;
    margin-top: 5px;
    border: 1px solid #ccc;
    border-radius: 4px;
}
button[type="submit"] {
    margin-top: 20px;
    background-color: #007bff;
    color: white;
    padding: 10px 16px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    width: 100%;
} button[type="submit"]:hover { background-color: #0056
```

How to Send the mail in Django

Step 1: Update settings.py with Email Configuration

- You must configure your SMTP (mail server) settings. Below is an example using **Gmail SMTP**:

```
# settings.py # Use SMTP as the email backend service
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'          # Gmail's SMTP server
EMAIL_PORT = 587                       # Port for sending emails (587 for TLS)
EMAIL_USE_TLS = True                   # Use TLS (Transport Layer Security) for secure connection
EMAIL_HOST_USER = 'your_email@gmail.com'        # Your Gmail address used to send emails
EMAIL_HOST_PASSWORD = 'your_app_password'       # App password (NOT your Gmail password)
#views.py import this
```

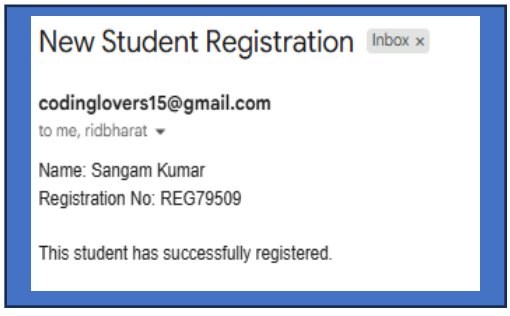
- from django.core.mail import send_mail

Views.py

```python
from django.shortcuts import render, redirect
from formdata.models import Student
import random
from django.core.mail import send_mail
def home(request):
    if request.method == 'POST':
        name_1 = request.POST.get('name')
        father_name_1 = request.POST.get('father_name')
        dob_1 = request.POST.get('dob')
        profile_image_1 = request.FILES.get('profile_image')
        marksheet_pdf_1 = request.FILES.get('marksheet_pdf')
        video_file_1 = request.FILES.get('video_file')
        student = Student.objects.create( # Save student data
            name=name_1,
            father_name=father_name_1,
            dob=dob_1,
            profile_image=profile_image_1,
            marksheet_pdf=marksheet_pdf_1,
            video_file=video_file_1
        ) # Generate random registration number
        reg_number = f"REG{random.randint(10000, 99999)}"
        # Email details
        subject = "New Student Registration"
        message = f'''Name: {name_1}\nRegistration No: {reg_number}\n\n
        This student has successfully registered.'''
        from_email = 'codinglovers15@gmail.com'
        to_email = ['rajeshpd339@gmail.com', 'ridbharat@gmail.com']
        # Send email
        send_mail(subject, message, from_email, to_email, fail_silently=False)
        return redirect('success')
    return render(request, 'form.html')
def showdata(request):
    students = Student.objects.all()
    return render(request, 'success.html', {'students': students})
```

setting.py

```python
import os
#For Static Files (CSS, JS)
STATIC_URL = '/static/'
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
#For Media Files (User uploads)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

#for email send
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'codinglovers15@gmail.com'
EMAIL_HOST_PASSWORD = 'kmsdpywmpersvlpi'  # App password with no
```



Steps to Get App Password for `codinglovers15@gmail.com`

1. **Go to** https://myaccount.google.com/
2. **Sign in** with your Gmail account: `codinglovers15@gmail.com`
3. On the left menu, click **"Security"**
4. Under **"Signing in to Google"**, make sure:
   - **2-Step Verification** is **ON**
   - If it's **OFF**, turn it on first and complete the setup
5. After 2-Step Verification is enabled, you'll see an option below it: **"App Passwords"**
   👉 Or visit directly: https://myaccount.google.com/apppasswords
6. Select:
   - **App:** `Mail`
   - **Device:** `Other` (enter something like "Django Server")
7. Click **Generate**
8. Google will show a **16-character app password** like:

```
nginx

abcd efgh ijkl mnop
```

9. Use this password (no spaces) in your Django settings:

```
python

EMAIL_HOST_PASSWORD = 'abcdefghijklmnop'
```

Sending Emails in Django

Step 1: Update settings.py with Email Configuration

You must configure your SMTP (mail server) settings. Below is an example using **Gmail SMTP**:

```
# settings.py
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'your_email@gmail.com'     # Your email address
EMAIL_HOST_PASSWORD = 'your_app_password'    # App password (NOT your Gmail password)
```

- Use environment variables or django-environ to hide passwords in production.

Step 2: Enable App Password in Gmail (if using Gmail)

1. Go to https://myaccount.google.com/
2. Enable **2-Step Verification**
3. Go to **Security → App Passwords**
4. Generate a password for "Mail" and "Other" app (use that as EMAIL_HOST_PASSWORD)

Step 3: Create a Django View to Send Email

```
# views.py
from django.core.mail import send_mail
from django.http import HttpResponse
def send_test_email(request):
    send_mail(
        subject='Hello from Django',
        message='This is a test email.',
        from_email='your_email@gmail.com',
        recipient_list=['receiver@example.com'],
        fail_silently=False,
    )
    return HttpResponse("Email has been sent!")
```

Optional: Send Email with Attachment

```
def send_email_with_attachment(request):
    email = EmailMessage(
        'Subject Here',
        'Here is the message.',
        'your_email@gmail.com',
        ['receiver@example.com']
    )
    email.attach_file('path/to/your/file.pdf')
    email.send()
    return HttpResponse("Email with attachment sent!")
```

Step 4: Add URL Pattern

```
# urls.py
from django.urls import path
from . import views
urlpatterns = [
    path('send-email/', views.send_test_email, name='send_email'),
] Visit http://127.0.0.1:8000/send-email/ to trigger the email.
```

Optional: Send HTML Email

```
from django.core.mail import EmailMessage
from django.template.loader import render_to_string
def send_html_email(request):
    subject = "Welcome to MySite"
    message = render_to_string('email_template.html', {'user': 'Sangam'})
    email = EmailMessage(subject, message, 'your_email@gmail.com', ['receiver@example.com'])
    email.content_subtype = 'html'  # Main content is text/html
    email.send()
    return HttpResponse("HTML email sent!")
```

RESEARCH INNOVATION DISCOVERY
RID
BHARAT

RESEARCH INNOVATION DISCOVERY
RID
BHARAT

RESEARCH INNOVATION DISCOVERY
RID
BHARAT