**‘Fresh install of django in virtual env**

Mkdir djangoProject

Cd djangoProject

Pip install virtual env

Virtualenv env

env\Scripts\activate

Pip install django

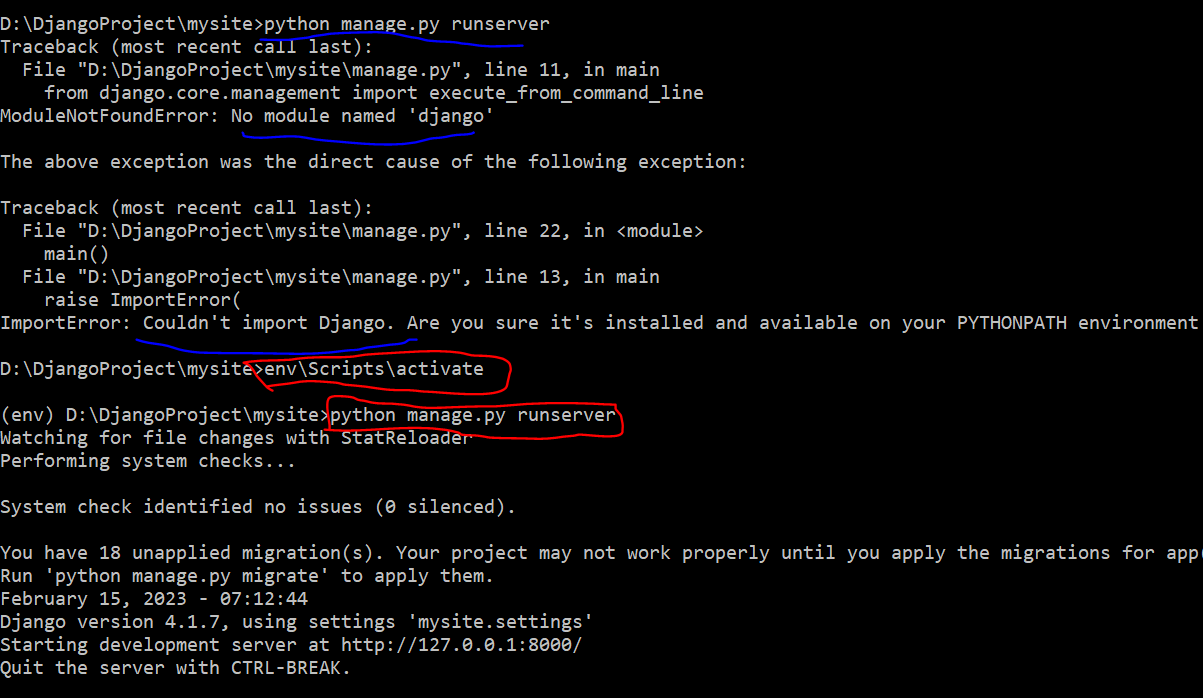
Django-admin

Python mange.py runserver

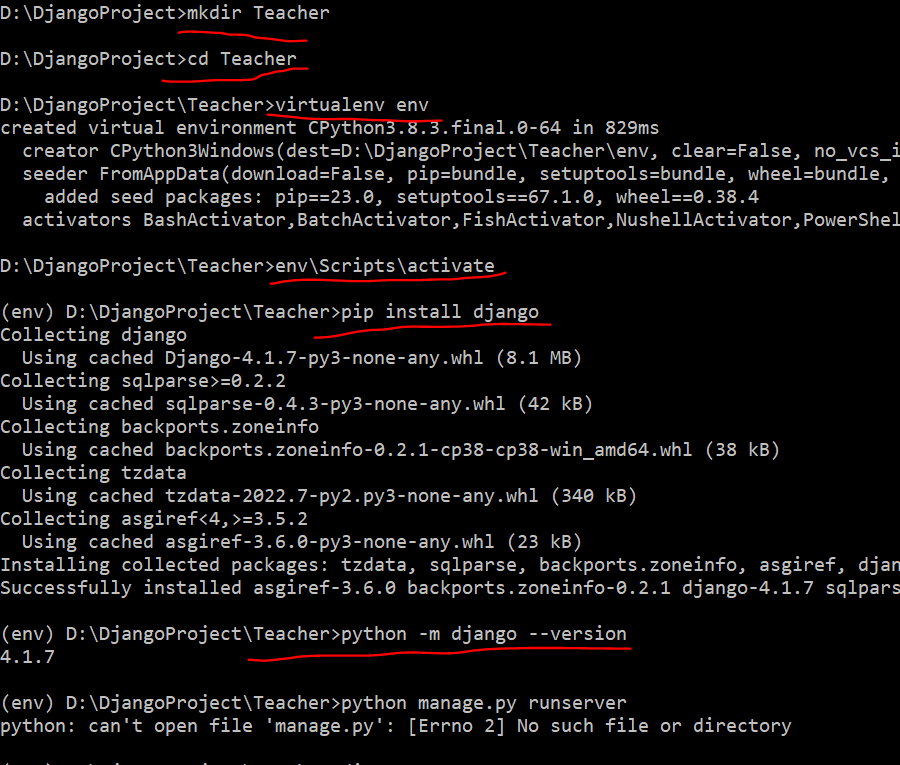
Python –m django –version

Derop all table >python manage.py migrate management\_app zero

**First activate the virtual env then run mange.py**



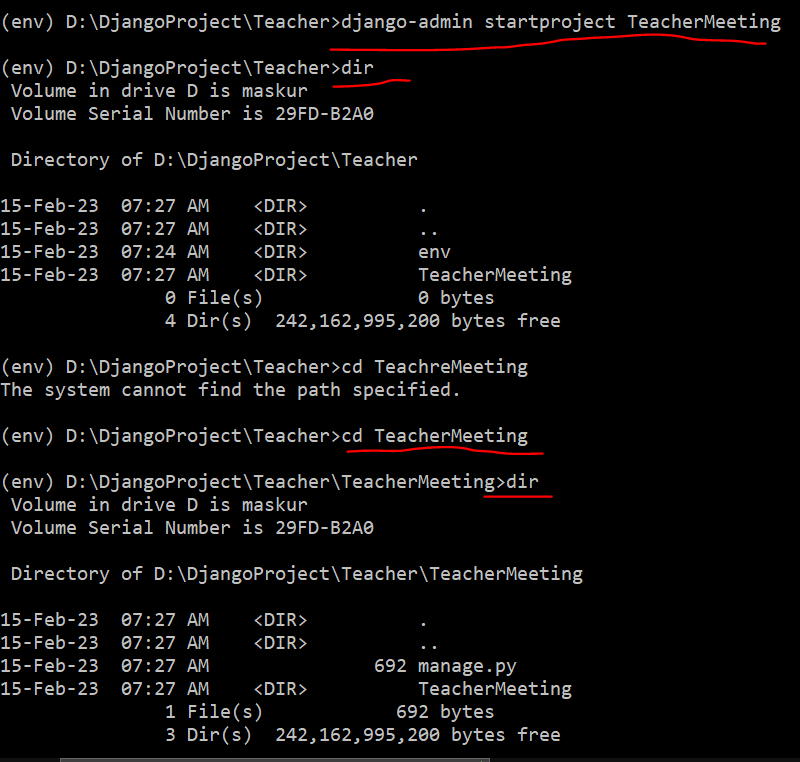
Complete guideline to install django

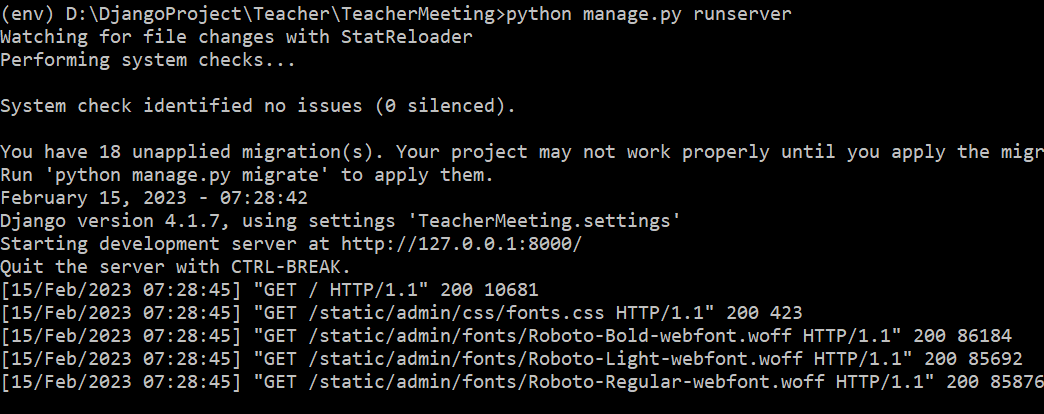




If the the app level template doesn’t find then check the project settings add the app name in installed apps

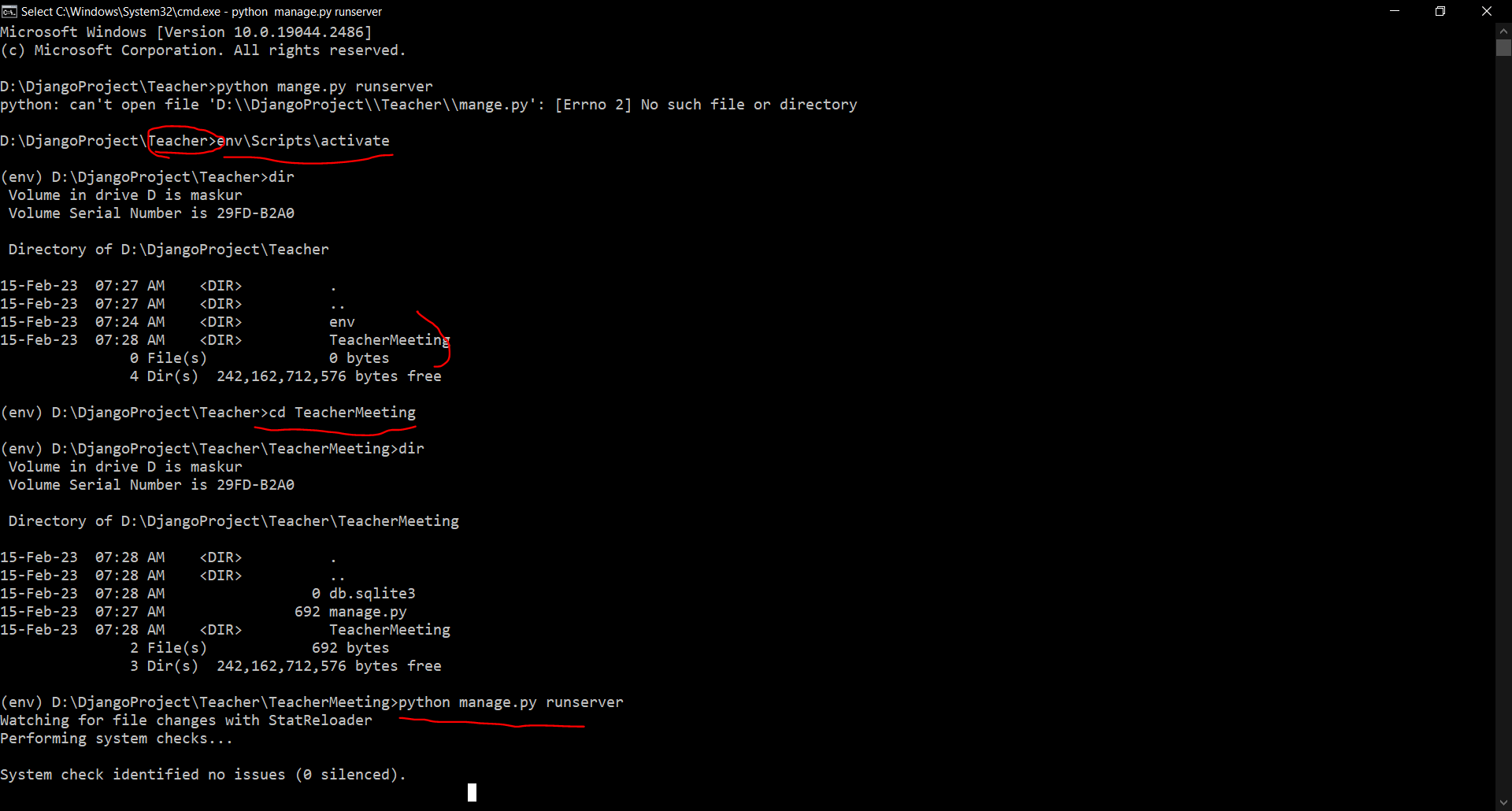
Project level template add -> go to settings then add os.path





Now you can go to browser : 127.0.0.1:8000

After closing this terminal and again to start server go to the Teacher folder and activate the env by (env\Scripts\activate)



Base.html

<body>

{% block content %}

{% endblock content %}

</body>

Home.html

{% extends "base.html" %}

{% block content %}

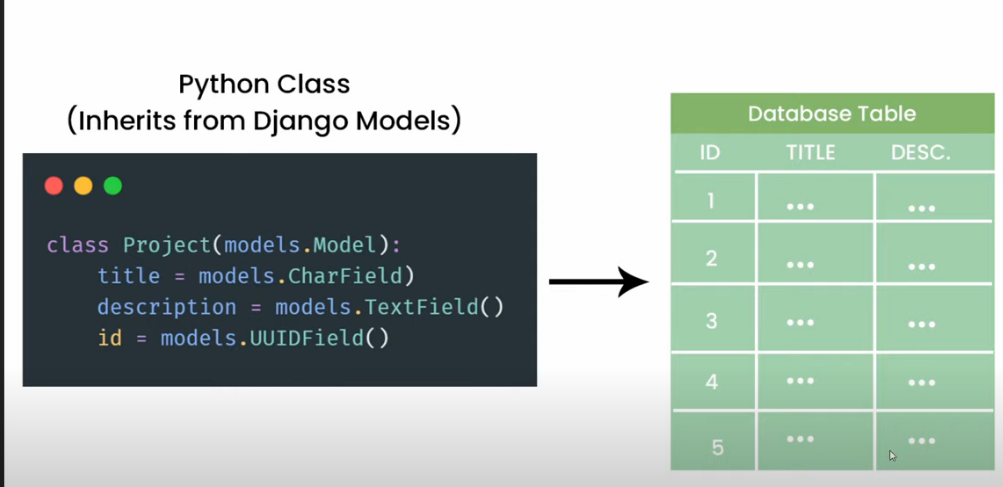
<h1>Welcome</h1>

<p>This is the home page</p>

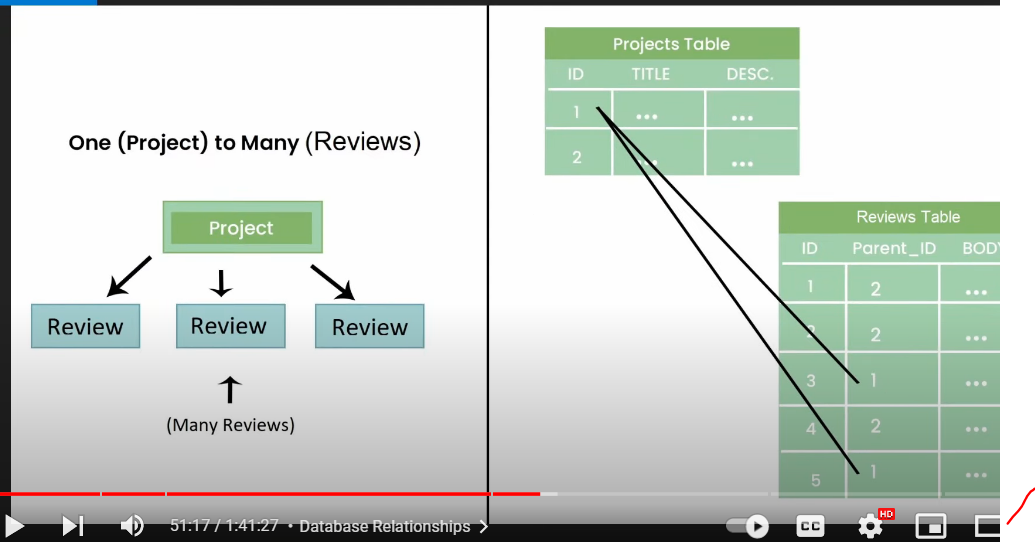
{% endblock content %}

Python manage.py migrate

Python manage.py createsuperuser



Python manage.py makemigrations



Linking url path in html page :

Two ways by url name or direct path

            <td> <a href="/project/{{project.id }}"> {{project.title }} </a></td>

Github:first time

git init

git add -A

git commit -m "Initial commit"

git branch -M main #this wil change the baranch master to main

git remote add origin https://github.com/your-username/your-repository.git

git push -u origin main # it will track the main branch for next push

…………………….

Git status

Git pull origin main

Git add –a # it will add all file for commit

Git commit # this will commit all the file in local

Then there will be open a msg prompet in cmd to enter msg press (i) to close the editor press ctrl+c then type :wq

* Then to push the local commit to remote server send .
* Git push origin main
* To add remote server use
* Git add remote “LINK of the project”

git checkout -b main # this will create a branch main locally and remotely

|  |
| --- |
| git remote -v #check the remote repo  git remote add origin [git@github.com-rahul-office:rahul-office](mailto:git@github.com-rahul-office:rahul-office) # adding the remote repo where to push  git config user.name “maskur12”  git config user.email “mail@gmailc”  make sure from where the push is happening |

Multiple github account in local

|  |
| --- |
| Here's the full process to set up SSH keys for both your personal and work GitHub accounts on Windows:  1. \*\*Check Existing SSH Keys (Optional):\*\*  - Navigate to your SSH directory (`C:\Users\YourUsername\.ssh`) in File Explorer.  - Check if you already have SSH key files (`id\_rsa` and `id\_rsa.pub`). If you do, you can skip generating new keys for your personal account. If not, proceed to the next step.  2. \*\*Generate SSH Keys for Personal Account (If Needed):\*\*  - Open Git Bash. If you don't have Git Bash installed, you can download and install it from the Git website.  - Run the following command to generate SSH keys:  ```  ssh-keygen -t rsa -b 4096 -C "your.email@example.com"  ```  Replace `"your.email@example.com"` with the email associated with your personal GitHub account.  - Follow the prompts to generate the SSH key pair. You can choose the default location (`C:\Users\YourUsername\.ssh`) and optionally set a passphrase for added security.  3. \*\*Add SSH Key to GitHub (Personal Account):\*\*  - Copy the contents of your public key file (`id\_rsa.pub`). You can do this by opening the file in a text editor like Notepad and copying the contents.  - Log in to your personal GitHub account in a web browser.  - Go to "Settings" > "SSH and GPG keys" > "New SSH key."  - Paste your public key into the "Key" field and give it a descriptive title.  - Click "Add SSH key" to save.  4. \*\*Generate SSH Keys for Work Account:\*\*  - Follow the same process as step 2 to generate SSH keys, but use your work email for the `-C` option:  ```  ssh-keygen -t rsa -b 4096 -C "work.email@example.com"  ```  Replace `"work.email@example.com"` with the email associated with your work GitHub account.    5. \*\*Add SSH Key to GitHub (Work Account):\*\*  - Follow the same process as step 3 to add your work SSH key to your work GitHub account.  6. \*\*Configure SSH Config File (Optional):\*\*  - If you want to manage multiple SSH keys and GitHub accounts more easily, you can create or modify the SSH config file.  - Open Git Bash and navigate to your SSH directory (`cd ~/.ssh`).  - Edit the SSH config file using a text editor like Nano:  ```  nano config  ```  - Add the following configuration:  ```  # Personal GitHub account  Host github.com  HostName github.com  User git  IdentityFile ~/.ssh/id\_rsa    # Work GitHub account  Host github-work  HostName github.com  User git  IdentityFile ~/.ssh/id\_rsa\_work  ```  Replace `id\_rsa\_work` with the name of your work SSH key file.  - Save the file and exit the text editor.  7. \*\*Update Git Configurations: \*\*  - Open Git Bash and run the following commands to configure Git with your GitHub usernames:  ```  ```  Replace `"Your Personal GitHub Username"` and `"your.personal.email@example.com"` with your personal GitHub username and email.  - You may need to configure Git with your work GitHub username as well if you haven't already:  ```  git config --global user.name "Your Work GitHub Username"  git config --global user.email "your.work.email@example.com"  ```  8. \*\*Clone and Use Repositories:\*\*  - Now, you can clone repositories from both your personal and work GitHub accounts using SSH URLs.  - When you interact with repositories, Git will use the appropriate SSH key based on the host specified in the repository URL.  By following these steps, you'll have SSH keys set up for both your personal and work GitHub accounts on Windows, allowing you to securely authenticate with remote repositories using SSH. |
| git clone [git@github-work:M-SABIL/test.git](mailto:git@github-work:M-SABIL/test.git) # this is the github-work host set in the config file use this which is in .ssh  git clone [git@github-work:a1qa-education/M.SABIL.git](mailto:git@github-work:a1qa-education/M.SABIL.git)  <https://github.com/a1qa-education/java_unit_selenium.git>  git clone [git@github-work:a1qa-education/java\_unit\_selenium.git](mailto:git@github-work:a1qa-education/java_unit_selenium.git%20) |

|  |
| --- |
| Configuration for the current repository  Navigate to the repository  git config user.name  git config user.email |
| Configure the user for github  git config --global user.name "Your Personal GitHub Username"  git config --global user.email [your.personal.email@example.com](mailto:your.personal.email@example.com)  This account will be use when any commit to the remote reopo |
| Override Local Configurations (Optional):  If you're working on a repository that should use a different identity than your global default (for example, a work repository), you can override the global configurations with local configurations specific to that repository. Navigate to the repository directory and run the following commands with your work account credentials:  git config user.name "Your Work GitHub Username"  git config user.email "your.work.email@example.com" |

|  |
| --- |
| When you use `git clone` to clone a repository, the default remote name for the repository is typically set to "origin", and the URL for the remote repository is set to the repository's GitHub URL.  If you're working with multiple GitHub accounts and you want to ensure that the cloned repository uses the correct SSH key for authentication, you can configure the SSH URL for the remote repository accordingly. This is especially important if you have different SSH keys associated with different GitHub accounts.  Here's how you can specify the SSH URL with the correct host alias for the remote repository when cloning:  ```bash  git clone git@github.com:username/repo.git  ```  Replace "username" with your GitHub username and "repo" with the name of the repository you want to clone.  If you have multiple SSH keys set up and you want to use a specific SSH key for authentication with a particular GitHub account, you can configure SSH aliases in your SSH config file (`~/.ssh/config`) to specify which SSH key to use for different hosts. For example:  ```bash  Host github.com  HostName github.com  User git  IdentityFile ~/.ssh/id\_rsa\_personal  Host github-work  HostName github.com  User git  IdentityFile ~/.ssh/id\_rsa\_work  ```  In this example, "github.com" is the alias for your personal GitHub account, and "github-work" is the alias for your work GitHub account. Each alias specifies a different SSH key (`id\_rsa\_personal` for personal account and `id\_rsa\_work` for work account).  Then, when cloning a repository, you can use the appropriate host alias:  ```bash  git clone git@github-work:username/repo.git  ```  This ensures that Git uses the correct SSH key associated with the specified host alias for authentication when interacting with the remote repository. |

Add static file :

To do this create static folder in the project

Then add this folder in the setting.py

STATICFILES\_DIRS = [

   BASE\_DIR / 'static'

   #os.path.join(BASE\_DIR,'static')

]

To set the upload image location use media root in setting:

MEDIA\_ROOT = BASE\_DIR / 'static/images'

Use the media url for user upload which define the directory wehere the user upload content will be saved :

MEDIA\_URL = '/images/'

Configure the urls in main project urls.py

urlpatterns+=static(settings.MEDIA\_URL,document\_root = settings.MEDIA\_ROOT)

use a featured image field in database models

    featured\_image = models.ImageField(null=True,blank=True)

we can access the image in html page directly :

<img style="max-width: 200px;" src="{{book.featured\_image.url}}" alt="">

Or specifiy it in the function in models

define the function to return the file path of image to show in the html page

   @property

    def imageURL(self):

        try:

            img= self.featured\_image.url

        except:

            img=''

        return img

in html to show the form we can use the shortcut :

{{form.as\_p} or we can use custom way .

Use enctype=to submit pdf,image file

<form action="" method="POST" enctype="multipart/form-data">

    {% csrf\_token %}

     {% for field in form  %}

        <div>

            {{field.label}}

            {{field}}

        </div>

      {% endfor %}

    <input type="submit" value="Create">

</form>

And we need to change the views.py where we add request.files

       form = ProjectForm(request.POST,request.FILES)

production :

set debug =False

add the static root = base\_dir / ‘staticfiles’

python manage.py collectstatic

confifure the urls

urlpatterns+=static(settings.STATIC\_URL,document\_root = settings.STATIC\_ROOT)

whitenoise install and add to middleware

------Migrating a single app:

Python manage.py makemigrations app\_name;

Xxx\_migrations file will generate

Python manage.py sqlmigrate app\_name Xxx\_file

Python manage.py migrate

Django restful Api

Mkdir restCrud

Cd restCrud

Virtualenv env

Env\scripts\activate

Pip install django-admin

Django-admin startproject restfulApiCrud

Cd restfulApiCrud

Code .

Python manage.p startapp employeeApi

Pip install djangorestframework

Set the installed app: rest\_framework

For postgres

Pip install psycopg2

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'restfulapidb' ,

        'USER': 'postgres',

        'PASSWORD': 'postgres',

        'HOST' :'localhost'

    }

}

Changing superuser password:

Python manage.py changepassword user\_name

Broken pipe error

Use a try catch block to rescue any exception

Deploy app to heroku

>pip install psycopg2 gunicorn django-heroku whitenoise

Collecting psycopg2

Procfile

runtime

pip freeze > requirements.txt

Deploy on Render

Add the staticurlpatterns so that in server gunicorn can detect

from django.contrib.staticfiles.urls import staticfiles\_urlpatterns

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('todoapp.urls')),

]

urlpatterns+=staticfiles\_urlpatterns()

To add django database url : pip install dj-database-url

Pip install gunicorn

pip install psycopg2-binary

Create a postgres database in render

Copy external database link

Go to settings and set this

DATABASES ={

    "default":dj\_database\_url.parse(os.environ.get("DATABASE\_URL"))

}

Got to project cmd and set the dabatase\_url

set DATABASE\_URL=postgres://tickler\_user:l4FO2axIW2qdZPQ8clUhvXsvbilrJ6cr@dpg-ckglbskldqrs73dkj8v0-a.oregon-postgres.render.com/tickler

echo %DATABASE\_URL%

python manage.py migrate

ALLOWED\_HOSTS = ['\*']

Type this to add all the apckeges nedded to

pip freeze > requirements.txt

then changes the requirenst.txt manually set crpytographay==2.8 if there there is a issue in the build process when deploy on render .

upload the project on github

init git inside the project and then upload

after uploading go to render website

**create webservices and select github then select the project**

**give the name of the webservice**

**in gunicorn app write gunicorn projectName.wsgi:application**

**go to advanced environment variable then add**

**PYTHON\_VERSION =3.8.3**

**DATABASE\_URL = PASTe THE INTERNAL DB URL FROM RENDER**

**THEN ITS WORK FINE**

**Static file handling**

**1.pip install whitenoise**

**2.** STATIC\_ROOT = BASE\_DIR / "staticfiles"

**Put this at the bottom of the settings file**

**3.**     "whitenoise.middleware.WhiteNoiseMiddleware",

**Put this whitenoise middleware just below the security middleware**

**4. python manage.py collectstatic**

**And also you must flow the django static file directory format in the app level todoap/statcic/tdoapp/css/style.css like this .**

**And use** <script src="{% static 'todoapp/js/script.js' %}"></script>

**Not the direct url path of the css .**

**Exception**

OperationalError at /register/

could not translate host name "dpg-ckglbskldqrs73dkj8v0-a" to address: Name or service not known

this is the issue of the database check the database url is set properly both external and internal in render

set DATABASE\_URL = “THE URL”

RestaApi

Pip install django

Pip install djangorestframework

Add the rest\_framework on installed app

Create serializer.py in the app

Djanog forms & model

|  |
| --- |
| Yes, exactly! Here's a breakdown of how these components work:  ### `Meta` Class:  - \*\*Purpose\*\*: Defines which model the form is based on and specifies the fields from the model that should be rendered in the form.  - \*\*`model` attribute\*\*: Links the form to a specific model (in this case, `CustomUser`).  - \*\*`fields` attribute\*\*: Specifies which fields from the model should be included in the form. For example, `('username', 'email')` limits the form to rendering only those two fields from the model, even though the model may have more fields.  ### Form's `\_\_init\_\_` Method:  - \*\*Purpose\*\*: Customizes the form's appearance or behavior. You can use it to define attributes for the form fields (such as adding `placeholder` values or custom CSS classes).  - \*\*Example\*\*: In your case, `self.fields['email'].widget = forms.EmailInput(attrs={'placeholder': 'Email'})` adds a placeholder for the email input field.  ### `clean` Method:  - \*\*Purpose\*\*: Used for validating form data. It processes the input data, allowing you to define custom validation logic.  - \*\*Example\*\*: The method `clean` reads the data entered in `password1` and `password2` and checks if they match. If they don’t, it adds an error to the form using `self.add\_error('password2', "Passwords do not match")`.  ### How it Works Together:  1. \*\*`Meta`\*\*: Defines which fields from the model should appear in the form.  2. \*\*`\_\_init\_\_`\*\*: Applies custom attributes (like placeholders) to the form fields.  3. \*\*`clean`\*\*: Validates the input data (e.g., checking if passwords match).  So the form is responsible for rendering the fields, adding custom behavior or design, and handling data validation, while the `Meta` class controls which model fields should be rendered. |