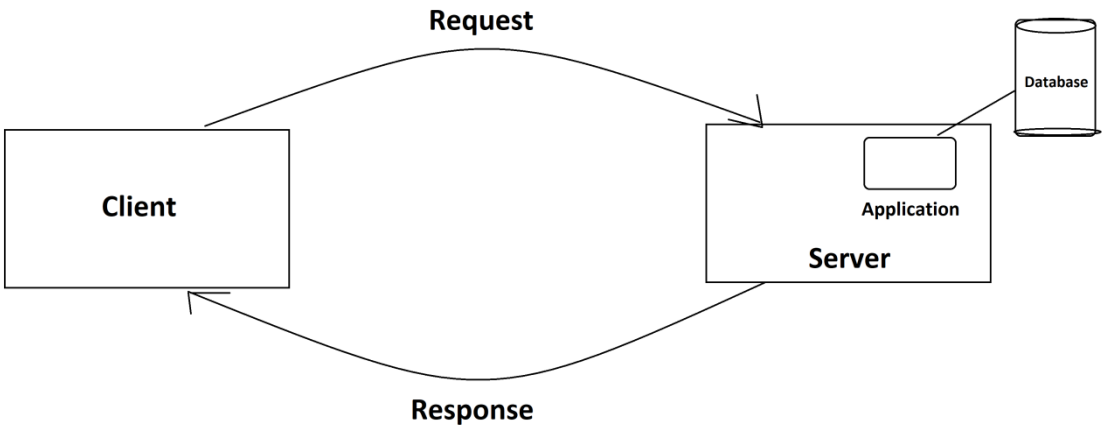
PYTHON WEB DEVELOPMENT USING DJANGO

INTRODUCTION

Introduction to Web Applications (Client – Server Environment)

Any application (Software) which can be accessed through web browsers (Chrome, Safari, Firefox, Opera, Edge etc.,) is said to be a web application. In this case Application will be stored at only one place i.e., in Server. All the client systems can access this application through web browsers. So, Client systems need to have only a web browser and internet connection (In case the server exists in remote place).



Client – Server Environment

As shown in above figure, all the requests from clients will reach the server. The Web Application which is available in the server will receives the requests, processes it and prepares the response and sent back to the corresponding client.

We can make any system as server by installing any available web server software like apache, tomcat, WebLogic etc.,

Web Designing vs Web Development:

A web designer will make our webpages look good. They mainly focus on the look and feel of the websites. For this they may use software like Photoshop to customize the website's visual elements. They also use code like HTML and CSS (Cascading Style Sheets) to create their designs. Web designers are specialized as User Experience (UX) designer (to keep visitors hooked with our website), User Interface (UI) designer (To improve the user interactivity), Visual designer (combination of both UX & UI).

Web developer will build the web designer's concept. Web developers are specialized as Back-end developer (responsible for developing core structure using Java, Python, PHP, SQL etc., that what users can't see), Front-end developer (responsible for developing webpages using HTML, CSS, JavaScript that what users can see), Full-stack developer (Both Front and Back End).

What is Django?

Django is a high-level python web framework that helps us in building and maintaining web applications. It helps us in developing applications very fast which in turns helpful in improving the productivity.

Django supports

- Object-Relational Mapping – ORM (provides a bridge between the data model and the database engine)
- Multilingual support (develop applications with various languages support), Framework support (Ajax etc.,)
- Administration GUI (ready to use user interface for administrative activities)
- Development Framework (supports end-to-end application development with an light weight web server)

SOFTWARE INSTALLATIONS

Software Installations

Python

Download and install latest version of python from https://www.python.org website.

To check version python version command is : python --version

PIP

Once you install latest version of python (after version 3.x), pip will automatically be installed. To install pip manually visit https://pip.pypa.io/en/stable/installing/ and follow the steps.

Django

To Install Django: pip install django

To Check Django Version: python –m django --version

Install Atom IDE

To Install it visit https://atom.io/ and download the IDE.

After installed Atom IDE do the following settings.

1) Goto File ->Settings (MAC O/S : preferences)

   click on install -> search “autocomplete-python” and install it. Search for “atom-django” and install it.

   Click on editor -> update the tab length to any highest number. Let us say 5.

Install MySql Software

Search in google as “mysql server download” or visit https://dev.mysql.com/downloads/installer/ and install mysql server from it.

If mysql-workbench not came with above download search in google and install it also

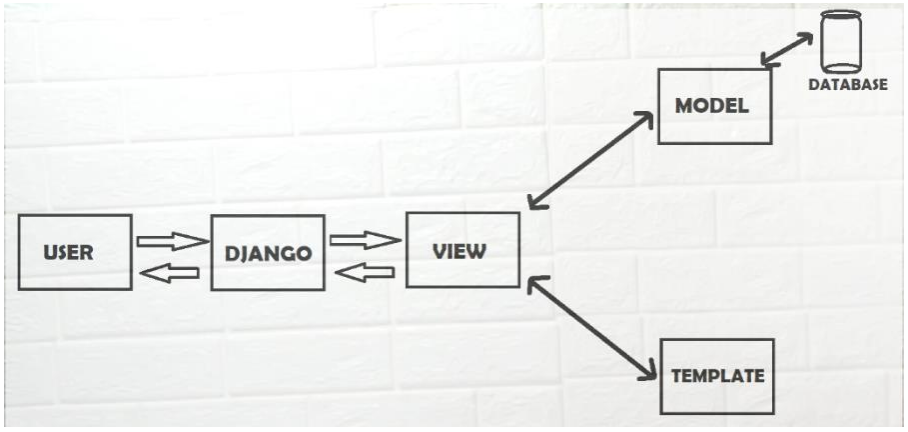
Install mysql client for our django applications :

Download mysqlclient from the below link

https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient

MVT (MODEL – VIEW – TEMPLATE) ARCHITECTURE

What is MVT (Model-View-Template)?



Django MVT (Model View Template) is a software design pattern used by Django framework in the process of developing web applications. It is collection of three important components Model, View and Template.

Model:

It helps to handle the database. We generally create a model for every table in the database. Whenever we want to access a table we will connect to the corresponding model which in turn prepares the queries and access the database and perform the requested action.

Template:

It is a presentation layer which handles user interface part. Generally, we create multiple templates where these are used as responses to the user actions.

View:

It is used to execute the business logic. It interacts with both model and template. We develop multiple views where each view is responsible for a specific action. While performing the action it will connect to model for database accessing related operations and after performing database related actions it will show a template as response to the client.

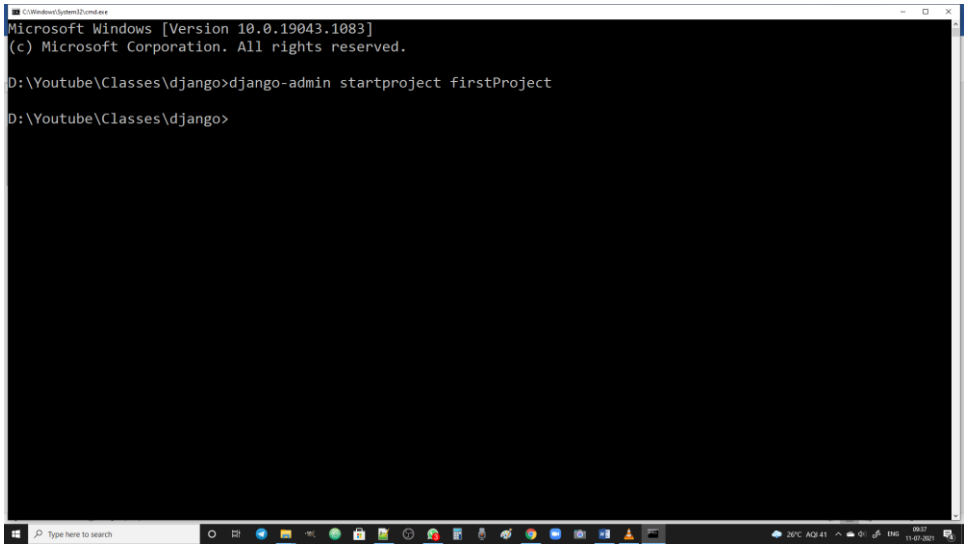
CREATING OUR FIRST DJANGO PROJECT

Django allows us to add multiple applications under same project. With the help of this feature we can easily use one application's modules in other applications.

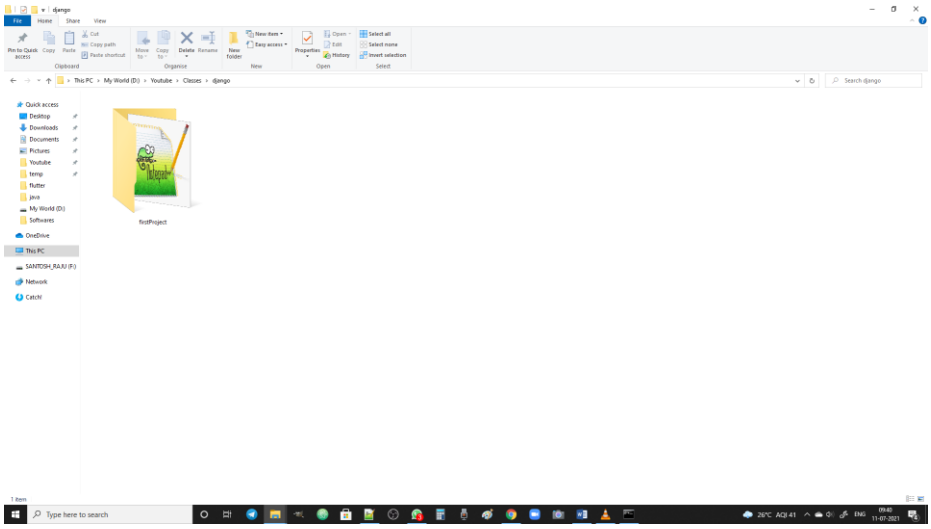
Creating First Project:

To create our first project, we need to run the below command

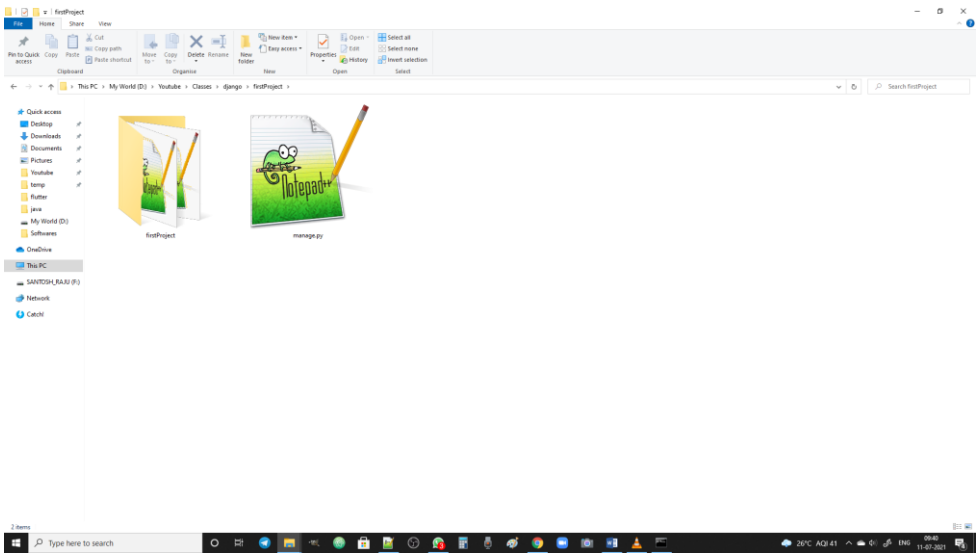
django-admin startproject projectname



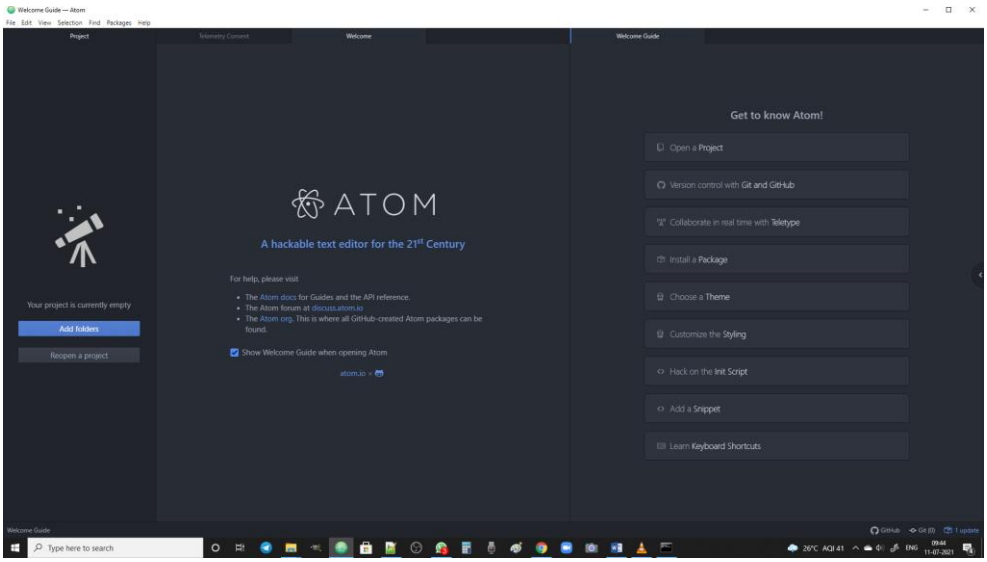
After running this command, it will creates a project folder with required files automatically.



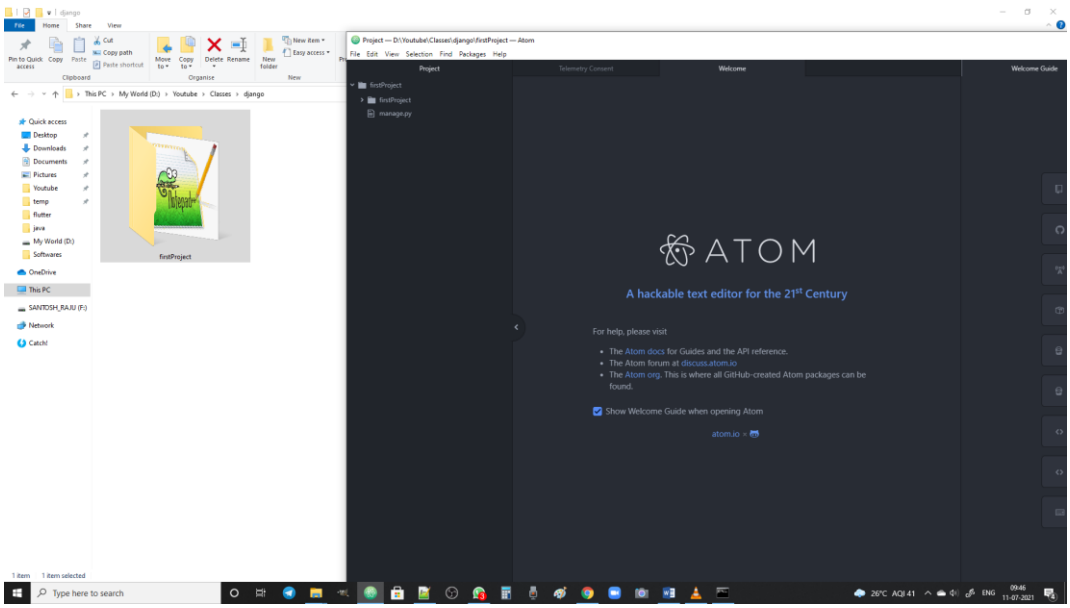
Inside this project folder, it creates a folder and a file(manage.py) as shown below.



Now let us open our IDE and load the recently created project into our IDE.

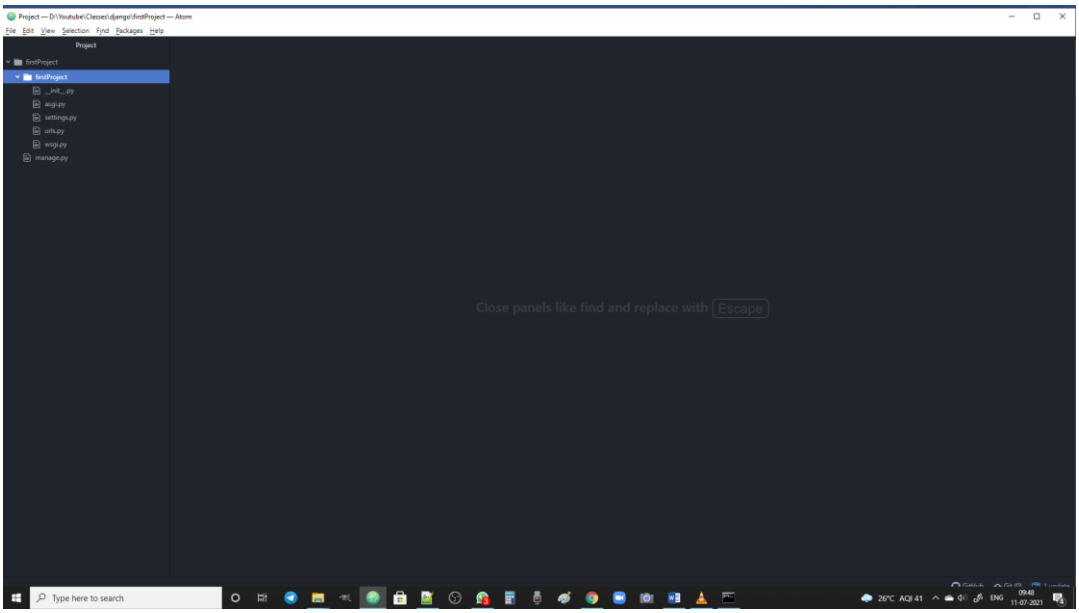


To load the project into atom, we can simply drag and drop the project folder into our ATOM.



Our project is successfully loaded into our ATOM.

Now let us observe the folder structure of our project



manage.py : We take the help of manage.py while we run our project every time

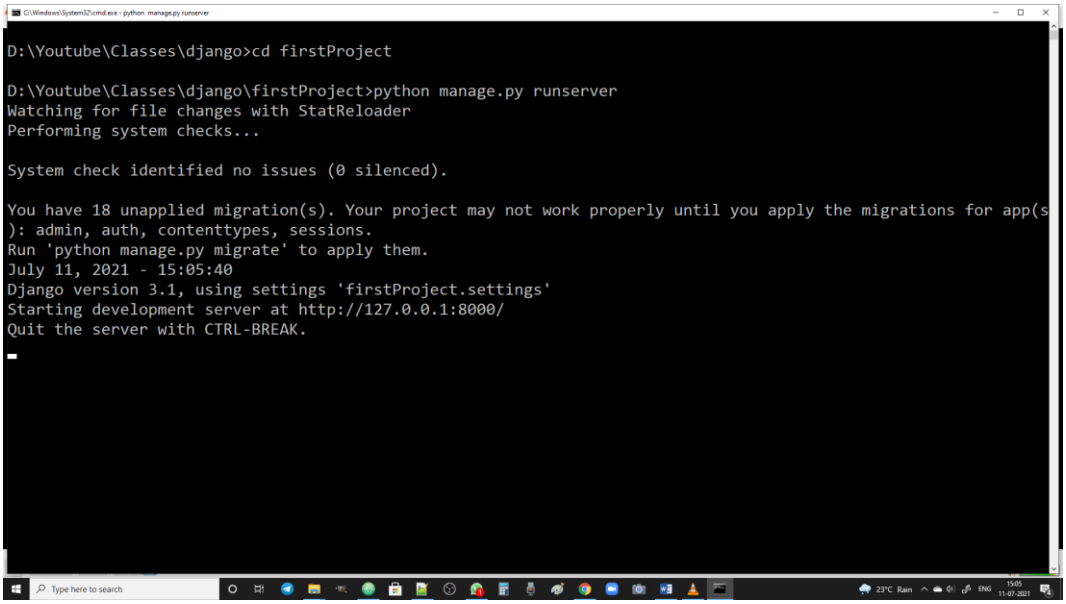
init.py : Indicates this is the python based project

settings.py : complete project settings can be configured here

urls:py : views and url mappings can be performed in this file.

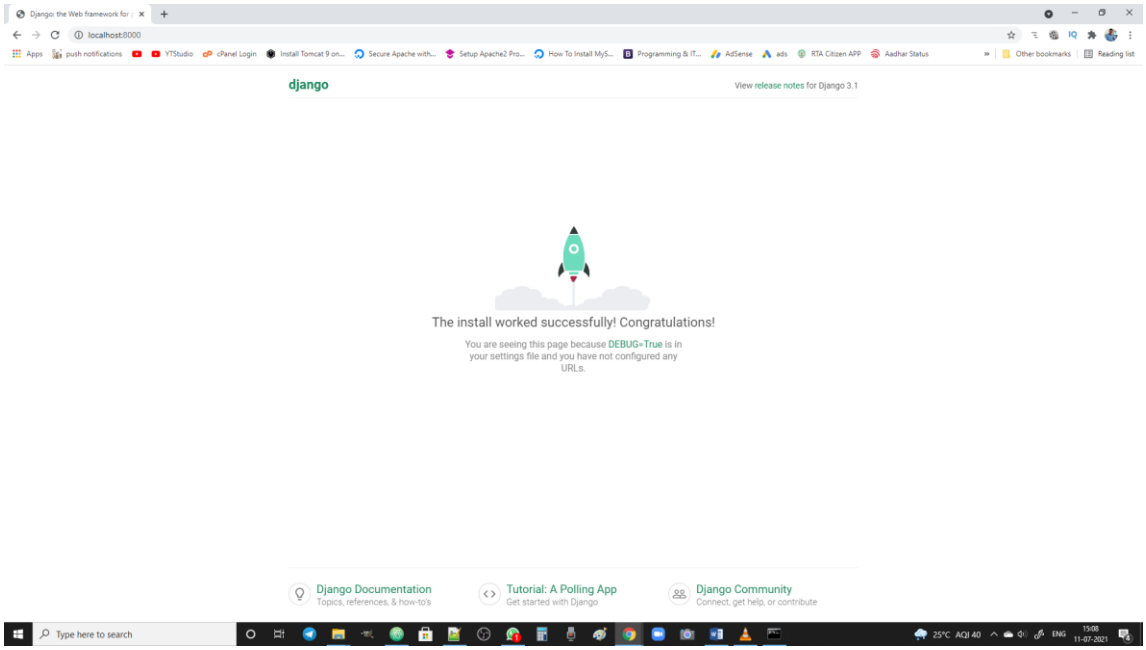
To Run our project we need to execute the below command

python manage.py runserver



Now we can run our project in web browser by using the url

http://127.0.0.1:8000 or http://localhost:8000



It is default output page provided by django. whenever we add applications then we may see our configured screens as output.

We can stop our server by using ctrl + c

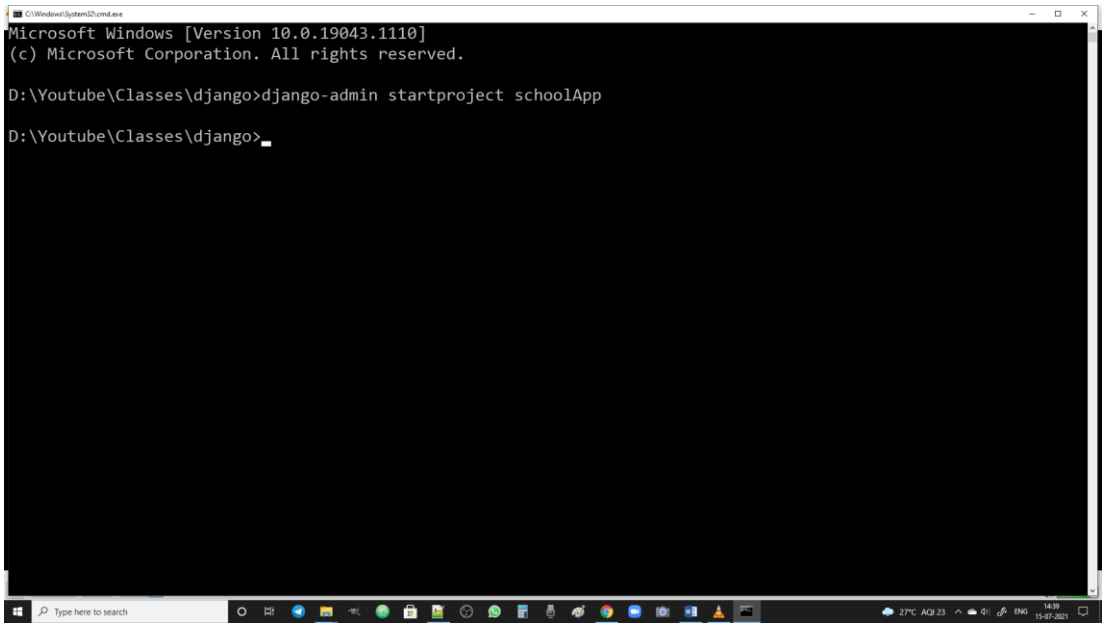
CREATING APPLICATIONS & VIEWS IN OUR DJANGO PROJECT

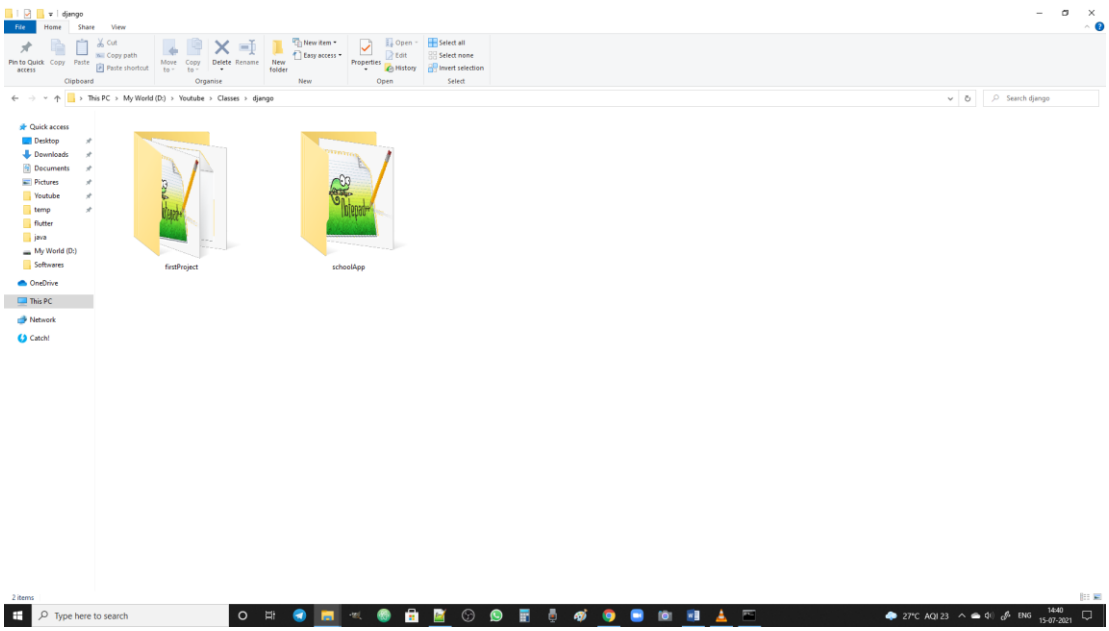
Now Let's learn how to add applications to our django project.

Steps to add application to the existing django project:

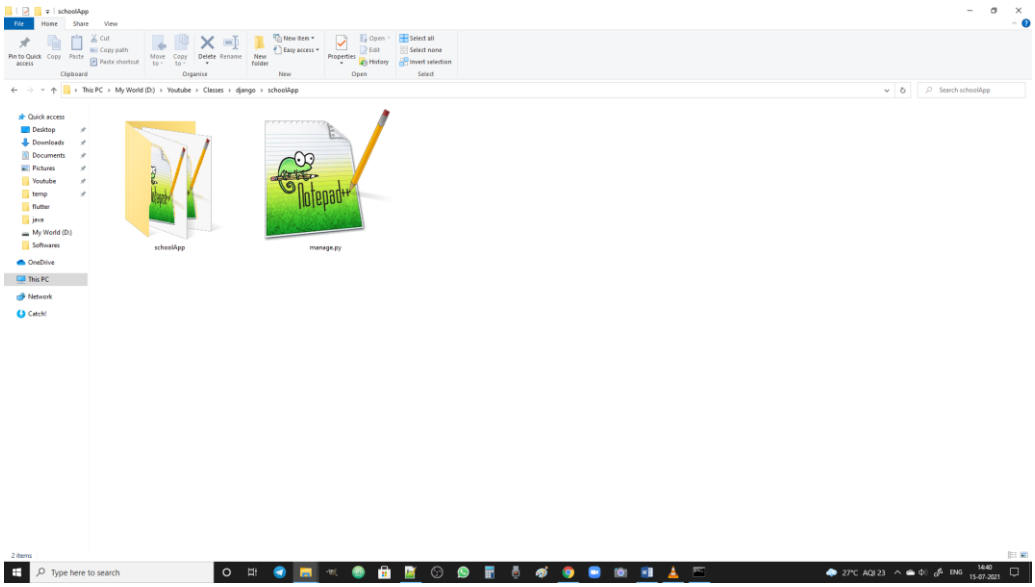
1) Create an application
2) Add this application's entry into settins.py
3) create views under the project
4) link views with urls in urls.py file
5) Run the project

First let's create a fresh project called schoolApp





Now schoolApp project is created and its folder is as shown below



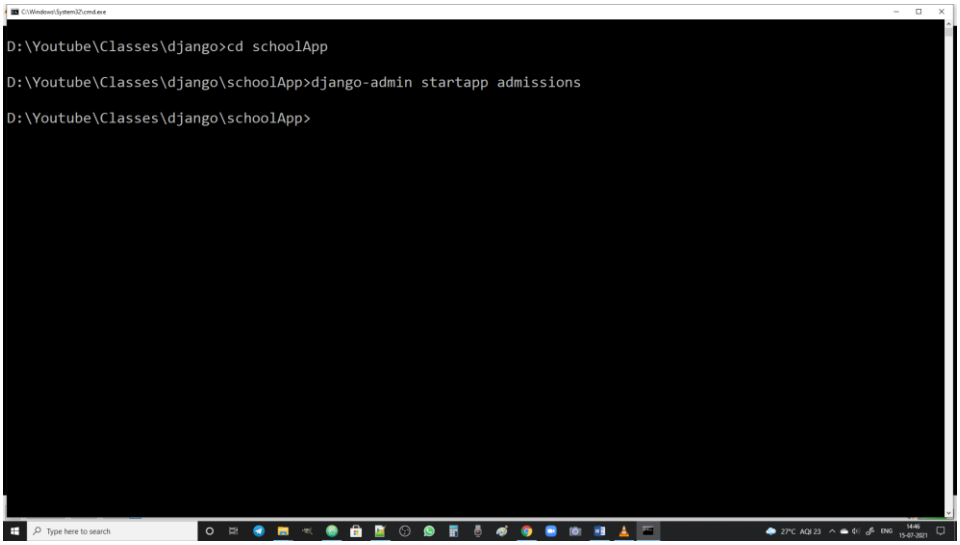
Let's load this project into atom IDE

Now let us create 'admissions' application into our schoolApp project

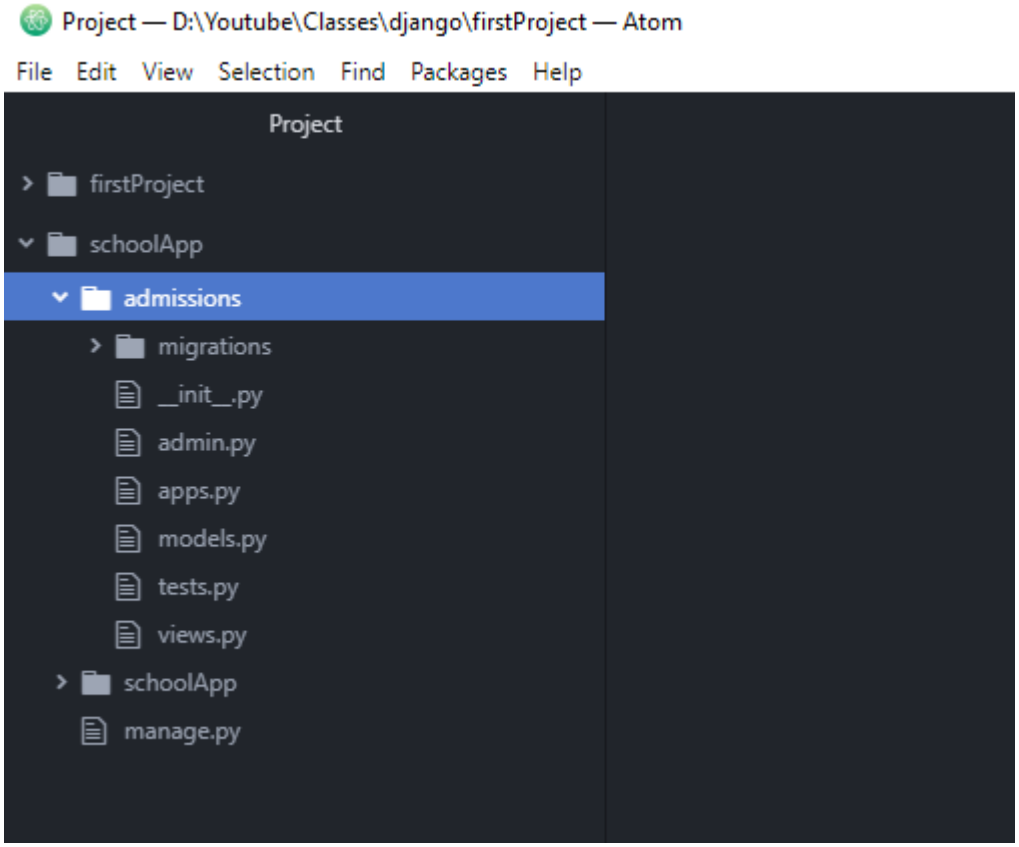
Step-1: Creating an Application

We can create an application by using startapp command as follows

django-admin startapp admissions



Now observe the admissions application's inner folder structure that is shown below



In this structure we create views in views.py file.

views – urls mappings can be done in urls.py file.

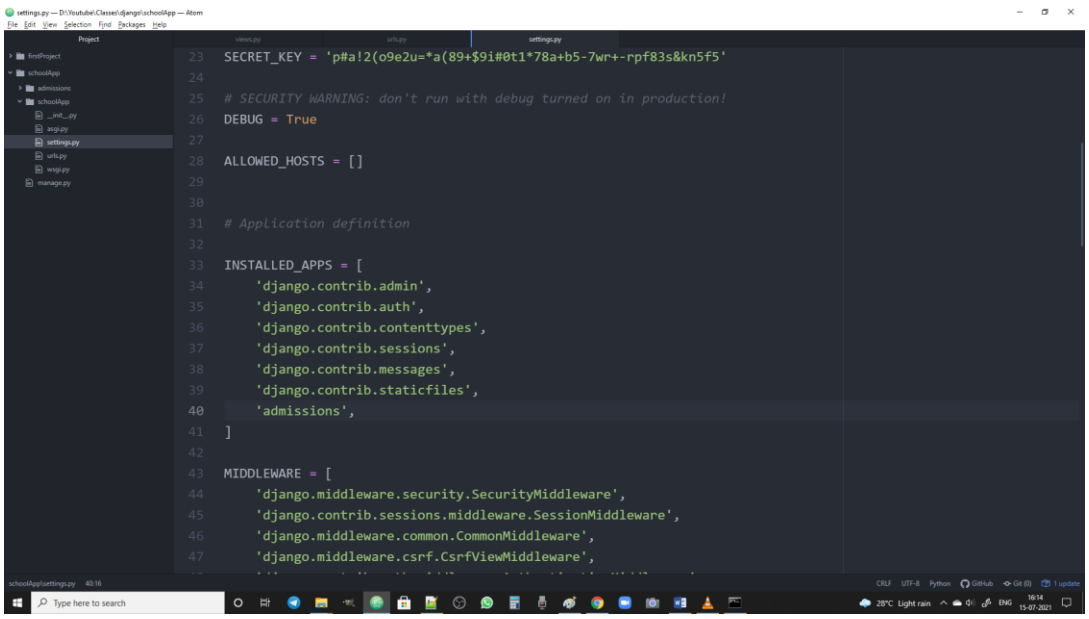
admin.py is related with administration related activities which will be discussed in further topics.

Step-2: Add this application's entry into settins.py

After creating the application we need to add it in settings.py so that it is available to the entire project so that one application's elements can be accessed by other applications.

To add this we need to open settings.py file

and in this file, under INSTALLED_APPS we need to add our application as follows.



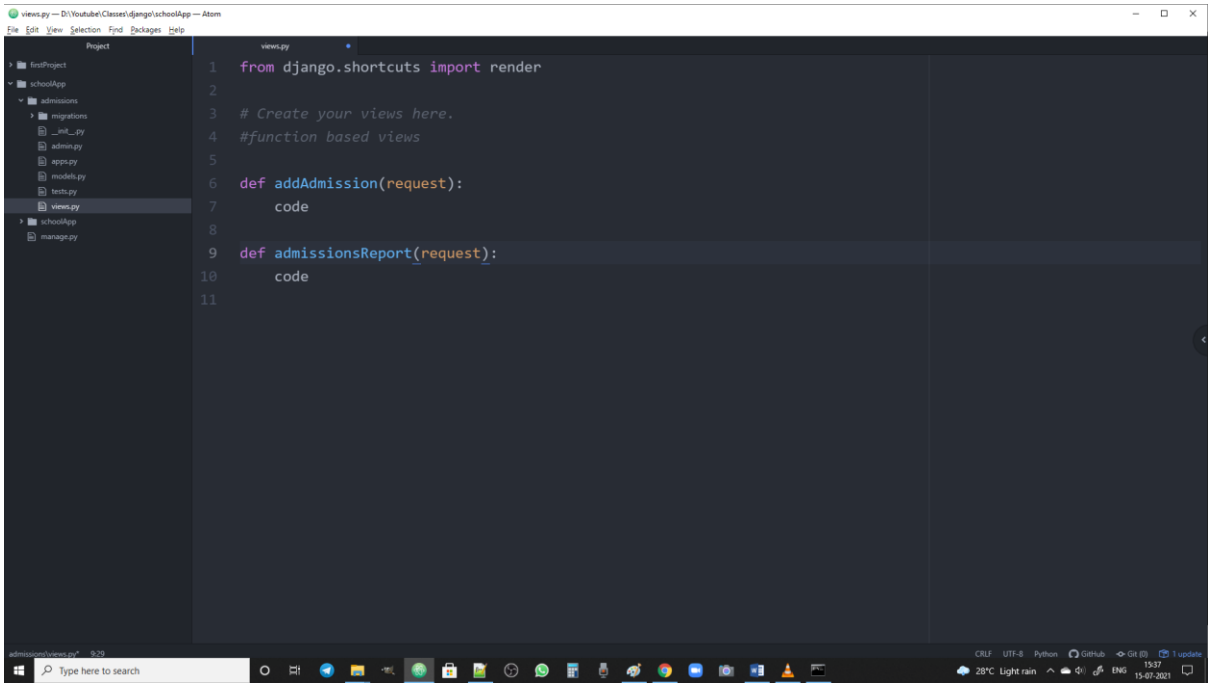
Step-3:Create views under the project

We can create the views by using functions (Function based views) or classes (Class Based Views) in views.py file. First let us discuss function-based views and then in further classes we will discuss about class-based views.

A function based view can be written as follows:

def viewname():

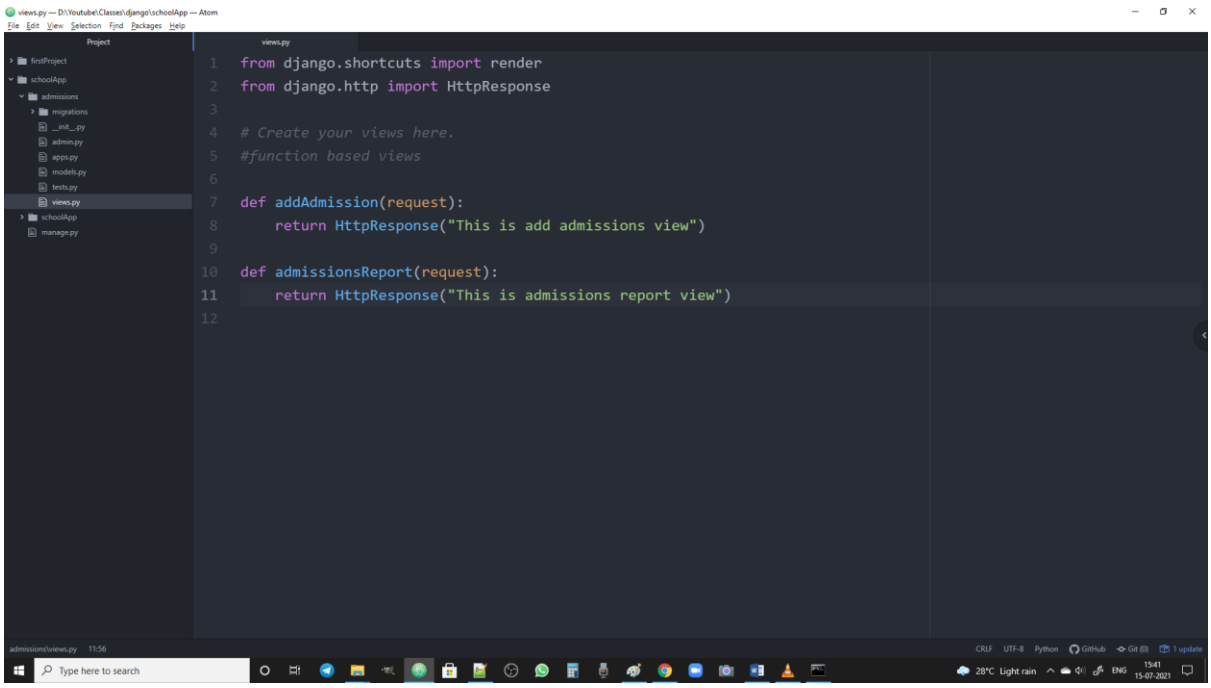
view code



Let's fill the code later. Here request corresponds to the request which is coming from clients.

Servers generally calls the corresponding views based on request from clients so that the particular view will be executed. All the parameters that are forwarded with request are available with this request argument.

To send response back to the client, Django uses HttpResponse.

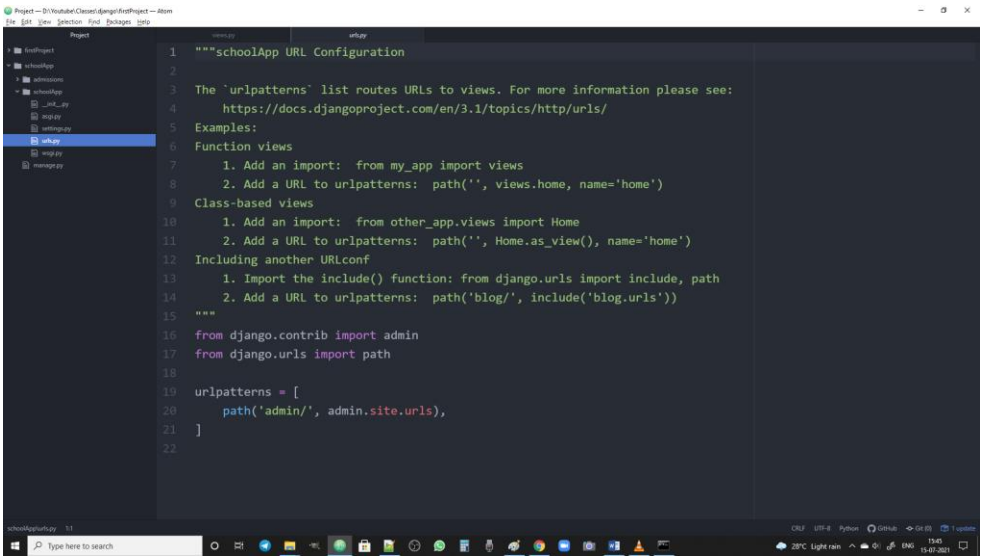


```python
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
#function based views

def addAdmission(request):
    return HttpResponse("This is add admissions view")

def admissionsReport(request):
    return HttpResponse("This is admissions report view")
```

Now whenever client calls addAdmission, then “This is add admissions view” will be sent as response. In the same way whenever client calls admissionsReport then “This is admissions report view” will be displayed as response.

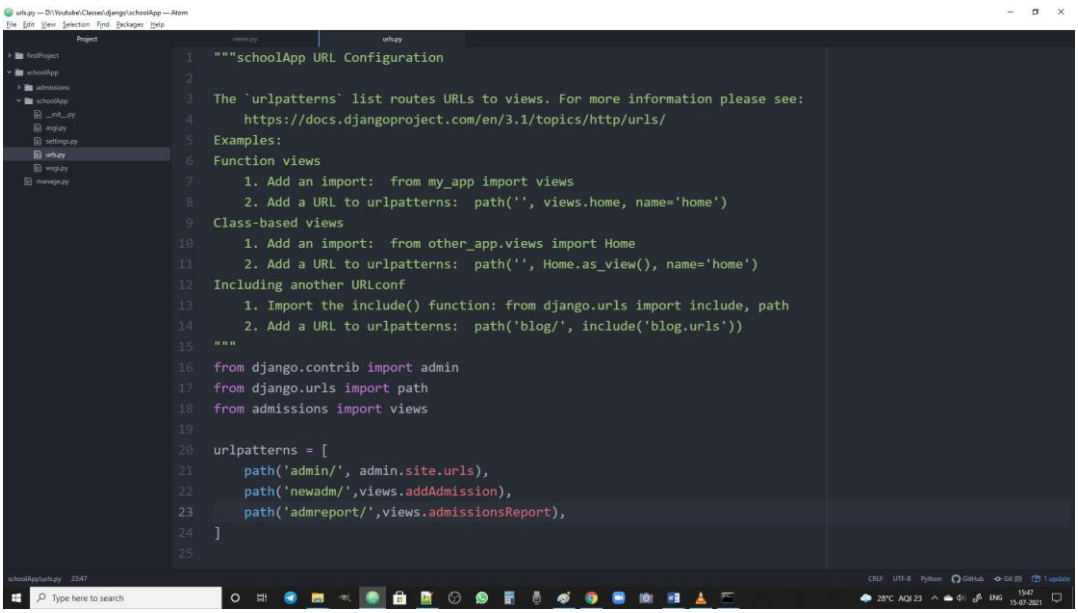
Client send the request to the server via urls. so we need to map each url with a view. These mappings can be done in urls.py file.

Step-4: link views with urls in urls.py file



```python
"""schoolApp URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

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

here under url-patterns, we will write the mappings between views and urls as follows



```
"""schoolApp URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from admissions import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('newadm/',views.addAdmission),
    path('admreport/',views.admissionsReport),
]
```

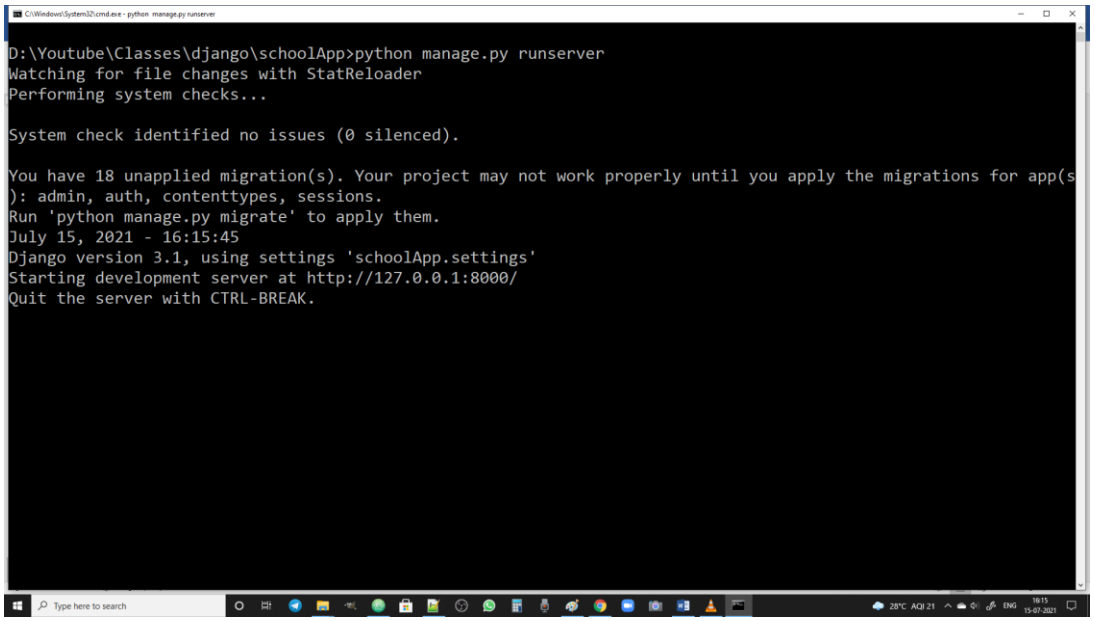
Observe here that

url newadm/ is mapped with addAdmissions view &

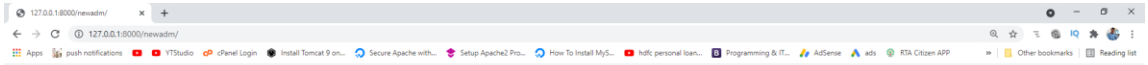
url admreport/ is mapped with admissionsReport

Step-5: Run the project

Now we can run the server and execute the project



```
D:\Youtube\Classes\django\schoolApp>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s
): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
July 15, 2021 - 16:15:45
Django version 3.1, using settings 'schoolApp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

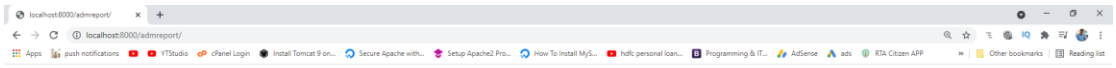


We can run our application as
http://127.0.0.1:8000/our_url

or

http://localhost:8000/our_url

please observe when we request /newadm then it checks in the urls.py with which view this particular view is mapped. then addAdmission view will be executed.



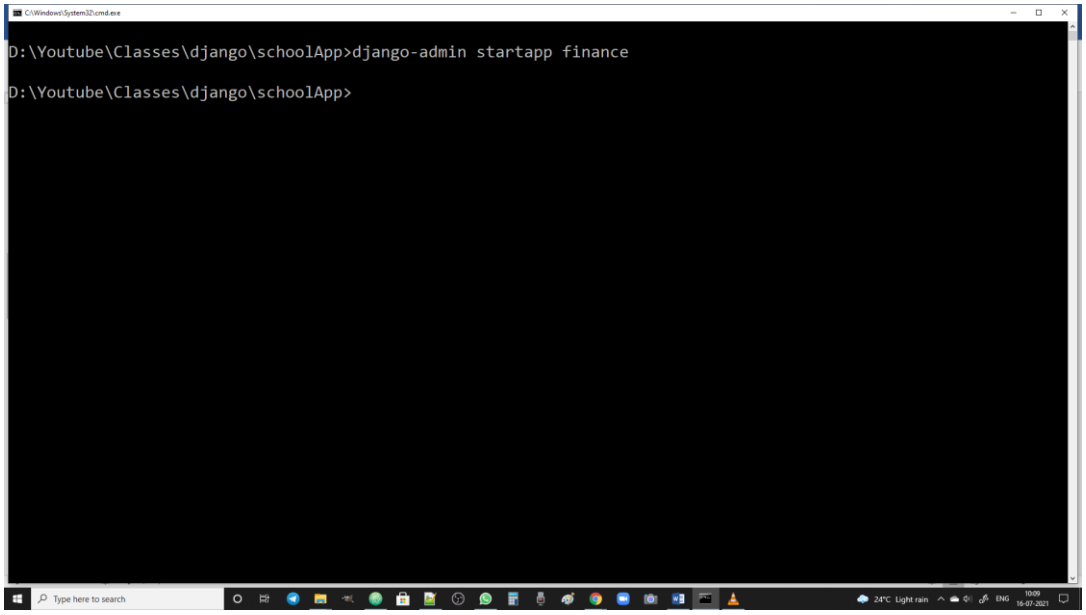
In the same way when client request with admreport/ url, then admissionsReport view will be executed.

Observe here that instead of ip address 127.0.0.1 we used localhost. (Both are same)

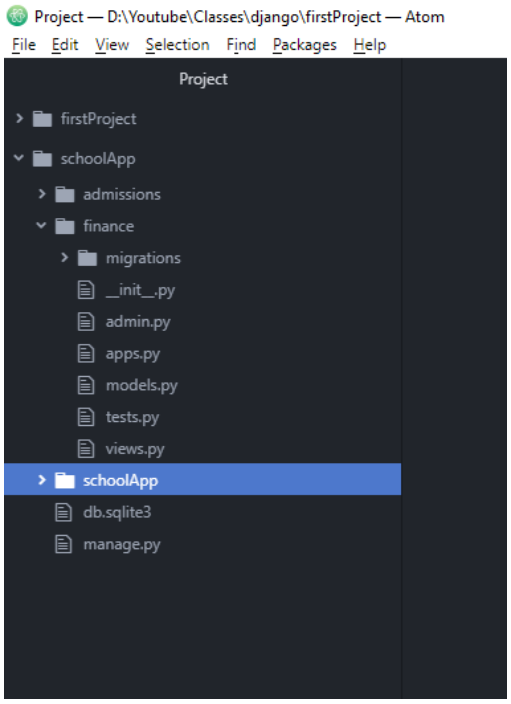
ADDING MULTIPLE APPLICATIONS TO OUR DJANGO PROJECT

Adding multiple applications to the existing project, is same as just discussed.

Now let us add finance project to our schoolApp project



finance app is created now.



Now let us add views to the finance application. These will be written in views.py of finance folder

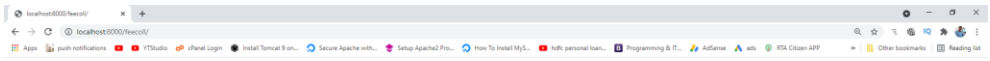
```python
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def feeCollection(request):
    return HttpResponse("I will collect the fee from this view")

def feeDuesReport(request):
    return HttpResponse("I will get fee dues report from this view")

def feeCollectionReport(request):
    return HttpResponse("I will get fee collection report from this view")
```

Now let us map the urls

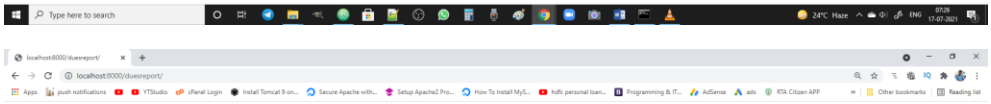
```python
"""
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from admissions import views as ad
from finance import views as fin

urlpatterns = [
    path('admin/', admin.site.urls),
    path('newadm/',ad.addAdmission),
    path('admreport/',ad.admissionsReport),

    path('feecoll/', fin.feeCollection),
    path('duesreport/',fin.feeDuesReport),
    path('collectionsreport/',fin.feeCollectionReport),
]
```

Observe here that, while importing views in finance, we introduced alias names ad,fin which corresponds to admissions, finance respectively. Otherwise both views (admissions, finance) will collided with each other

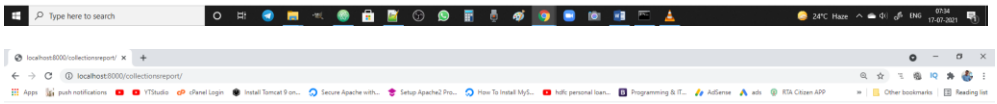
Now let us start our server and run the program.



I will collect the fee from this view



I will get fee dues report from this view



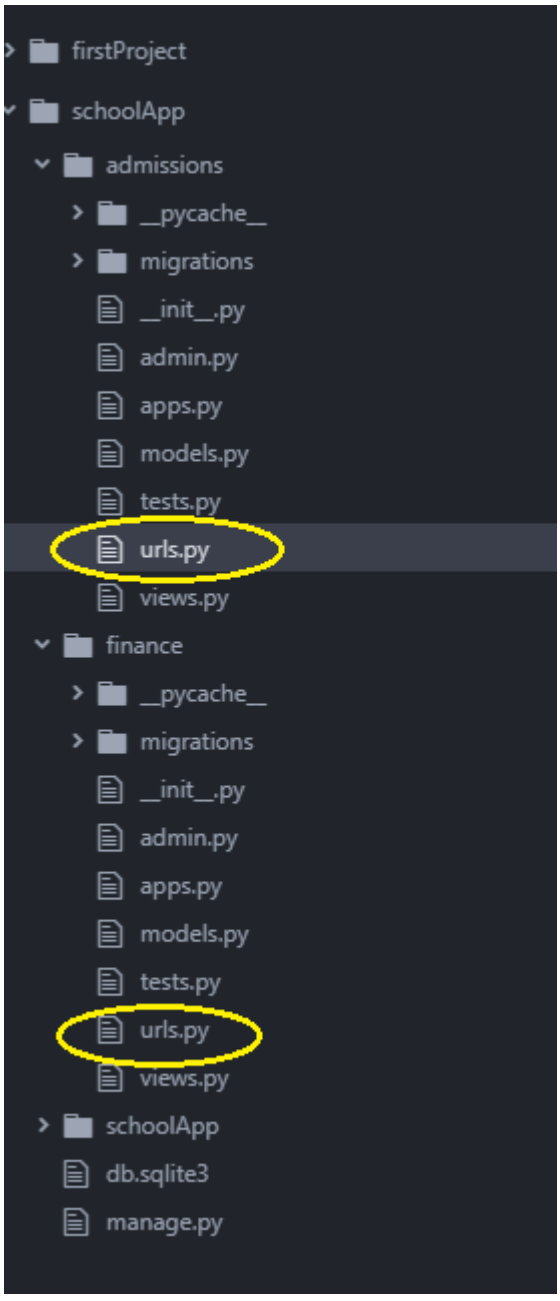
I will get fee collection report from this view

APPLICATION LEVEL URL MAPPINGS

Instead of writing all url mappings in the project level url mappings file (urls.py) we can do mappings in application level urls.py files also.

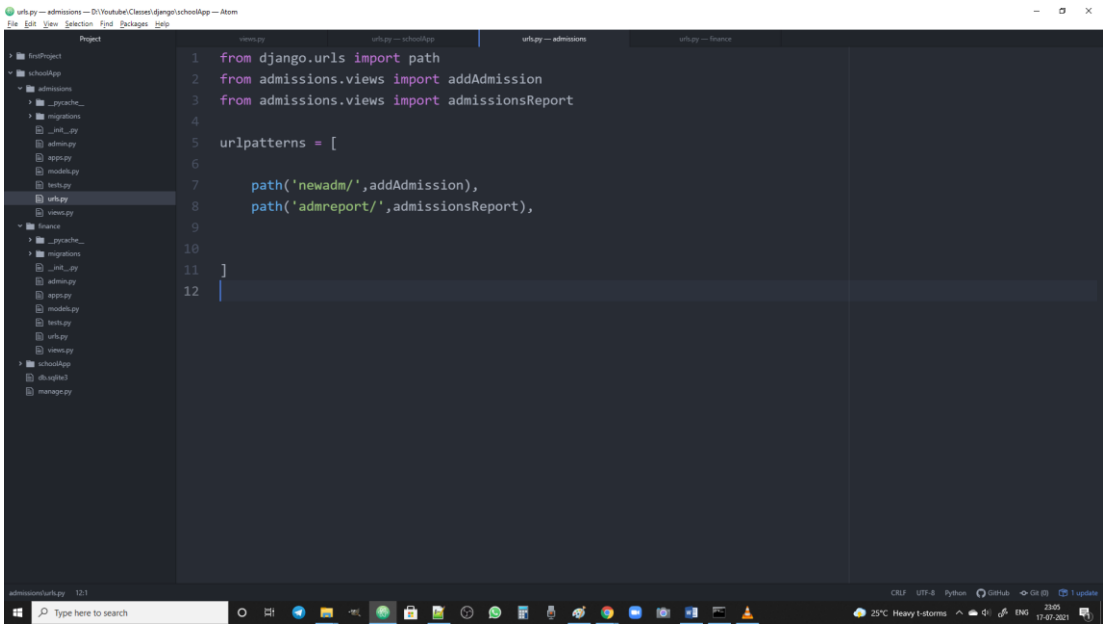
This will reduce the burden of using alias names (ad, fin which we used in previous topic).

First let us create urls.py files under each application

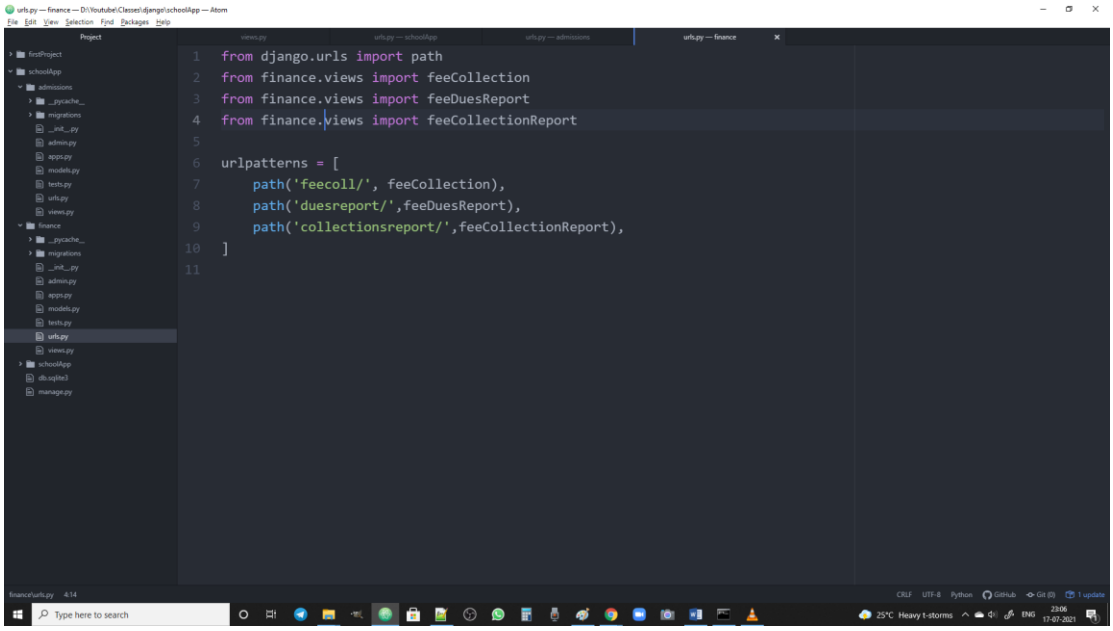


now let us implement application level mappings

The following is the admissions application's url mappings file

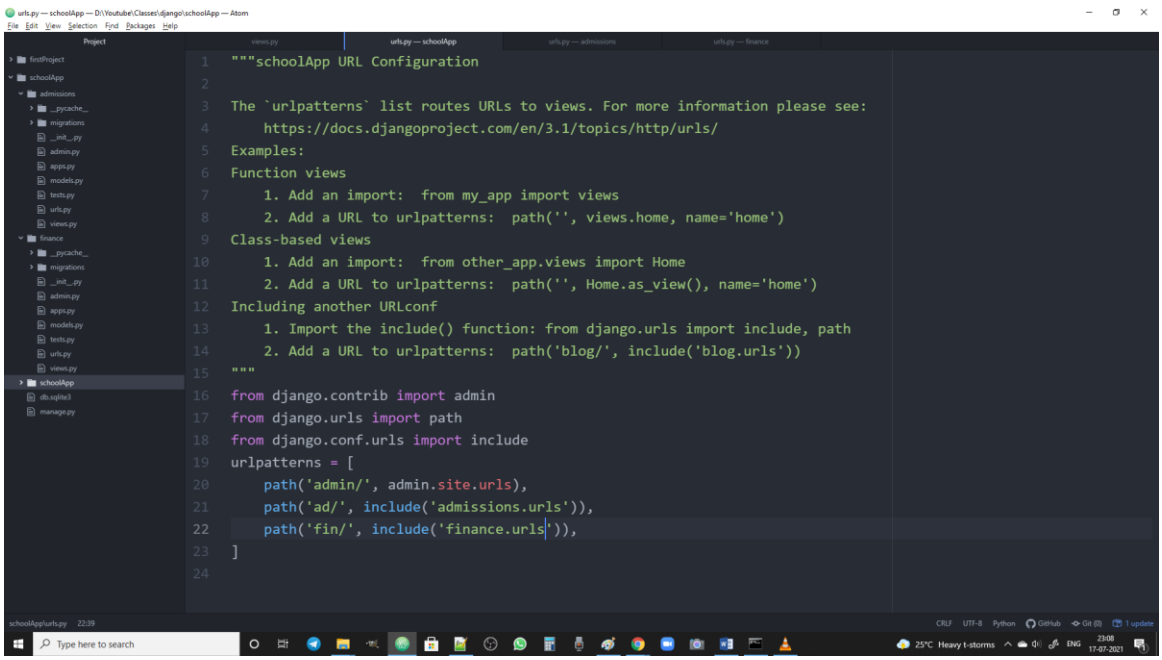


Following is the finance application's mappings



observe here that in each url mapping file, we individually imported that application's views and implemented the corresponding mappings.

now let us see the project level url mappings file.



```
"""schoolApp URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from django.conf.urls import include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('ad/', include('admissions.urls')),
    path('fin/', include('finance.urls')),
]
```

Above pic shows project level url mappings.

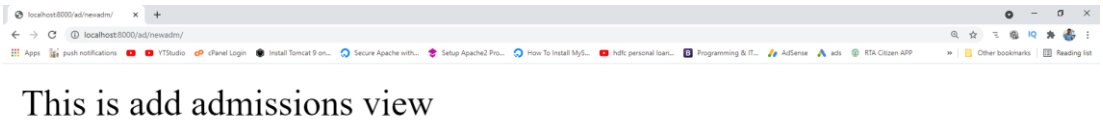
As application level mappings are already separated, we just need to include those url mappings into our project level urls.py by using include function.

include is available in django.conf.urls

Now we will run the application as follows

whenever we want to run admissions application's url newadm we need to add these project level mapping /ad/ to the actual url as follows.

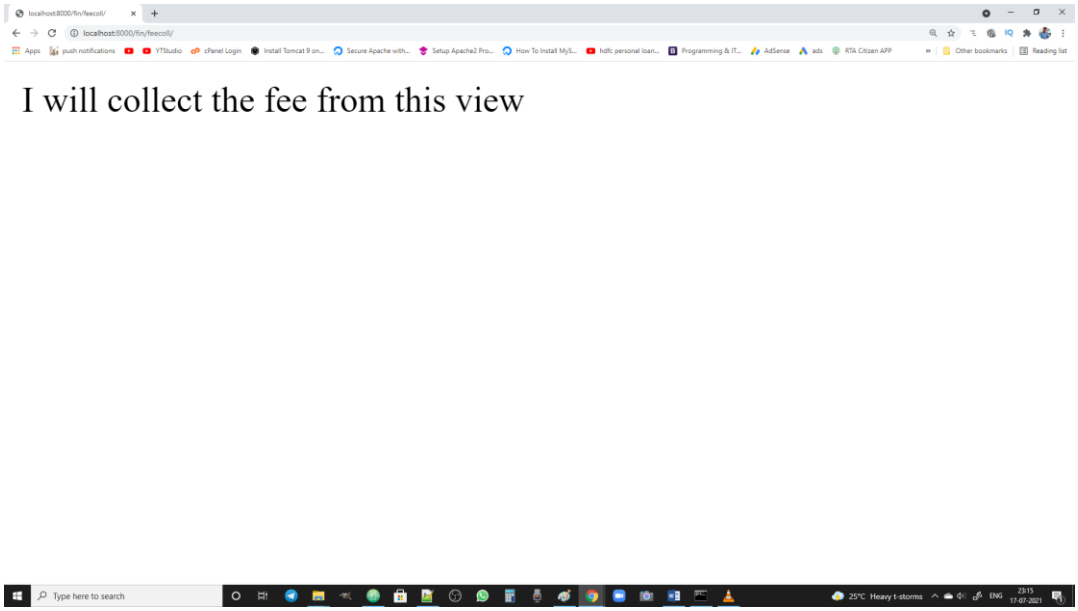
http://localhost:8000/ad/newadm



This is add admissions view

In the same way, whenever we want to run finance application's url feecoll we need to add these project level mapping /fin/ to the actual url as follows.

http://localhost:8000/fin/feecoll



CREATING TEMPLATES IN DJANGO

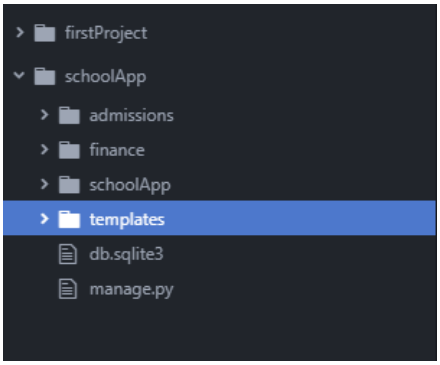
We use templates to generate dynamic responses.

Steps involved in creating templates

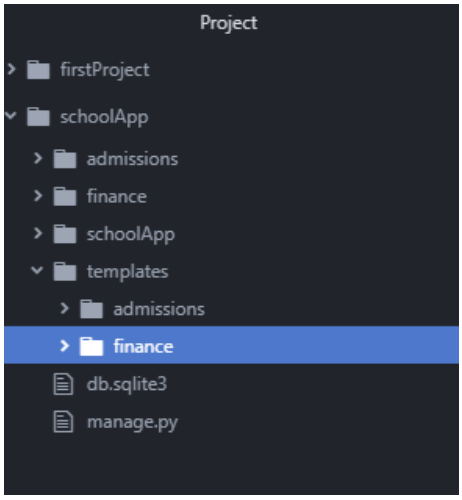
1) Setup the templates folder
2) setup DIRS in settings.py
3) Create a template
4) Use template in views
5) execute the project

Setup the templates folder:

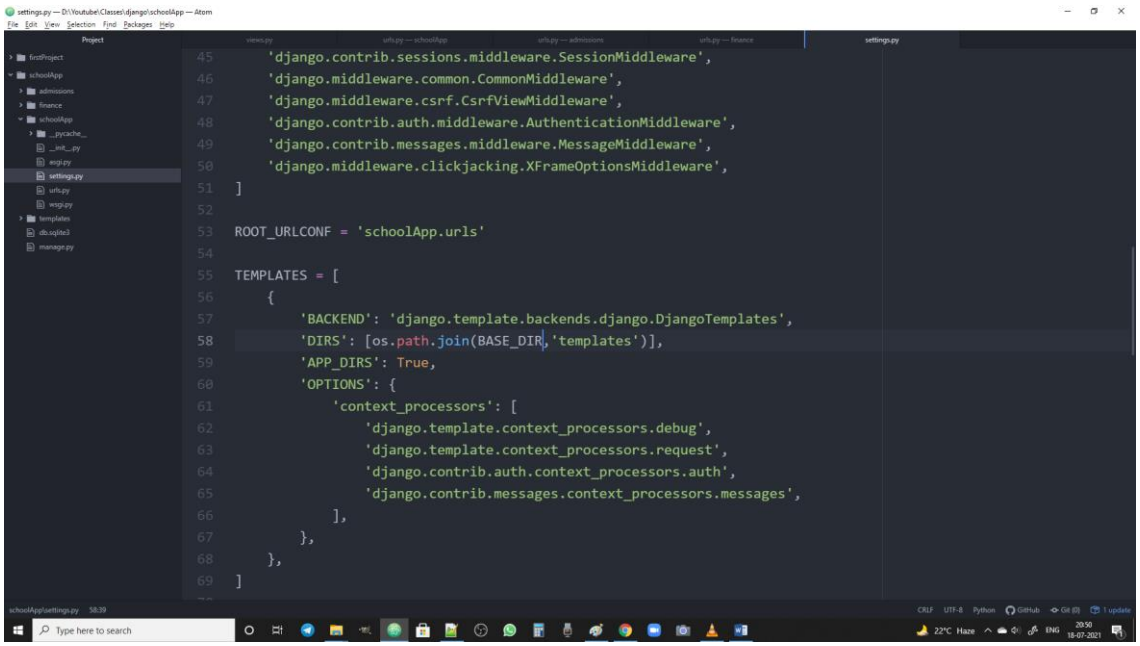
Create templates folder under project folder 'schoolApp'



To maintain application wise templates separately, let us create two subfolders under templates with the names admissions, finance



Setup DIRS in settings.py:



```
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'schoolApp.urls'

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

As shown above, we need to set the DIRS of TEMPALTES in settings.py as

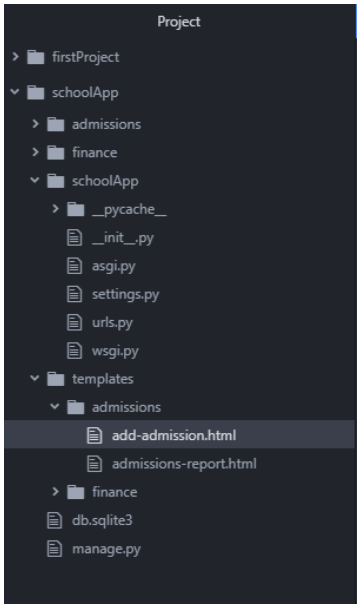
'DIRS': [os.path.join(BASE_DIR,'templates')],

which means that we are informing to the system that we will place all our templates under folder called ‘templates’ which is available in project directory (BASE_DIR indicates project directory).

Before doing above setting, we need to import os module in order to set the templates.

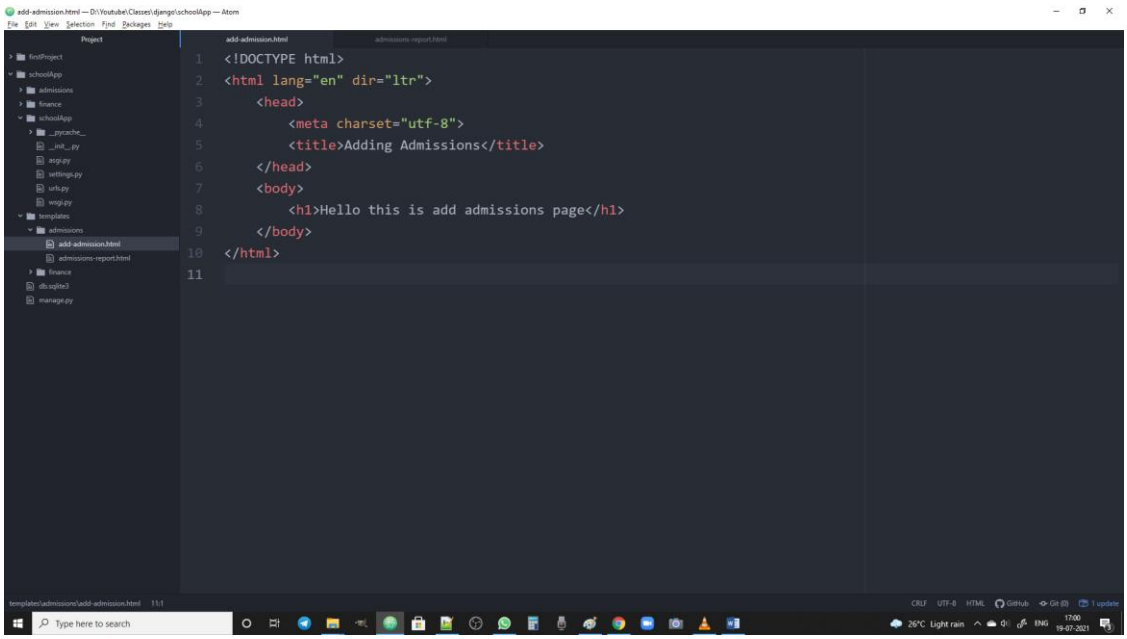
Creating Templates:

Templates are nothing but html files. Now let us create our templates under templates folder.

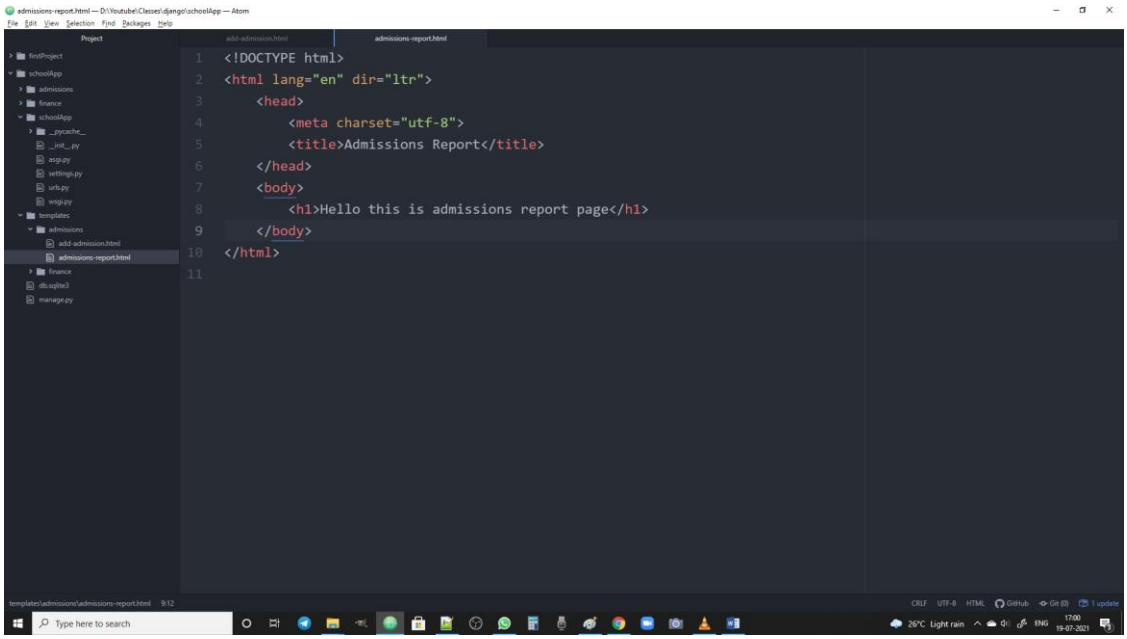


We created two template files (add-admission.html, admissions-report.html) under admissions folder of templates.

Below codes shows how these two html files were Implemented.



```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding Admissions</title>
    </head>
    <body>
        <h1>Hello this is add admissions page</h1>
    </body>
</html>
```

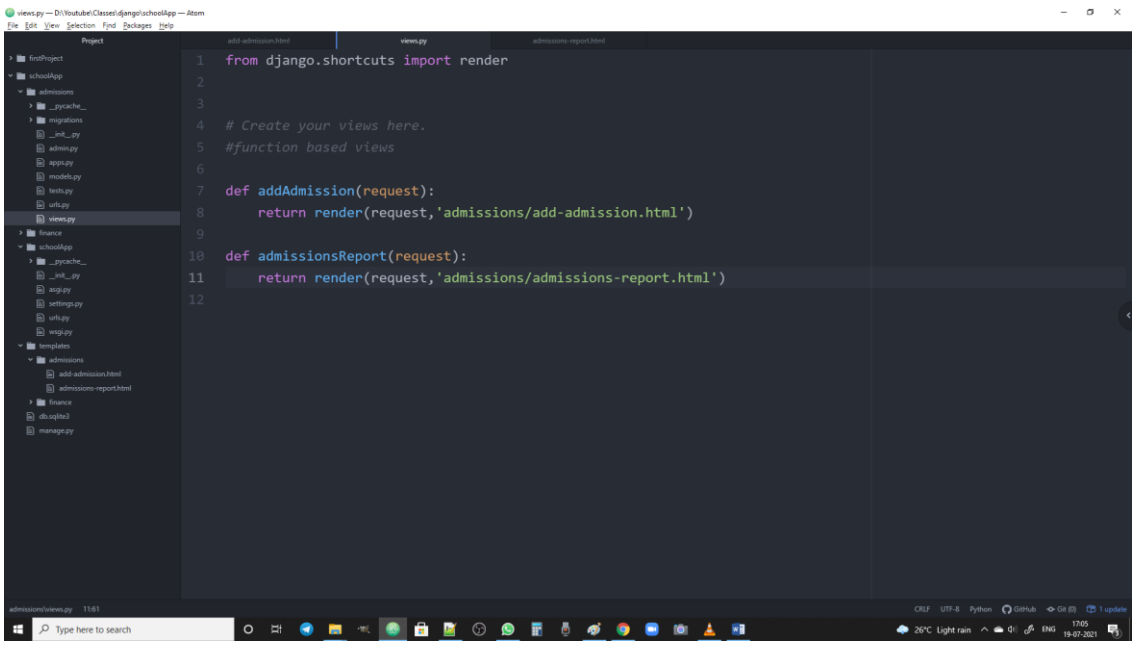


```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Admissions Report</title>
    </head>
    <body>
        <h1>Hello this is admissions report page</h1>
    </body>
</html>
```

Using templates in views:

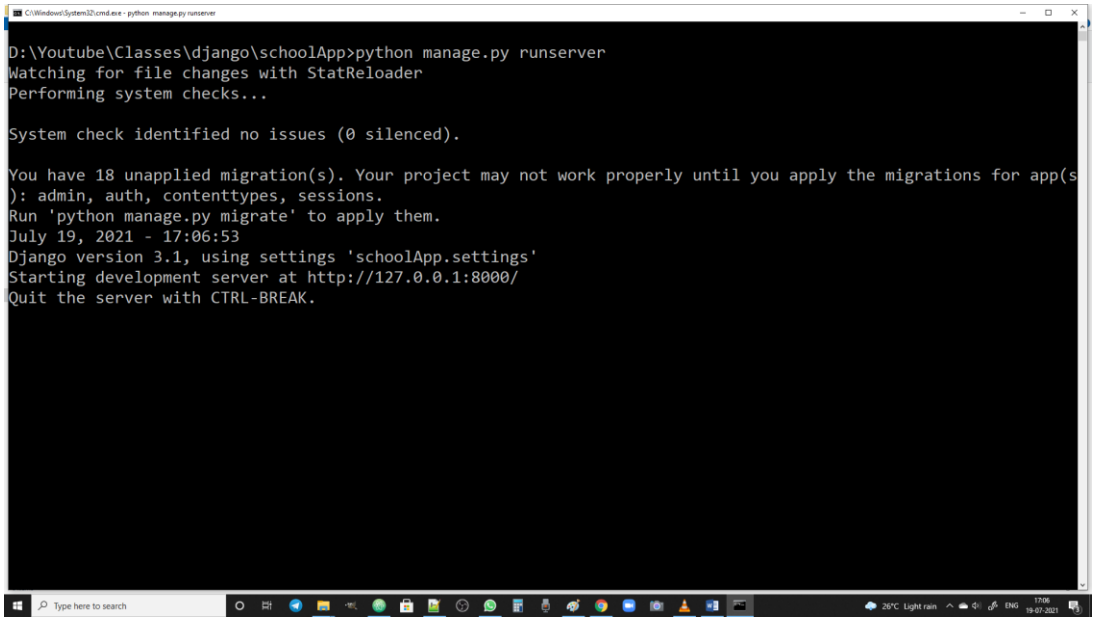
By using render, we can link templates and views.

render (request, template)



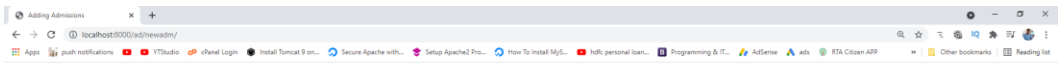
```
from django.shortcuts import render


# Create your views here.
#function based views

def addAdmission(request):
    return render(request,'admissions/add-admission.html')

def admissionsReport(request):
    return render(request,'admissions/admissions-report.html')
```

Execute the project:

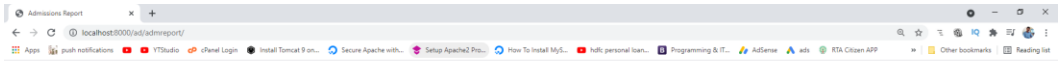


```
D:\Youtube\Classes\django\schoolApp>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s
): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
July 19, 2021 - 17:06:53
Django version 3.1, using settings 'schoolApp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Now let us run the application and check the output is taken from html templates or not.



Hello this is add admissions page

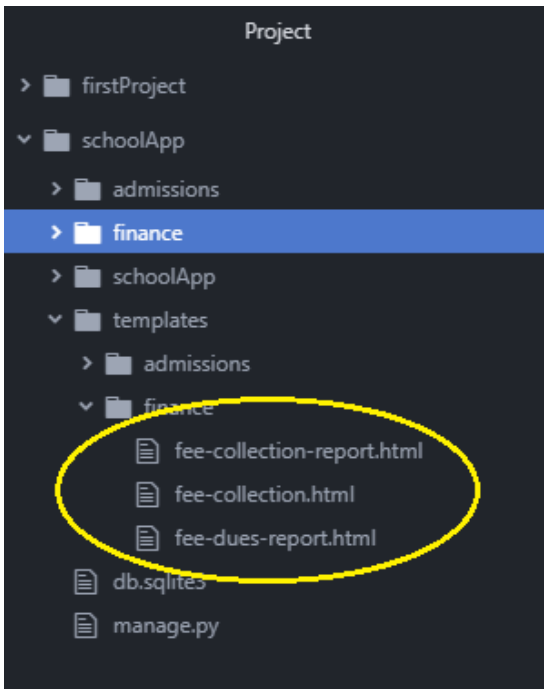


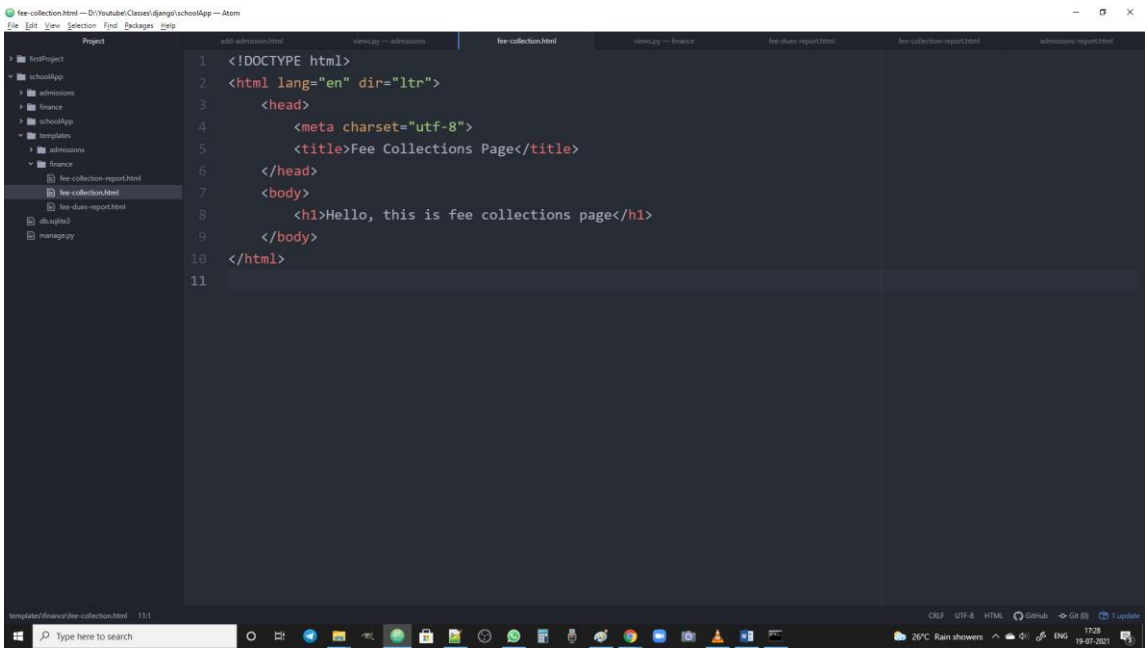
Hello this is admissions report page

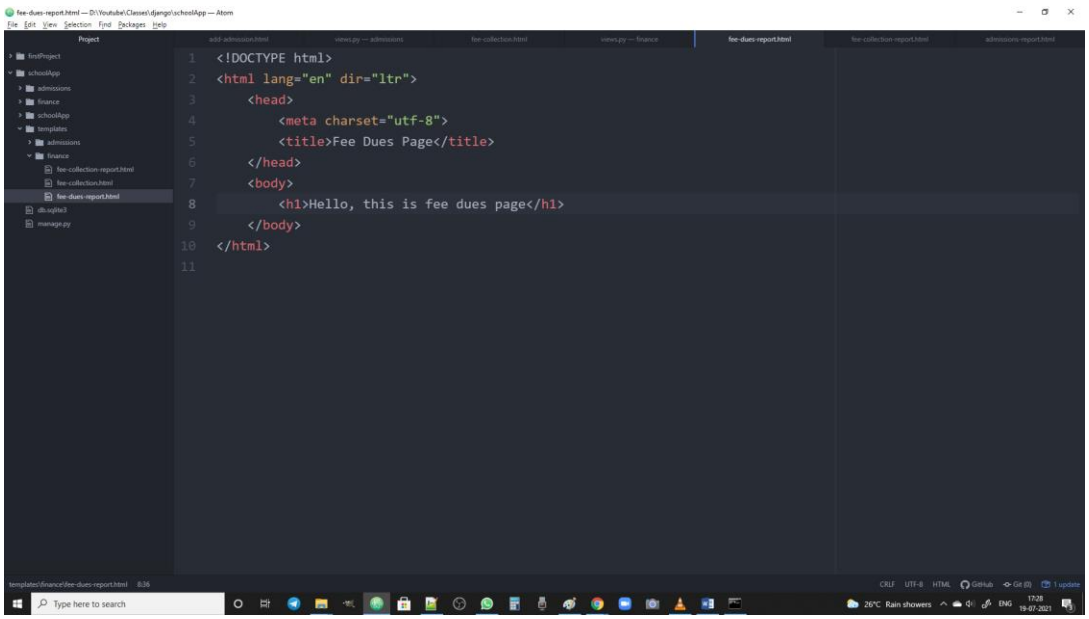
Observe here that views are linked with templates successfully.

Whenever a view is activated, corresponding template is displayed as output.

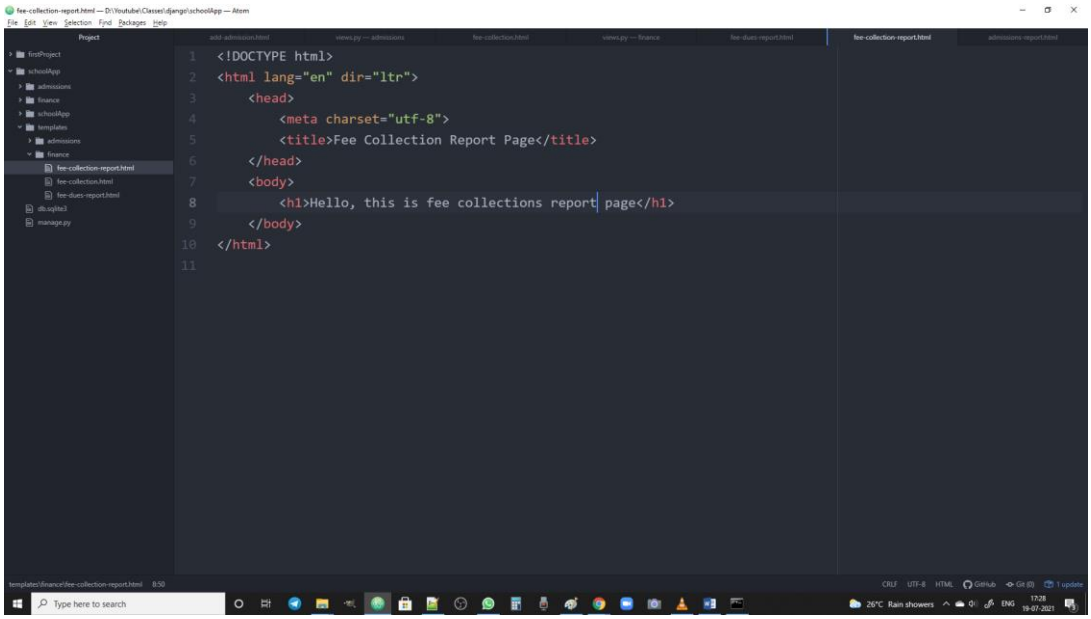
In the same way we can create three templates under finance folder of templates, and link those templates with views in views.py under finance application.


Project
firstProject
schoolApp
admissions
finance
schoolApp
templates
admissions
finance
fee-collection-report.html
fee-collection.html
fee-dues-report.html
db.sqlite3
manage.py

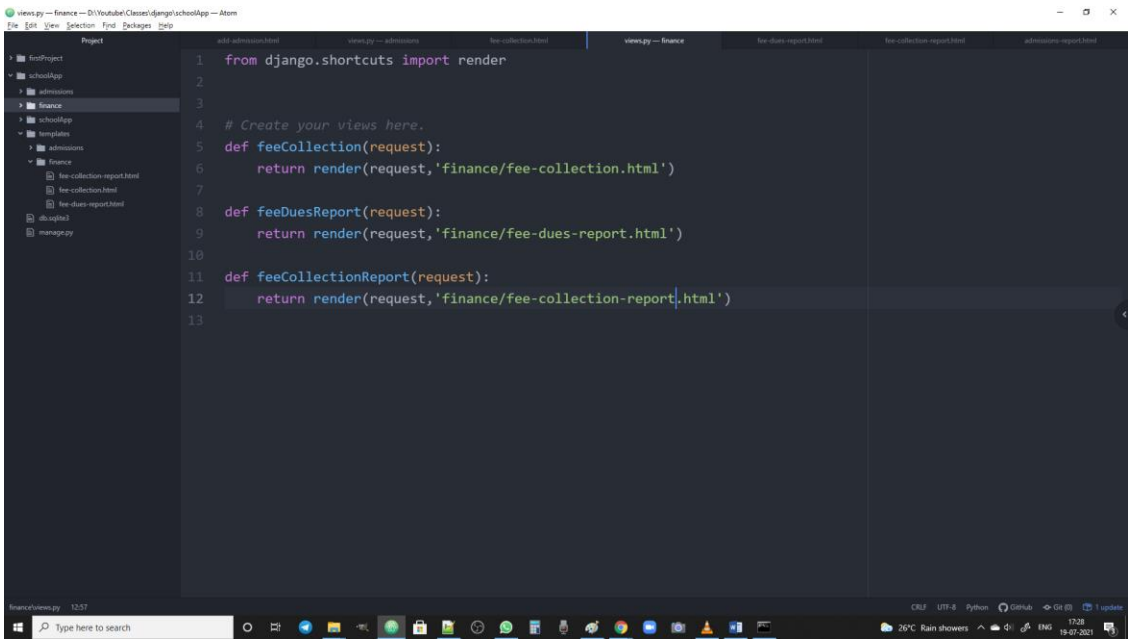

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
<meta charset="utf-8">
<title>Fee Collections Page</title>
</head>
<body>
<h1>Hello, this is fee collections page</h1>
</body>
</html>



```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Fee Dues Page</title>
    </head>
    <body>
        <h1>Hello, this is fee dues page</h1>
    </body>
</html>
```

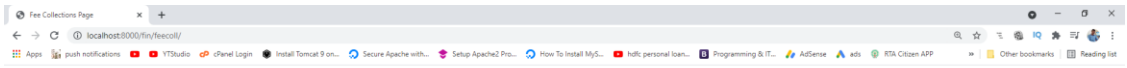


```
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Fee Collection Report Page</title>
    </head>
    <body>
        <h1>Hello, this is fee collections report page</h1>
    </body>
</html>
```



```
from django.shortcuts import render


# Create your views here.
def feeCollection(request):
    return render(request,'finance/fee-collection.html')

def feeDuesReport(request):
    return render(request,'finance/fee-dues-report.html')

def feeCollectionReport(request):
    return render(request,'finance/fee-collection-report.html')
```

Now we can see the finance template files as output when we run finance views (same as admissions part)



Hello, this is fee collections page

SENDING DATA FROM VIEW TO TEMPLATES

We can send data from view to templates so that we will generate the dynamic output by filling these values in the template file.

We generally use dictionaries in order to send the data to the templates as follows.

```
def addAdmission(request):

    values = {"name":"Santosh","age":36,"Address":"Visakhapatnam"}

    return render(request,'admissions/add-admission.html',values)
```

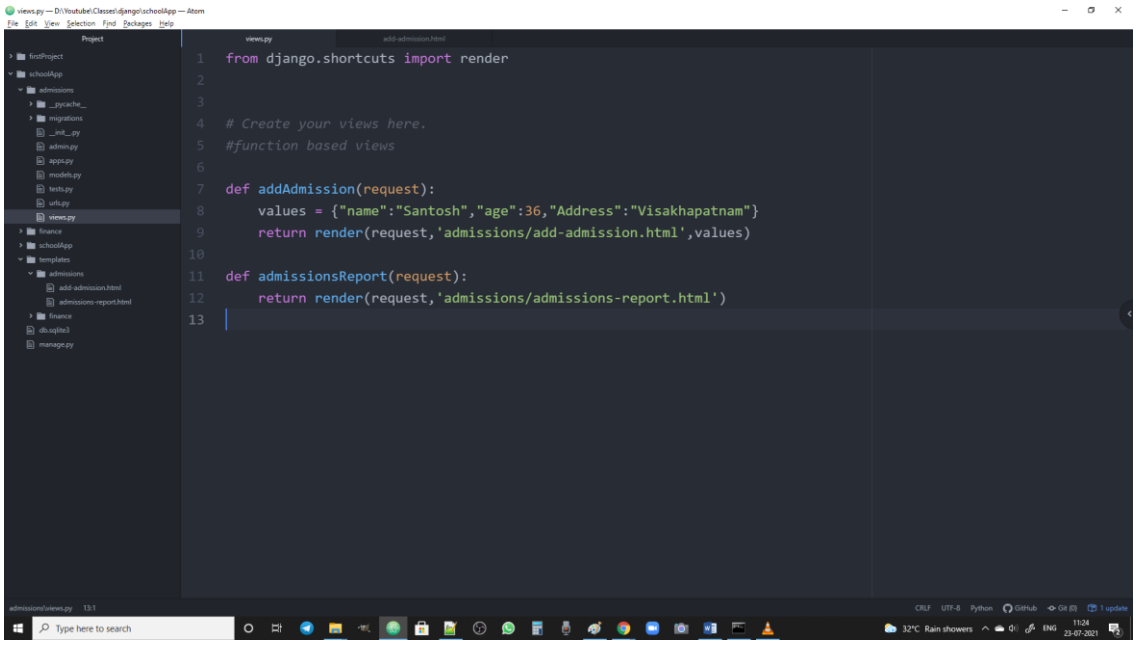
Observe here that a dictionary called 'values' is created and passed to the template through render method.

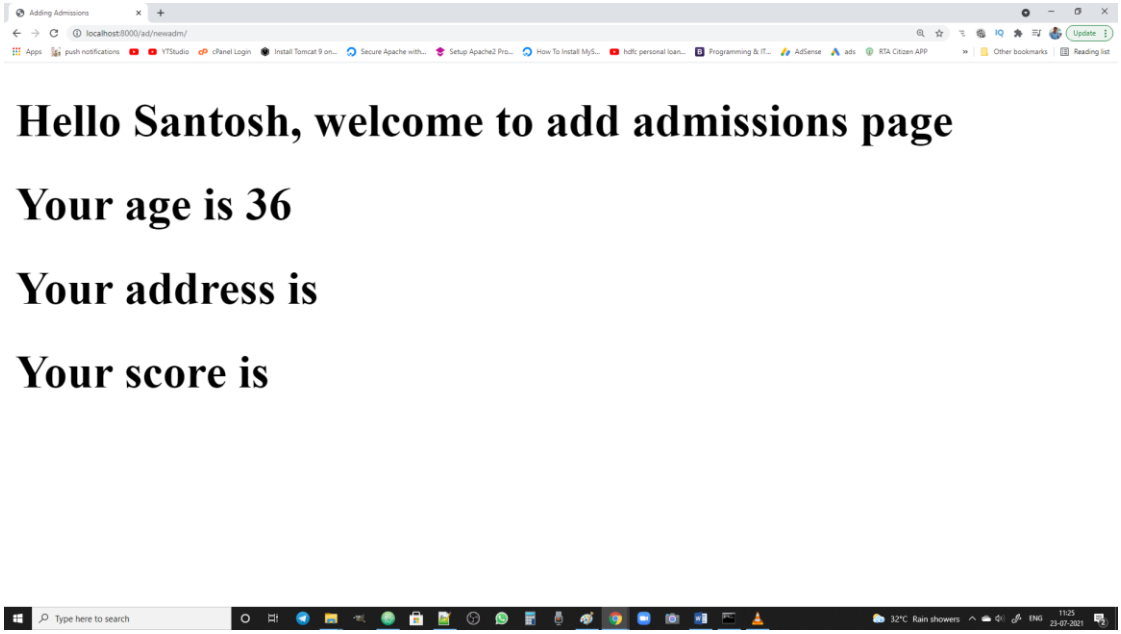
Now this dictionary values will be used in the template add-admission.html and dynamic response will be generated with the values in this dictionary as follows.

for example we can print the value of age in template as

```
{{age}}
```

Observe the full example below





Here observe that response is created dynamically by filling the values received from view.

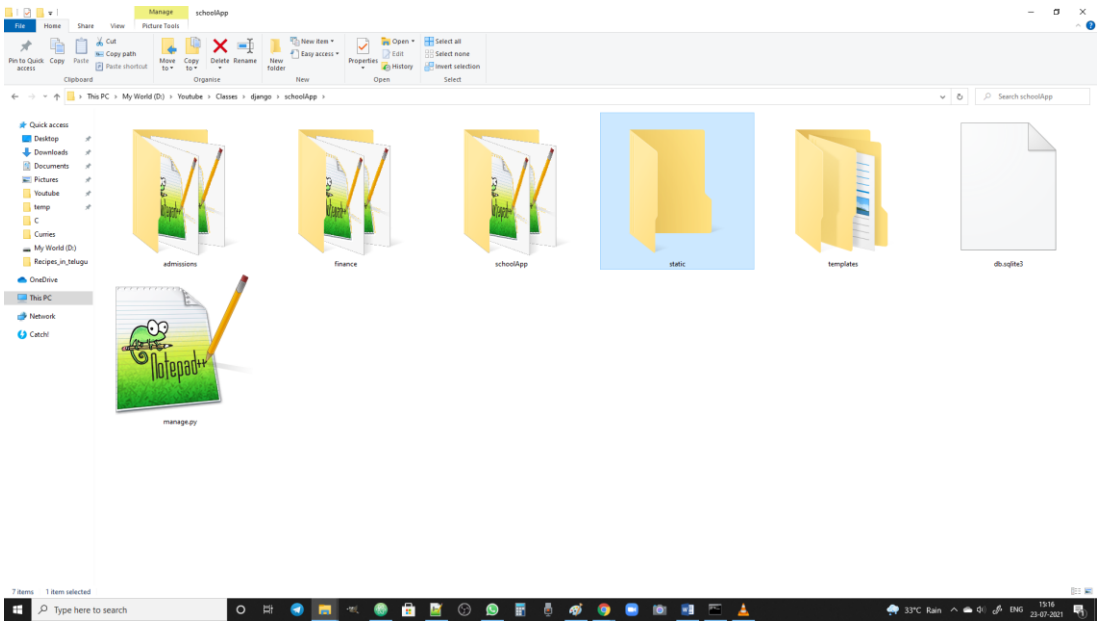
And also observe that score field is not available in the dictionary which is created in views.

So, in template, score is shown as empty as it is not received through dictionary.

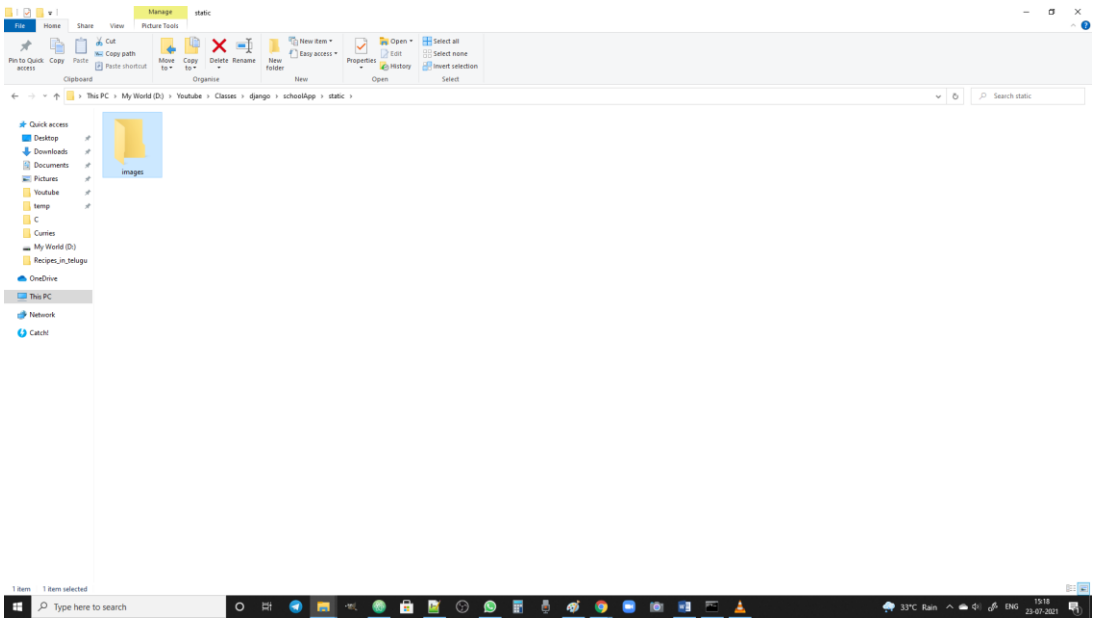
INSERTING IMAGES INTO TEMPLATES

To insert images into our templates,

Step – 1: We need to create a sub folder called ‘static’ in our project’s folder.

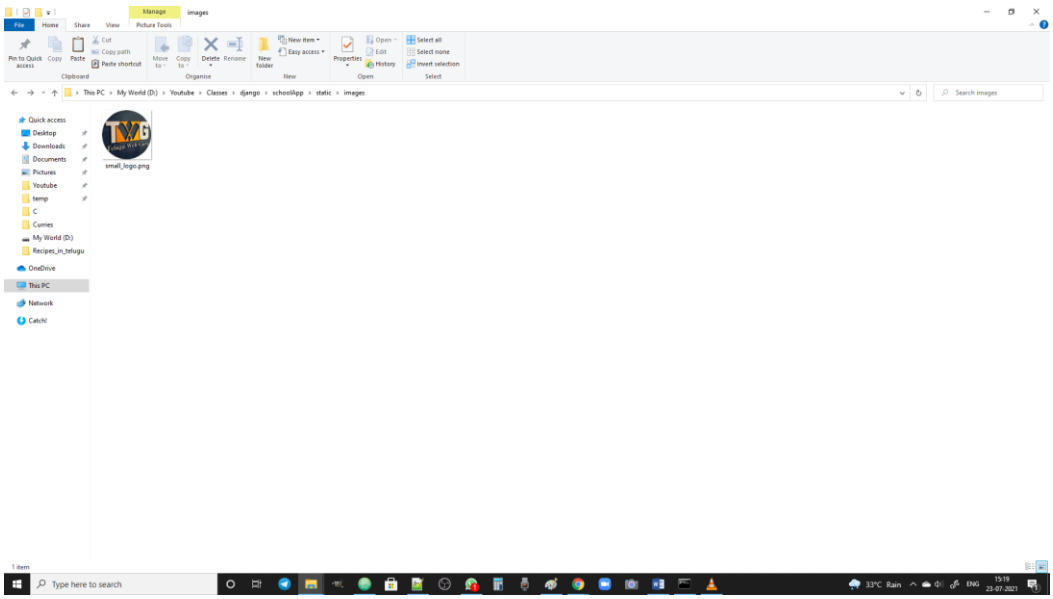


We may maintain separate folders (for images , logos etc.,) inside the static folder

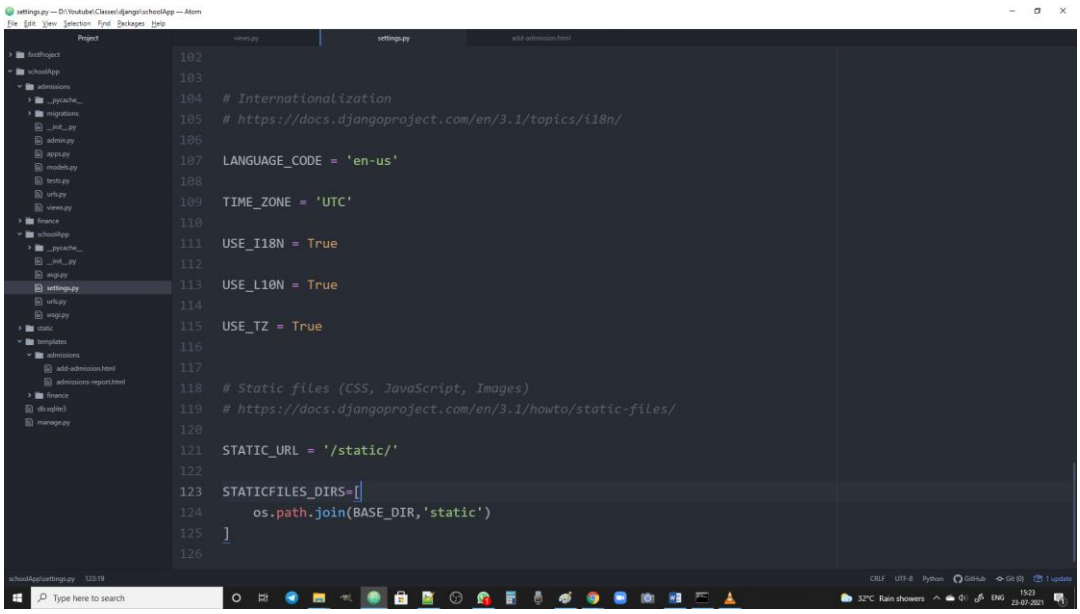


We may store the images in this images folder which are need to be inserted into our templates.

For example I stored one image called small_logo.png as follows.



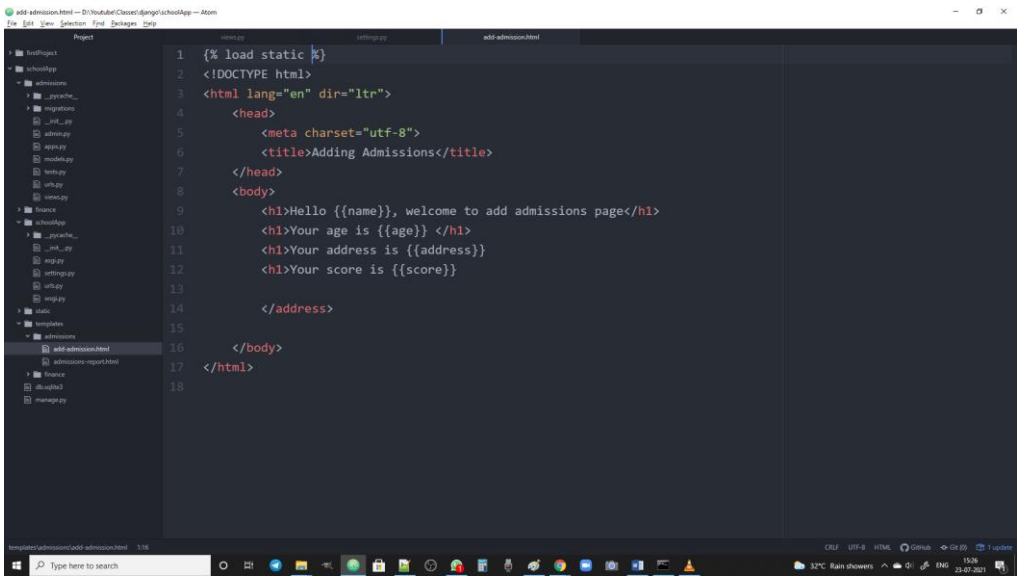
Step-2: Set the STATICFILES_DIRS setting in settings.py



```
# Internationalization
# https://docs.djangoproject.com/en/3.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS=[
    os.path.join(BASE_DIR,'static')
]
```

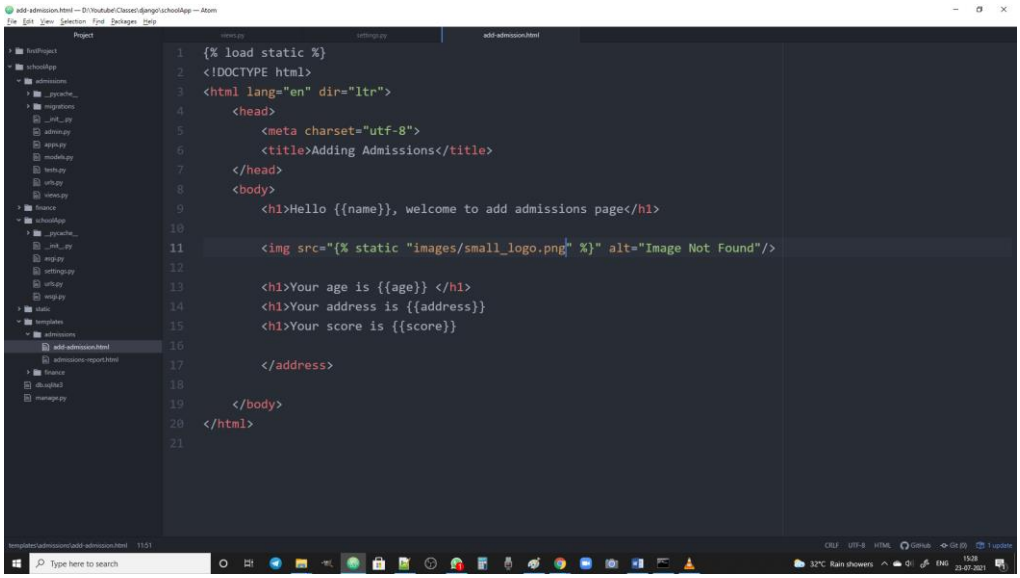
Step-3: Load the static files into templates.

To achieve this, we need to use template tags {%.....%}



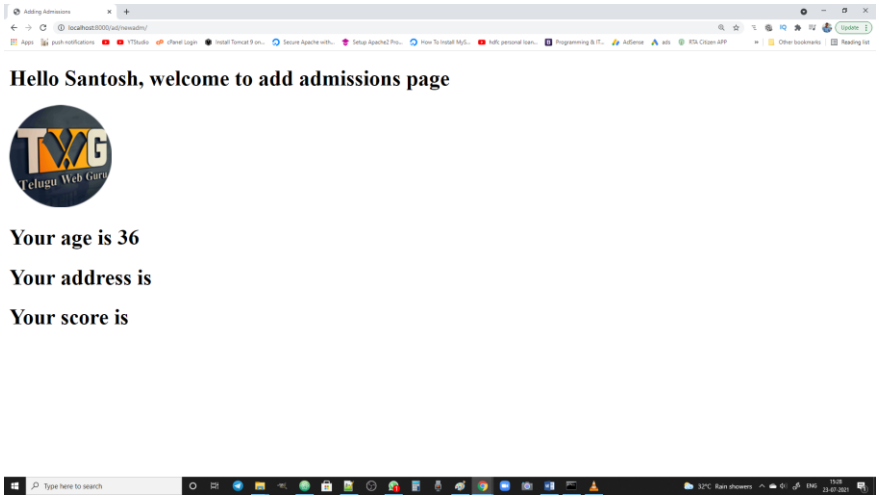
```
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding Admissions</title>
    </head>
    <body>
        <h1>Hello {{name}}, welcome to add admissions page</h1>
        <h1>Your age is {{age}} </h1>
        <h1>Your address is {{address}}
        <h1>Your score is {{score}}

        </address>

    </body>
</html>
```

Step-4: Insert images by using img tag as shown below.



```
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding Admissions</title>
    </head>
    <body>
        <h1>Hello {{name}}, welcome to add admissions page</h1>

        <img src="{% static "images/small_logo.png" %}" alt="Image Not Found"/>

        <h1>Your age is {{age}} </h1>
        <h1>Your address is {{address}}
        <h1>Your score is {{score}}

        </address>

    </body>
</html>
```

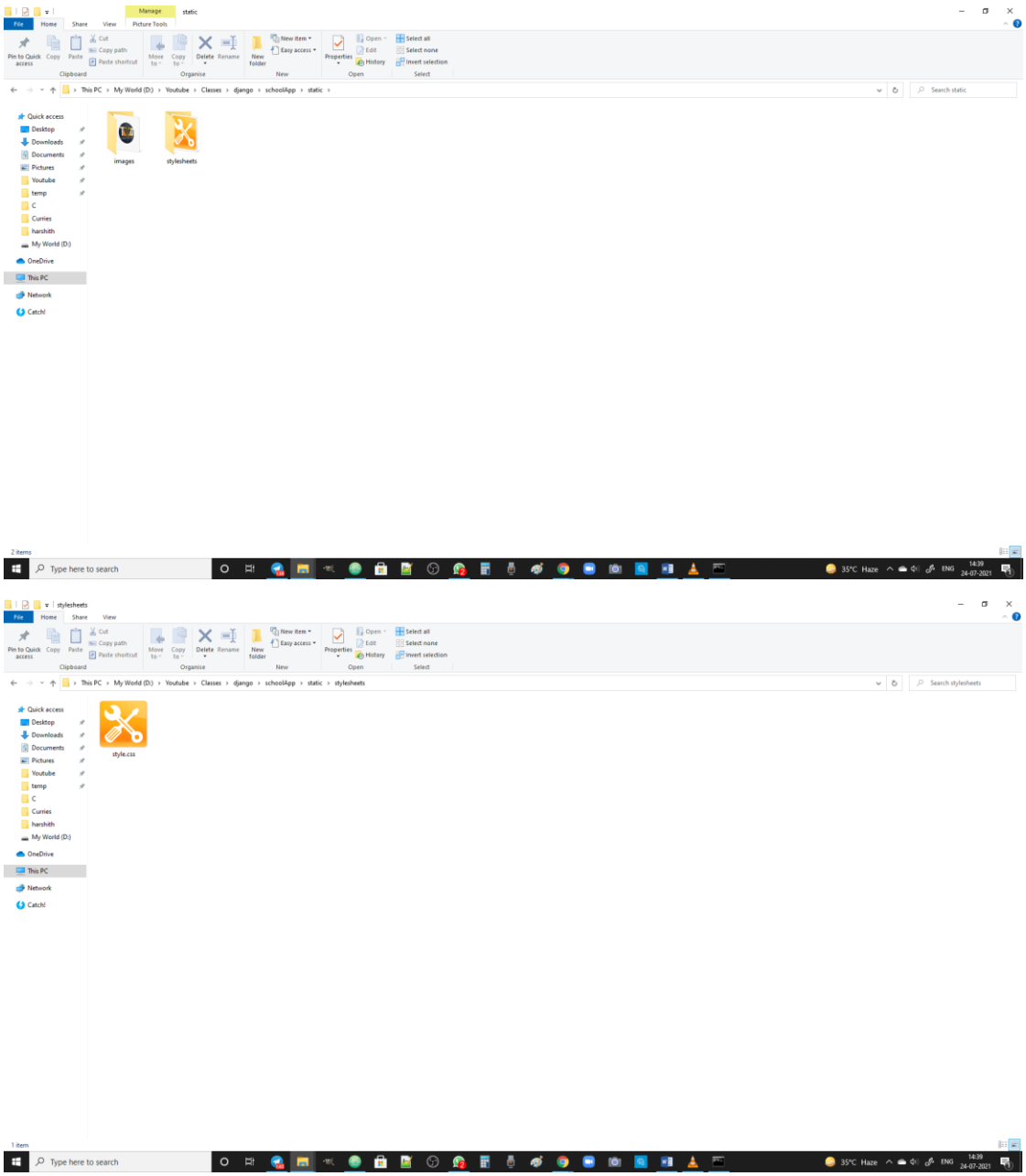
After inserting images using img tag, we can run the server and start our project so that our templates are loaded with images.



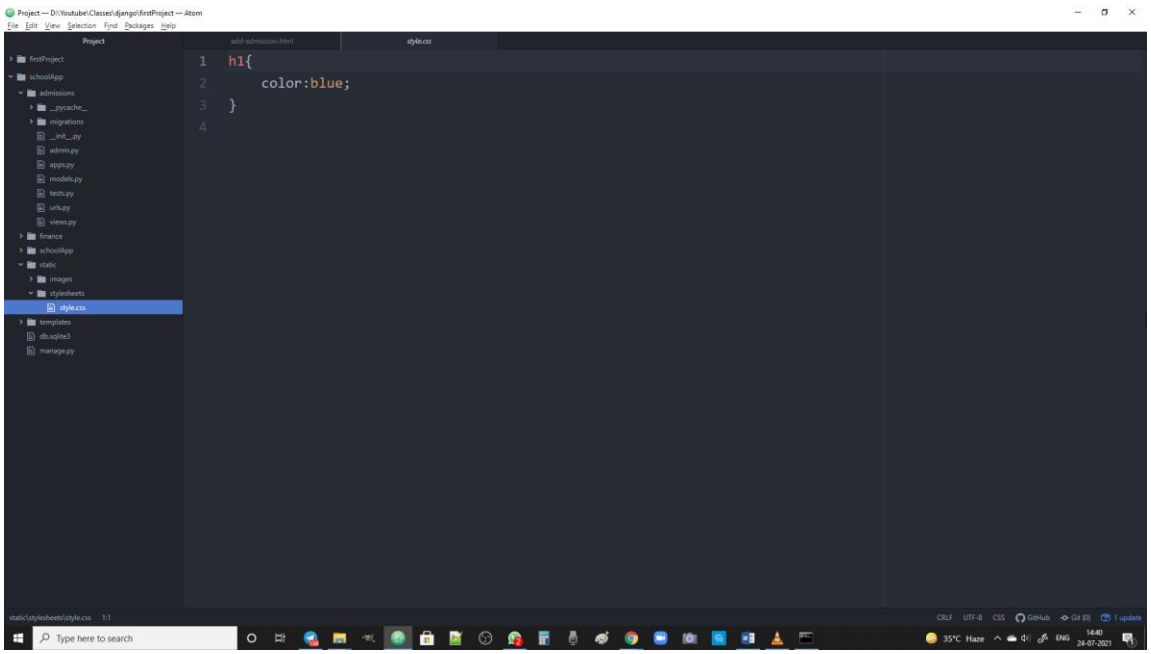
LINKING STYLESHEETS WITH TEMPLATES

Linking stylesheets follows the same procedure, which we followed to insert images into template.

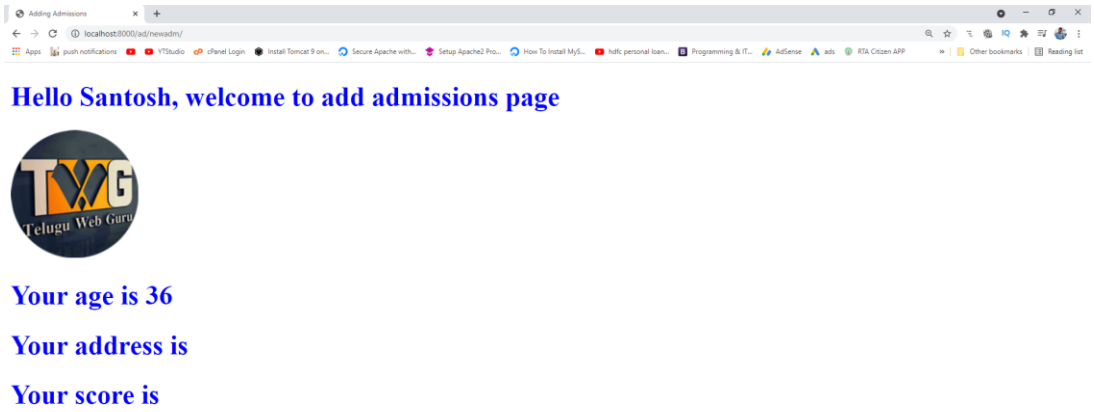
Let us create a folder stylesheet and create a style.css file into it



Now let us write styles code in style sheet file.



Now let us link this css file with our templates so that all the content within h1 tag will be displayed in blue colour.

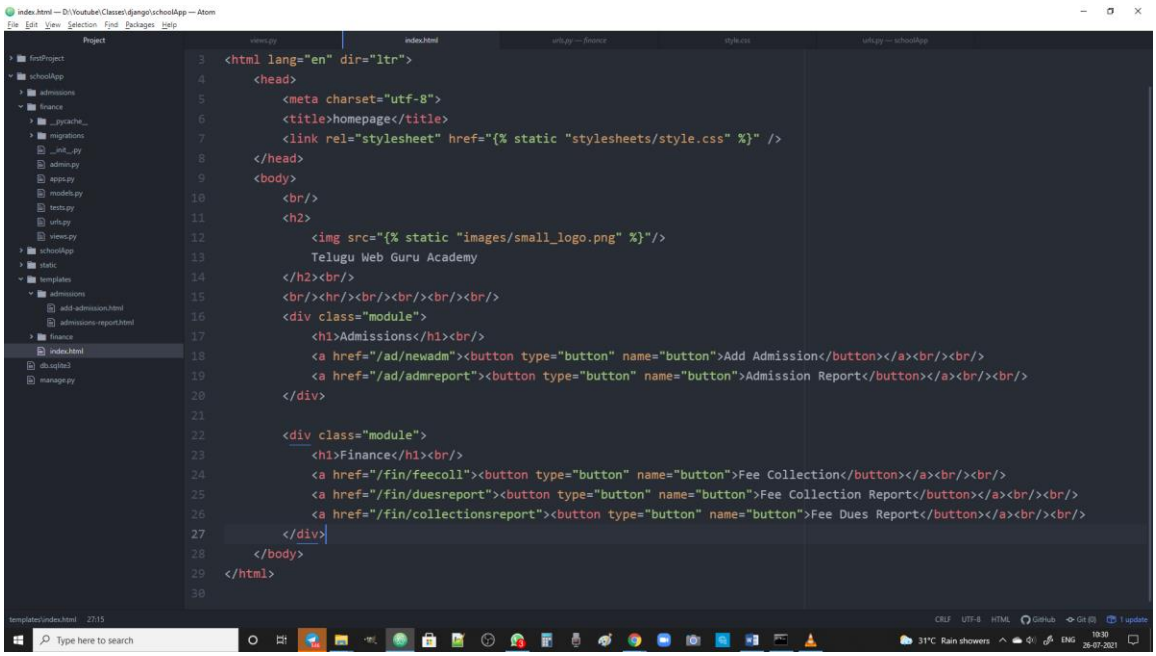


This is how we can link our stylesheets with templates and create beautiful webpages.

SETTING HOME PAGE

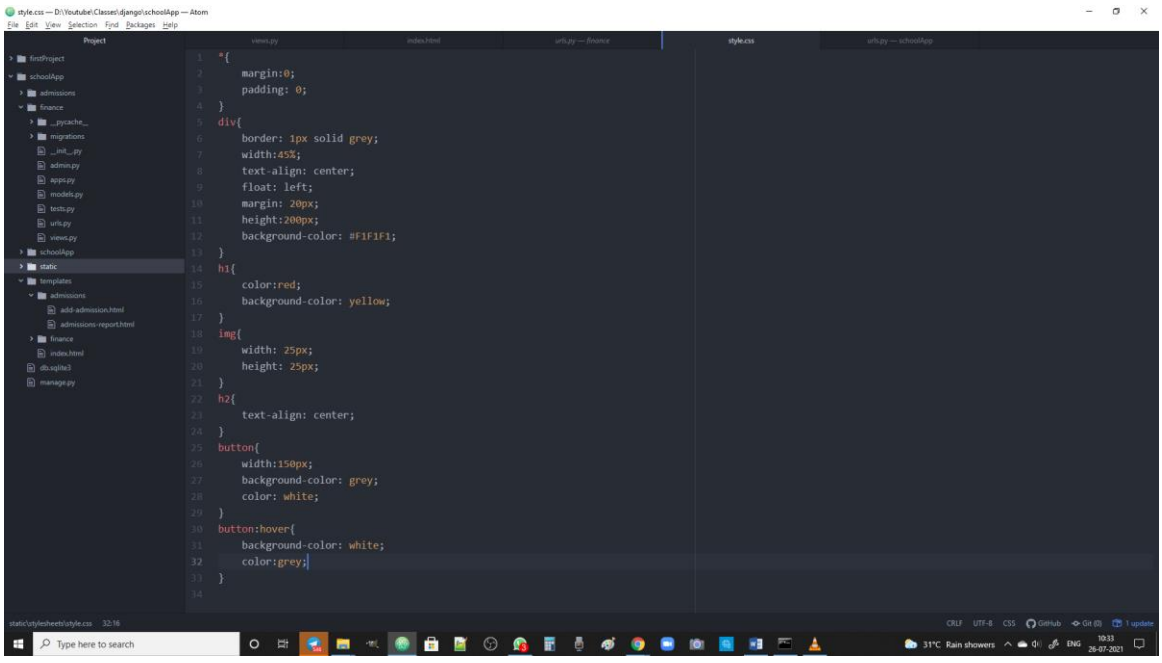
Now let us create a home page template with different navigations and set it as homepage

First let us create index.html template with all the available links as buttons.



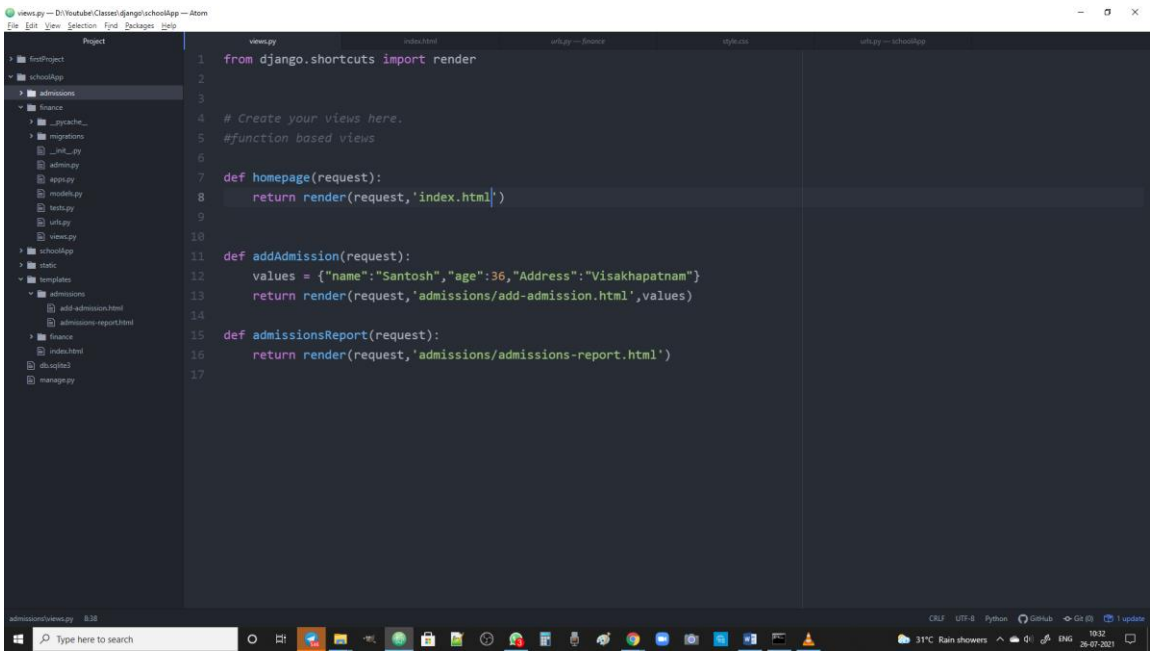
```
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>homepage</title>
        <link rel="stylesheet" href="{% static "stylesheets/style.css" %}" />
    </head>
    <body>
        <br/>
        <h2>
            <img src="{% static "images/small_logo.png" %}"/>
            Telugu Web Guru Academy
        </h2><br/>
        <br/><hr/><br/><br/><br/>
        <div class="module">
            <h1>Admissions</h1><br/>
            <a href="/ad/newadm"><button type="button" name="button">Add Admission</button></a><br/><br/>
            <a href="/ad/admreport"><button type="button" name="button">Admission Report</button></a><br/><br/>
        </div>

        <div class="module">
            <h1>Finance</h1><br/>
            <a href="/fin/feecoll"><button type="button" name="button">Fee Collection</button></a><br/><br/>
            <a href="/fin/duesreport"><button type="button" name="button">Fee Collection Report</button></a><br/><br/>
            <a href="/fin/collectionsreport"><button type="button" name="button">Fee Dues Report</button></a><br/><br/>
        </div>
    </body>
</html>
```

and write various styles to present the above web page beautiful way.



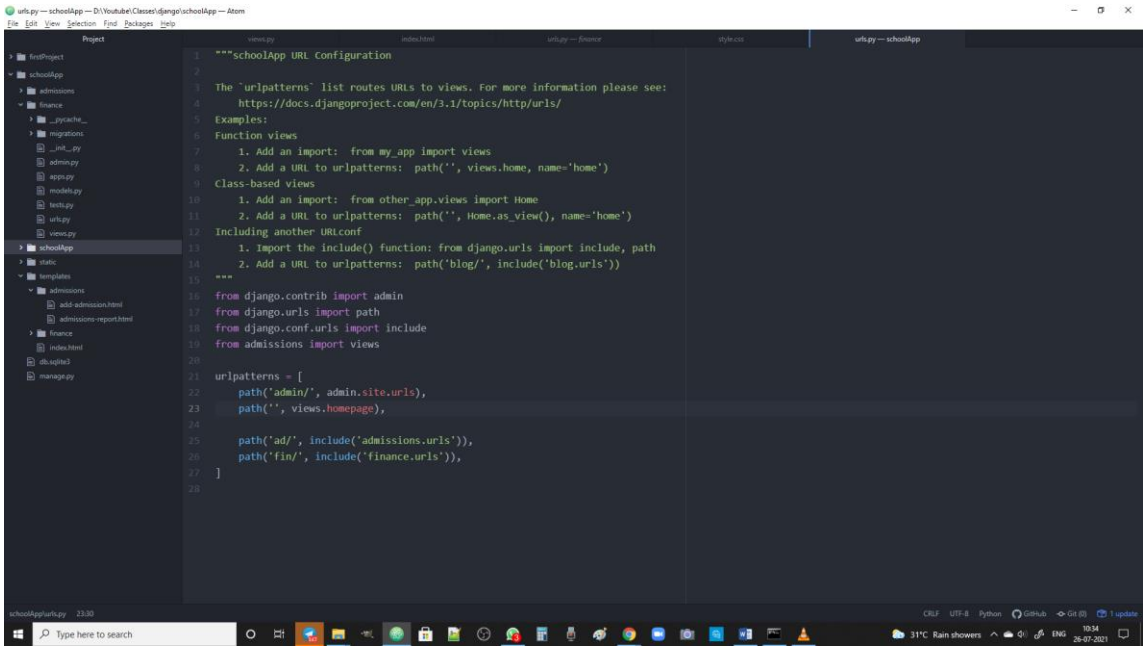
```
*{
    margin:0;
    padding: 0;
}
div{
    border: 1px solid grey;
    width:45%;
    text-align: center;
    float: left;
    margin: 20px;
    height:200px;
    background-color: #F1F1F1;
}
h1{
    color:red;
    background-color: yellow;
}
img{
    width: 25px;
    height: 25px;
}
h2{
    text-align: center;
}
button{
    width:150px;
    background-color: grey;
    color: white;
}
button:hover{
    background-color: white;
    color:grey;
}
```

Now let us create a view and link above template index.html with this view.



```python
from django.shortcuts import render


# Create your views here.
#function based views

def homepage(request):
    return render(request,'index.html')


def addAdmission(request):
    values = {"name":"Santosh","age":36,"Address":"Visakhapatnam"}
    return render(request,'admissions/add-admission.html',values)

def admissionsReport(request):
    return render(request,'admissions/admissions-report.html')
```

Let us map the view homepage with an url in urls.py so that it will be set as homepage.



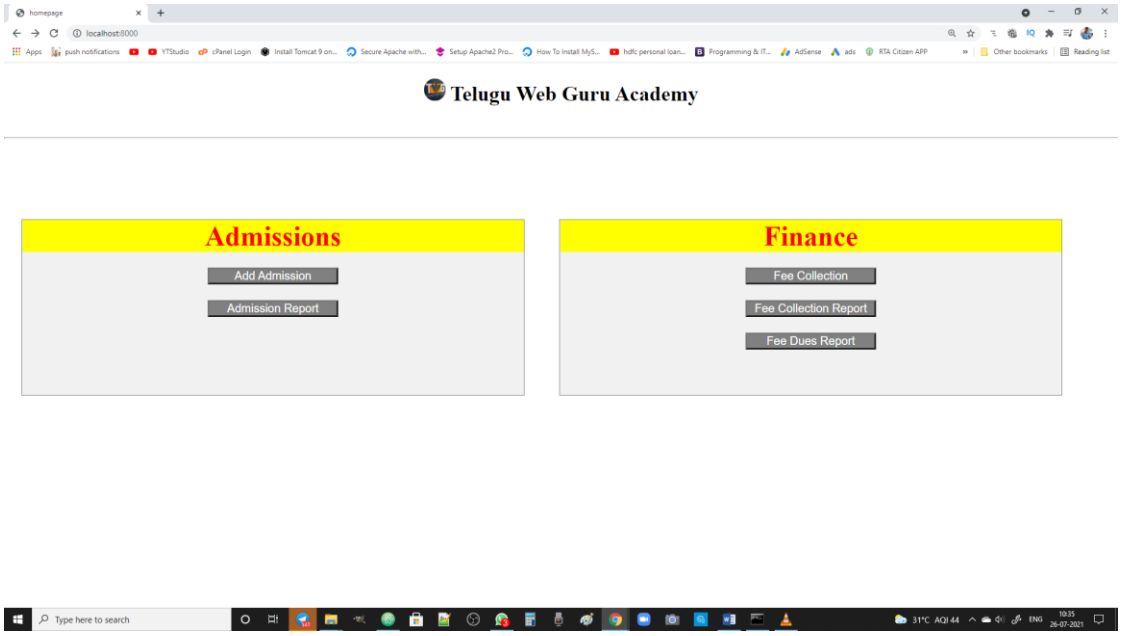
```python
"""schoolApp URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from django.conf.urls import include
from admissions import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.homepage),

    path('ad/', include('admissions.urls')),
    path('fin/', include('finance.urls')),
]
```

Now everything set.

Let us run the server and check whether the above page is set as homepage or not

```
D:\Youtube\Classes\django\schoolApp>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s
): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
July 26, 2021 - 10:35:24
Django version 3.1, using settings 'schoolApp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

and the output is:



INTRODUCTION TO MODELS

Model represents the data base tables.

So, for each table in the database we will create a separate model with table fields are declared as variables so that django will automatically create tables and prepares all the queries related to that table.

To create a model, we need to create a class that extends django.db.models.Model

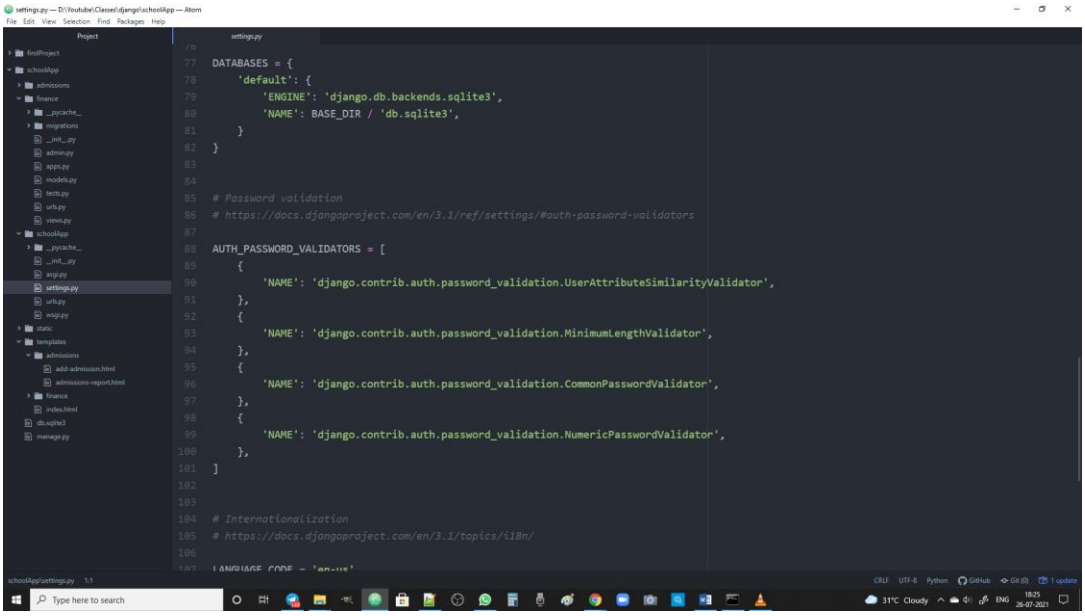
once a model is created, we can create the sql code by using makemigrations

create the tables automatically by using migrate

Along with our tables these migrations will create other tables related to administration, security, session management,

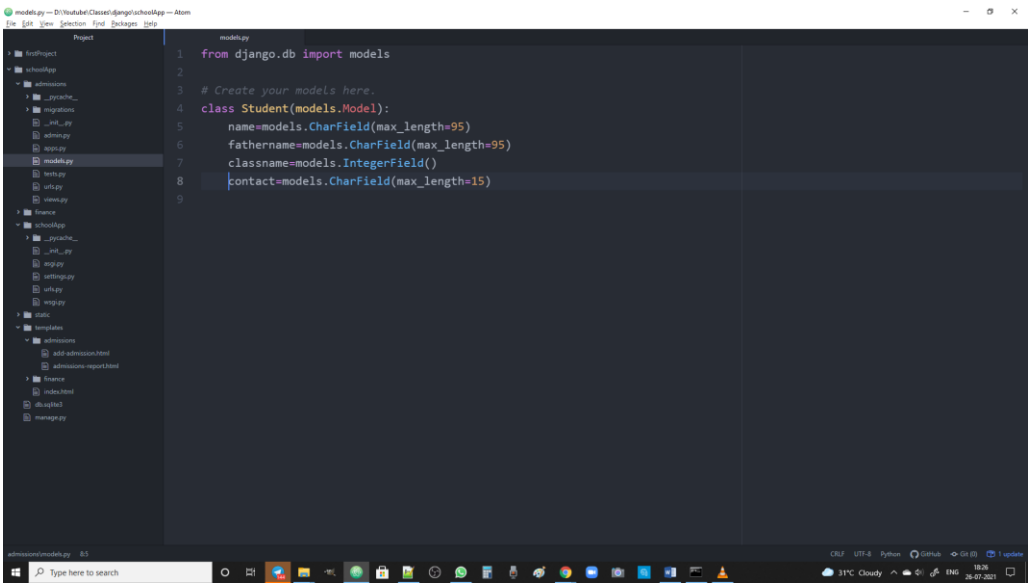
How to create a Model?

1) Configure database in settings.py



we are using default database sqlite which came with django installation.

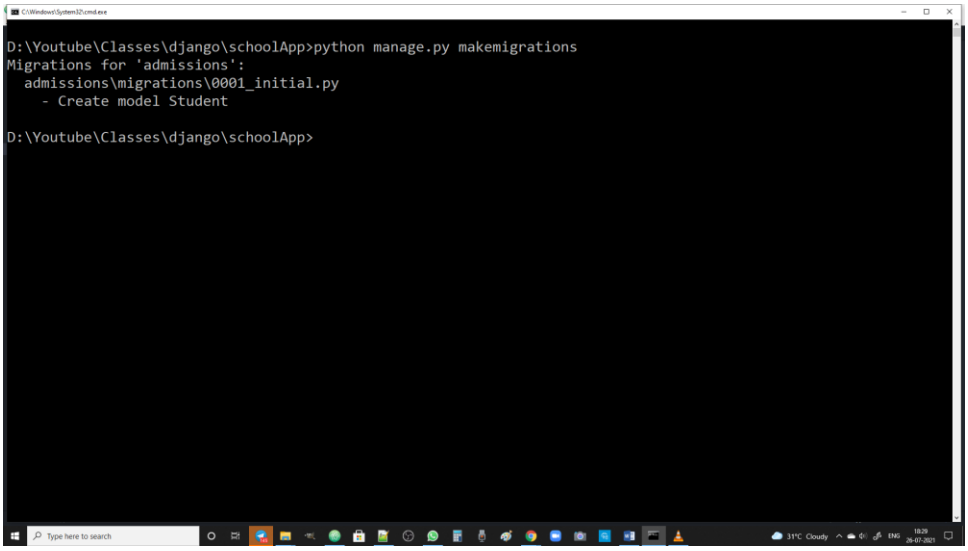
2) To create a model, open models.py file in our application folder
3) Each model is a Python class that subclasses django.db.models.Model.
4) Declare each attribute of the model represents a database field.



5) run the following command in command prompt (you must be in project folder)

python manage.py makemigrations

this will prepare all the required sql queries from the model defined.



6) run the following command to execute the created queries

python manage.py migrate

```
D:\Youtube\Classes\django\schoolApp>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, admissions, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying admissions.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

D:\Youtube\Classes\django\schoolApp>
```

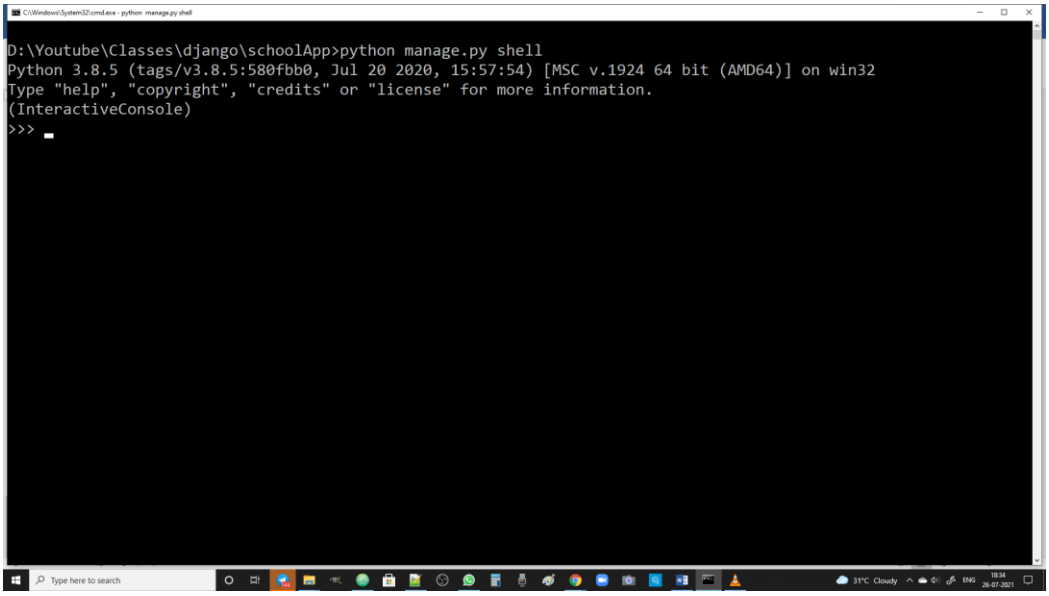
We can check how queries are prepared in the backend by django as follows

```
D:\Youtube\Classes\django\schoolApp>python manage.py sqlmigrate admissions 0001
BEGIN;
--
-- Create model Student
--
CREATE TABLE "admissions_student" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(95) NOT NULL
, "fathername" varchar(95) NOT NULL, "classname" integer NOT NULL, "contact" varchar(15) NOT NULL);
COMMIT;

D:\Youtube\Classes\django\schoolApp>
```

Now let us open sqlite and check how database and tables are created.

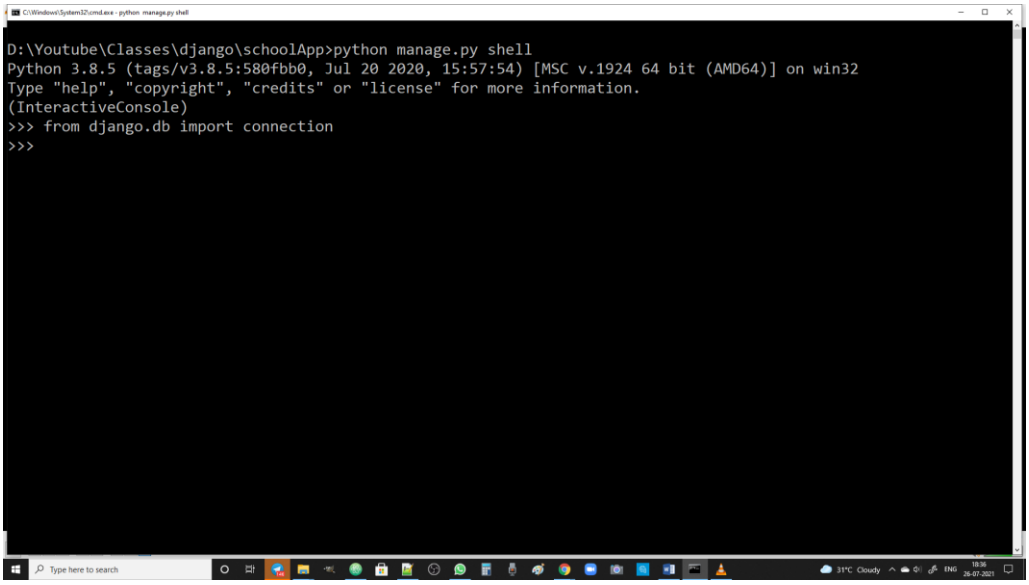
We can enter into sqlite by using following command

```
python manage.py shell
```

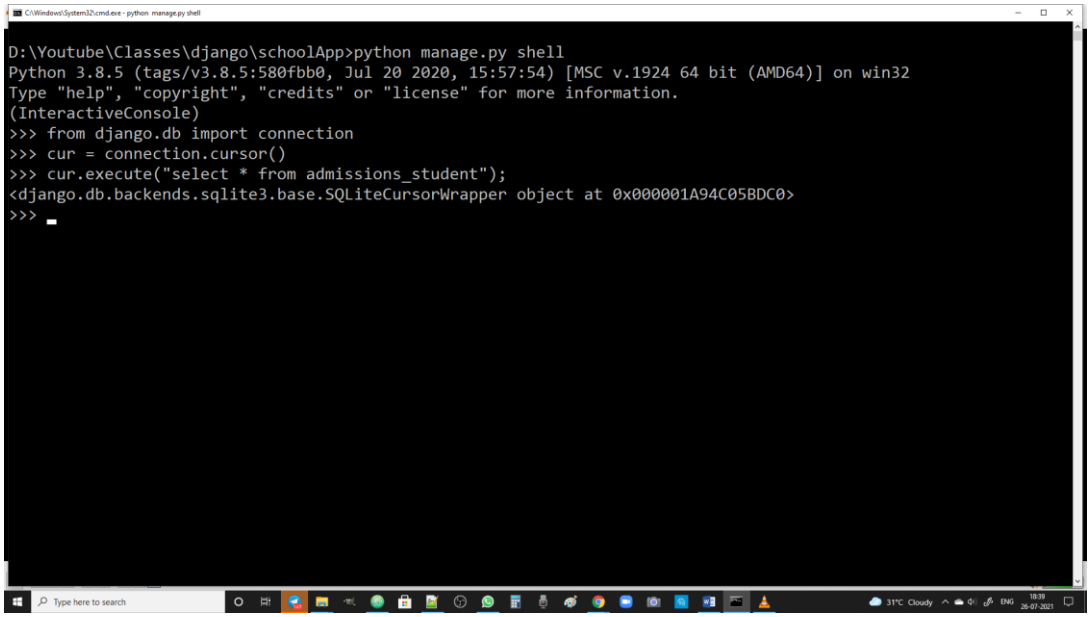


We can create a connection to the database by using following command

As we opened this command from schoolApp and in schoolApp settings we set the database settings for sqlite, so this command will open a connection with sqlite database.



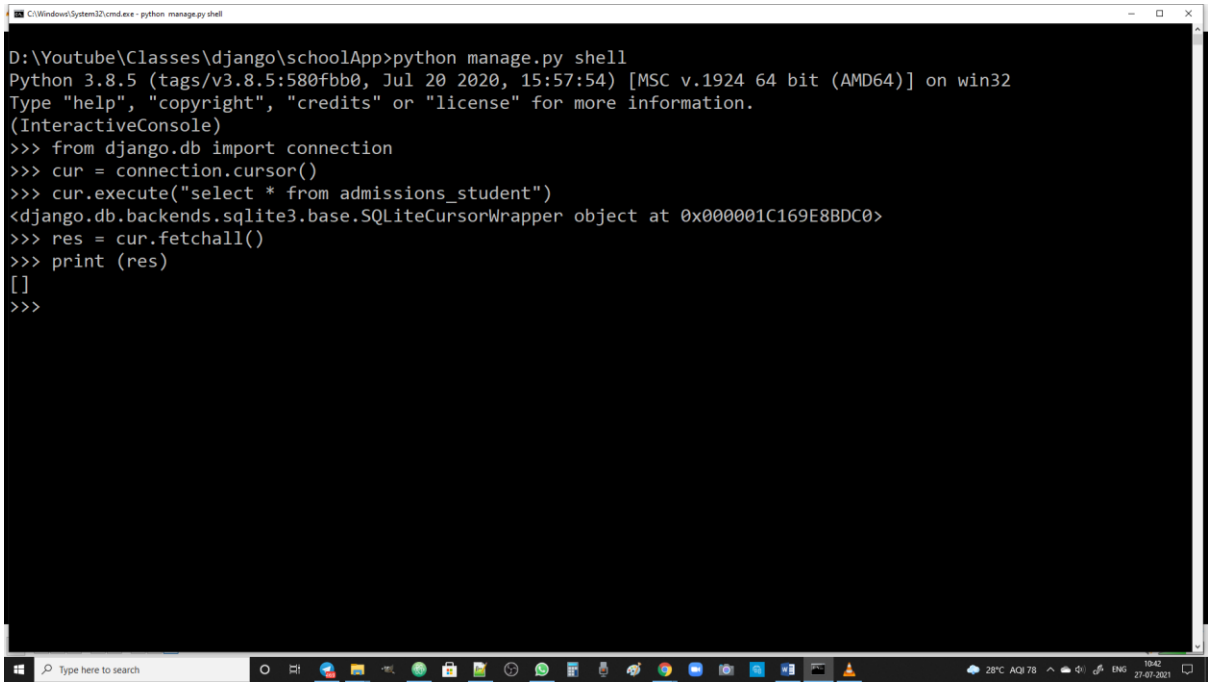
Now as we learned in python database connections topic, we need to open cursor, and execute the required query by using execute() method.



observe here that django creates tables as applicationname_modelname

that's why we tried to retieve the rows from the table admissions_student

we can see the result by using fetch operations.



As students are not yet added, It returned empty list.

USING MODELS IN VIEWS

Now let us see how to connect our model Student with the view so that we can do database operations and display the results in template.

Steps:

1) **import the model Student**

```
from admissions.models import Student
```

2) **get all the records from database using model**

```
result = Student.objects.all()
```

3) **create a dictionary and store above records in this dictionary**

```
students = {'allstudents' : result}
```

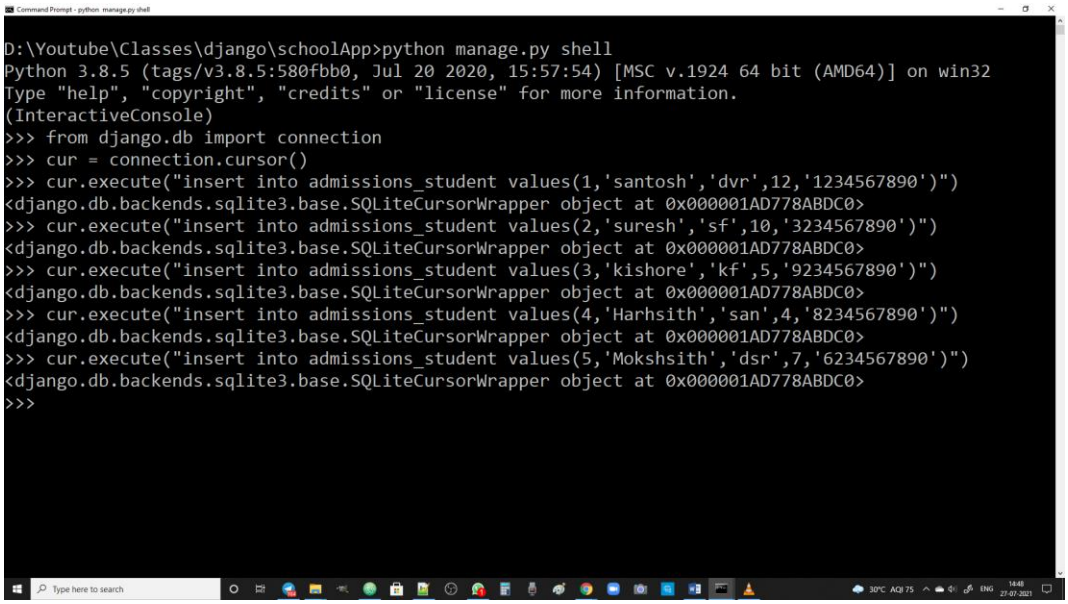
4) **pass the dictionary to template by using render method**

```
return render(request,'admissions/admissions-report.html' , students)
```

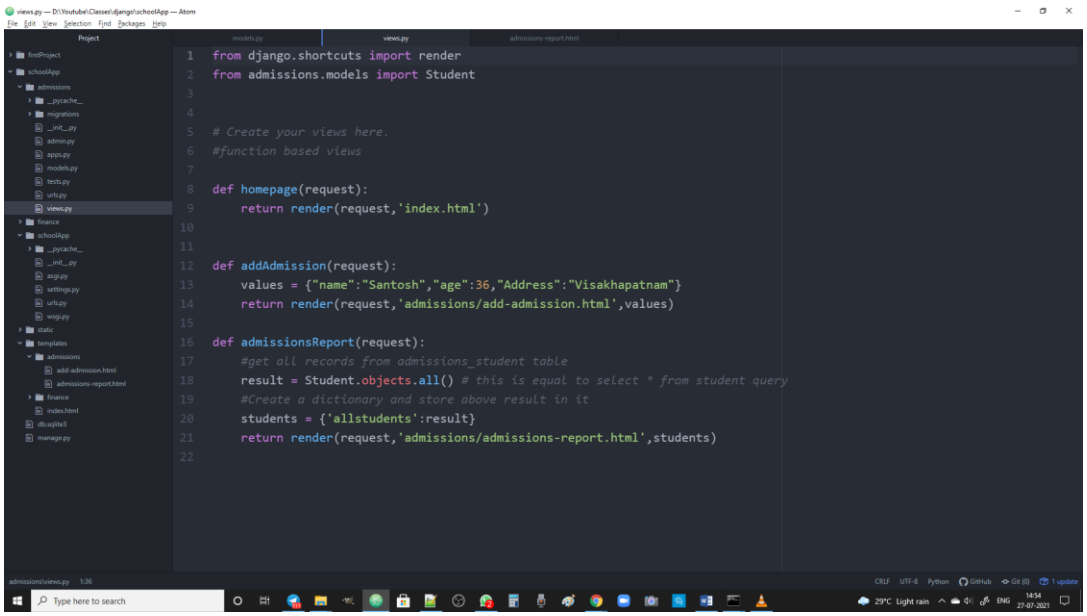
5) **Retrieve the values and display them in the template by using a for loop as follows**

```
{% for s in allstudents %}
    {{s.name}} -  {{s.fathername}}
     <br/>
{% endfor %}
```

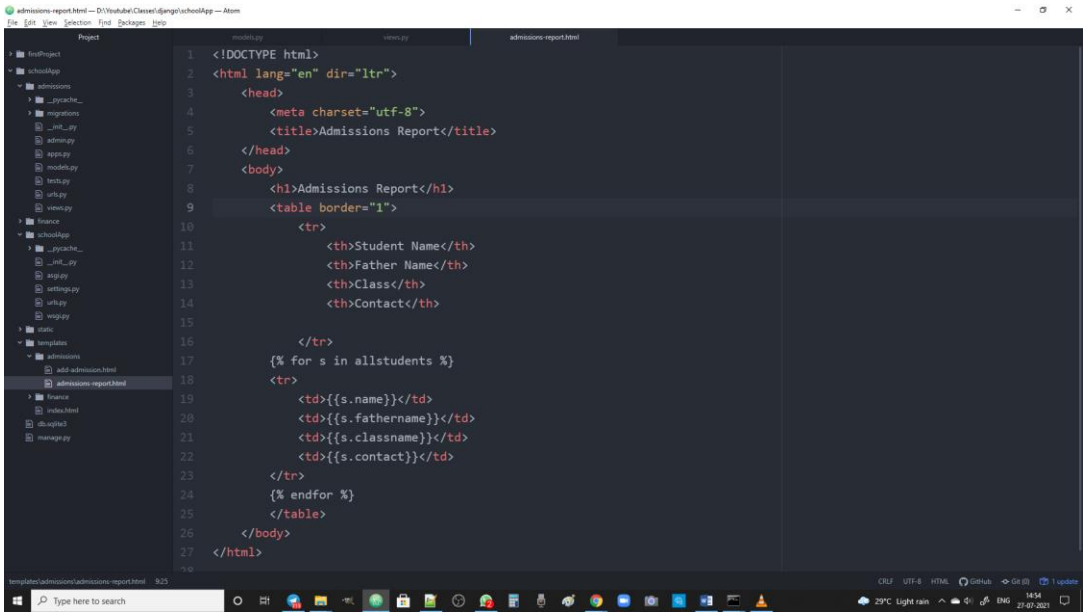
Now let us manually insert some records into the table



full code of views.py and template are as follows

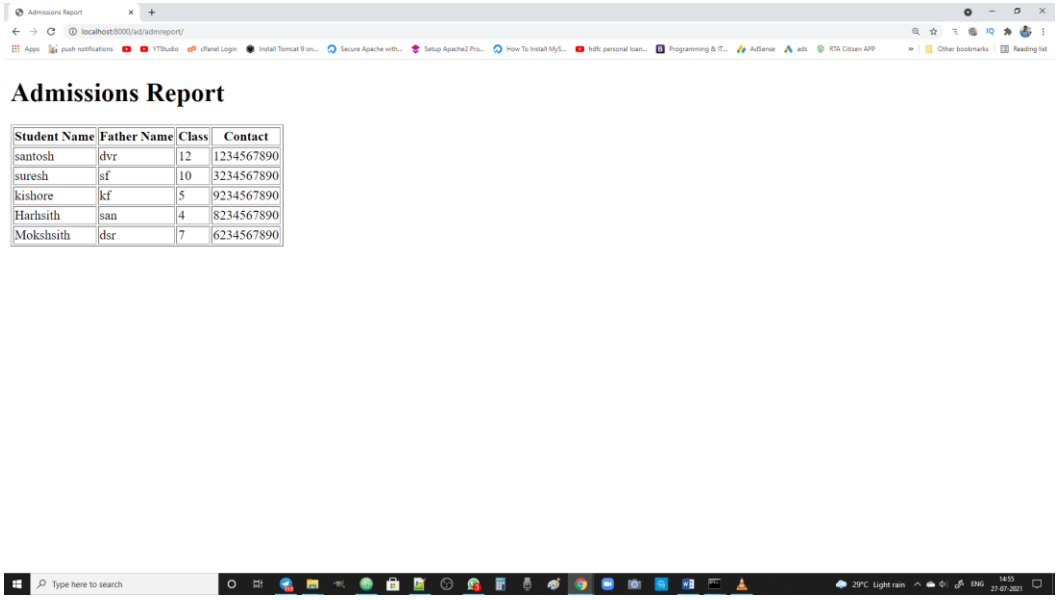


```python
from django.shortcuts import render
from admissions.models import Student


# Create your views here.
#function based views

def homepage(request):
    return render(request,'index.html')


def addAdmission(request):
    values = {"name":"Santosh","age":36,"Address":"Visakhapatnam"}
    return render(request,'admissions/add-admission.html',values)

def admissionsReport(request):
    #get all records from admissions_student table
    result = Student.objects.all() # this is equal to select * from student query
    #Create a dictionary and store above result in it
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students)

```



```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Admissions Report</title>
    </head>
    <body>
        <h1>Admissions Report</h1>
        <table border="1">
            <tr>
                <th>Student Name</th>
                <th>Father Name</th>
                <th>Class</th>
                <th>Contact</th>

            </tr>
        {% for s in allstudents %}
        <tr>
            <td>{{s.name}}</td>
            <td>{{s.fathername}}</td>
            <td>{{s.classname}}</td>
            <td>{{s.contact}}</td>
        </tr>
        {% endfor %}
        </table>
    </body>
</html>
```

Now let us start the server and run the program



Admissions Report

Student Name	Father Name	Class	Contact
santosh	dvr	12	1234567890
suresh	sf	10	3234567890
kishore	kf	5	9234567890
Harhsith	san	4	8234567890
Mokshsith	dsr	7	6234567890

This is how we can link models with the views and retrieve the records and display those results

DJANGO – MYSQL CONNECTIVITY

The First step that we need to do in order to implement django-mysql connectivity is setting the mysql values in settings.py

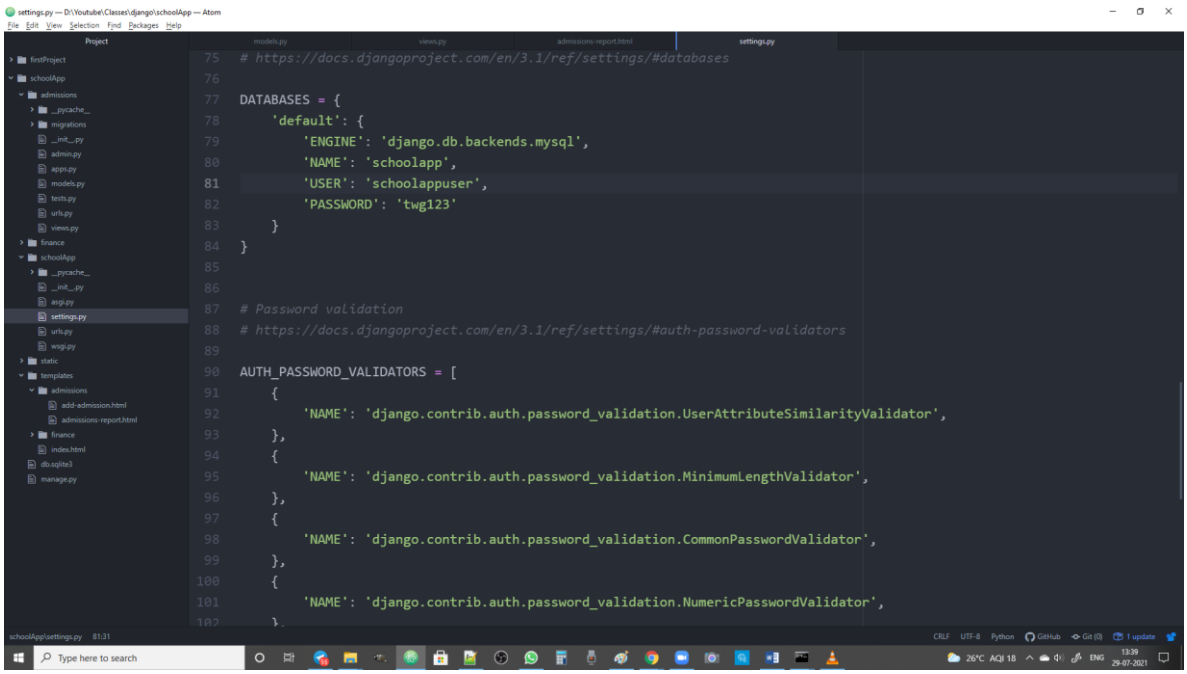
Before setting the values let us create a user in mysql with mysql_native_password option as follows.

```
mysql> create database schoolapp;
Query OK, 1 row affected (0.01 sec)

mysql> create user 'schoolappuser'@'localhost' identified with mysql_native_password by 'twg123';
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Now grant the permission to the user schoolappuser to access schoolapp database.

```
mysql> grant all on schoolapp.* to 'schoolappuser'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Now let's set the database setting in settings.py



```
# https://docs.djangoproject.com/en/3.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'schoolapp',
        'USER': 'schoolappuser',
        'PASSWORD': 'twg123'
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
```

Now let us run makemigrations and migrate commands to create the tables in mysql instead of sqlite.

python manage.py makemigrations

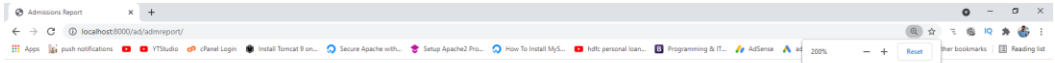
python manage.py migrate

```
mysql> use schoolapp;
Database changed
mysql> show tables;
+----------------------------+
| Tables_in_schoolapp        |
+----------------------------+
| admissions_student         |
| admissions_teacher         |
| 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             |
+----------------------------+
12 rows in set (0.00 sec)

mysql> desc admissions_student;
+------------+---------------+------+-----+---------+----------------+
| Field      | Type          | Null | Key | Default | Extra          |
+------------+---------------+------+-----+---------+----------------+
| id         | int           | NO   | PRI | NULL    | auto_increment |
| name       | varchar(1000) | NO   |     | NULL    |                |
| fathername | varchar(1000) | NO   |     | NULL    |                |
| classname  | int           | NO   |     | NULL    |                |
| contact    | varchar(1000) | NO   |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

Now let us insert some student data into the above table manually

```
mysql> desc admissions_student;
+------------+---------------+------+-----+---------+----------------+
| Field      | Type          | Null | Key | Default | Extra          |
+------------+---------------+------+-----+---------+----------------+
| id         | int           | NO   | PRI | NULL    | auto_increment |
| name       | varchar(1000) | NO   |     | NULL    |                |
| fathername | varchar(1000) | NO   |     | NULL    |                |
| classname  | int           | NO   |     | NULL    |                |
| contact    | varchar(1000) | NO   |     | NULL    |                |
+------------+---------------+------+-----+---------+----------------+
5 rows in set (0.04 sec)

mysql> insert into admissions_student values(1,'Santosh','dvr',4,'9999999999');
Query OK, 1 row affected (0.01 sec)

mysql> insert into admissions_student values(2,'Harshith','dsr',3,'8888899999');
Query OK, 1 row affected (0.01 sec)

mysql> insert into admissions_student values(3,'Mokshith','dsr',1,'7777799999');
Query OK, 1 row affected (0.01 sec)

mysql> select * from admissions_student;
+----+----------+------------+-----------+------------+
| id | name     | fathername | classname | contact    |
+----+----------+------------+-----------+------------+
|  1 | Santosh  | dvr        |         4 | 9999999999 |
|  2 | Harshith | dsr        |         3 | 8888899999 |
|  3 | Mokshith | dsr        |         1 | 7777799999 |
+----+----------+------------+-----------+------------+
3 rows in set (0.00 sec)

mysql>
```

Now let us run the program and check whether our django application can retrieve the data from the mysql table admissions_student or not



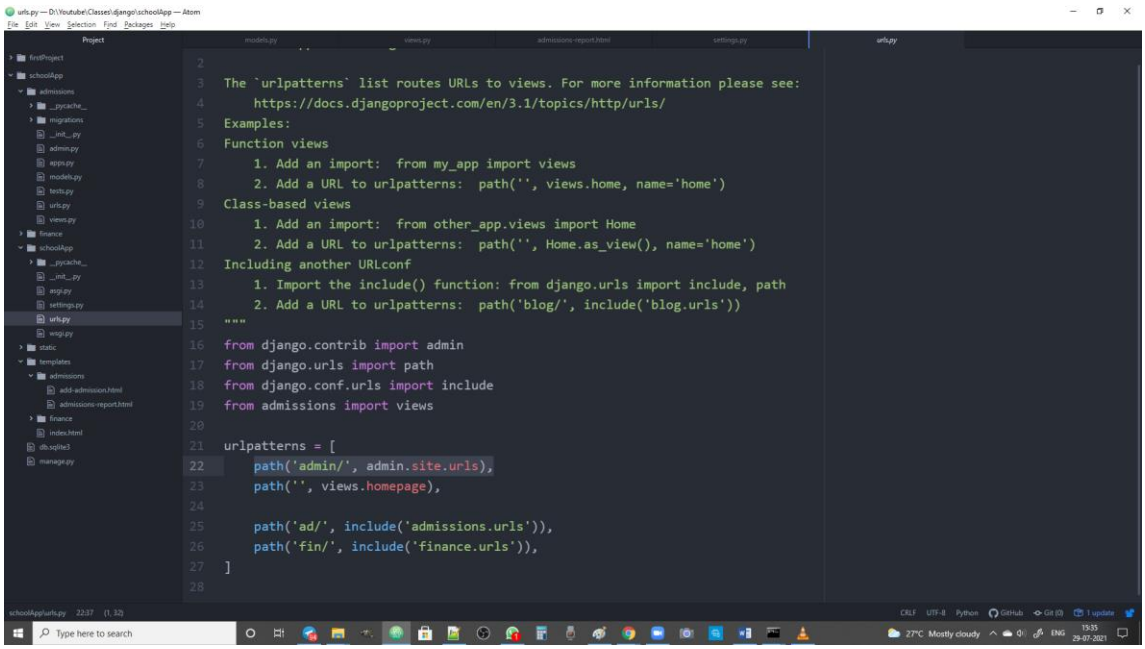
Admissions Report

Student Name	Father Name	Class	Contact
Santosh	dvr	4	9999999999
Harshith	dsr	3	8888899999
Mokshith	dsr	1	7777799999

DJANGO – ADMIN USER INTERFACE

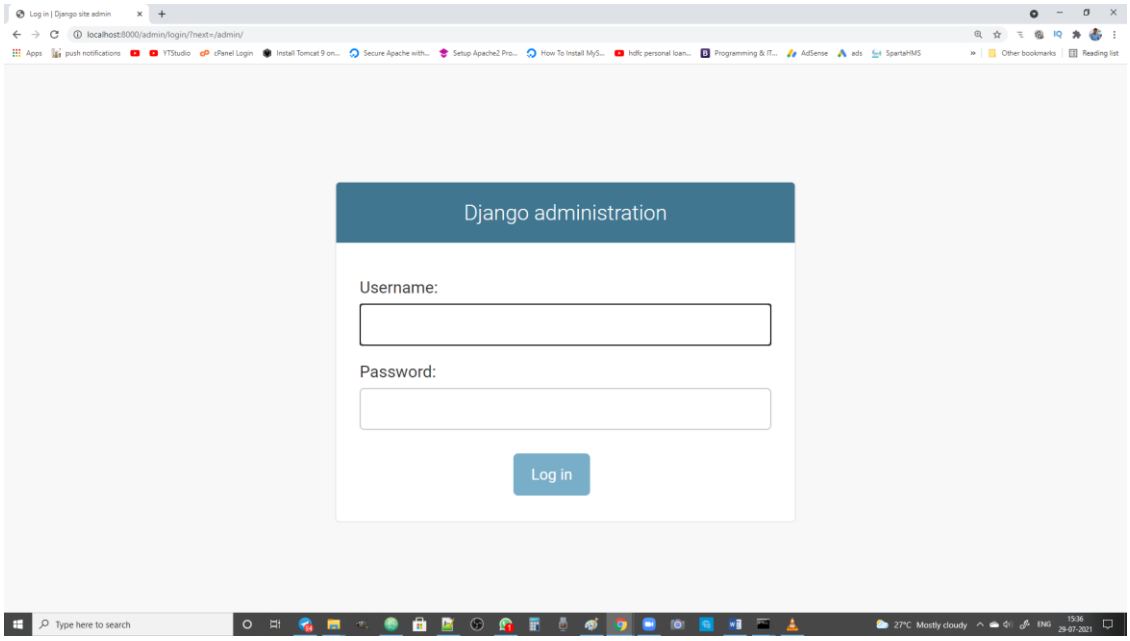
Django provides Administrator user interface for projects automatically.

In urls.py one url mapping is by default provided by django i.e., admin/



```
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.1/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from django.conf.urls import include
from admissions import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.homepage),

    path('ad/', include('admissions.urls')),
    path('fin/', include('finance.urls')),
]
```

If we run this url we will get administrator login page automatically.



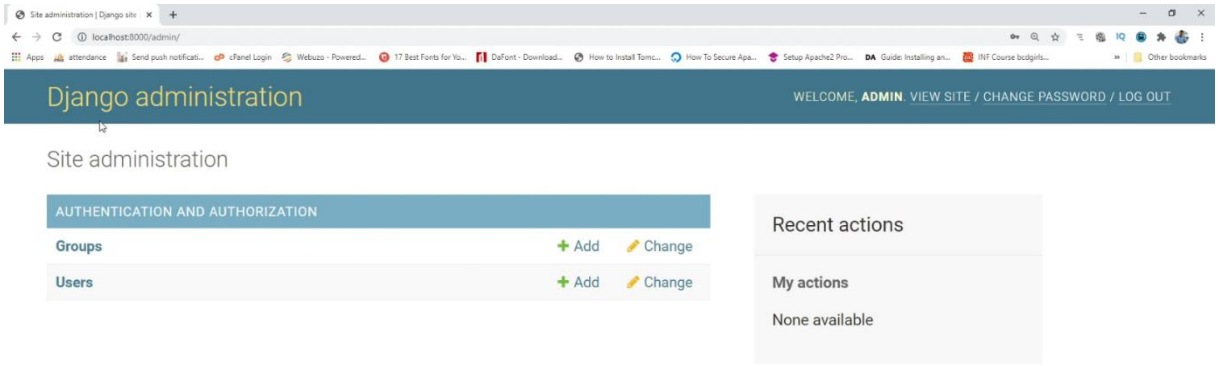
To login into this, first we need to create a super user so that we can use those credentials.

C:\WINDOWS\system32\cmd.exe - python manage.py createsuperuser

```
D:\Youtube\Classes\django\schoolApp>python manage.py createsuperuser
Username (leave blank to use 'santosh'): admin
Email address: teluguwebguru@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]:
```

Now we successfully created a superuser with the login details admin,admin

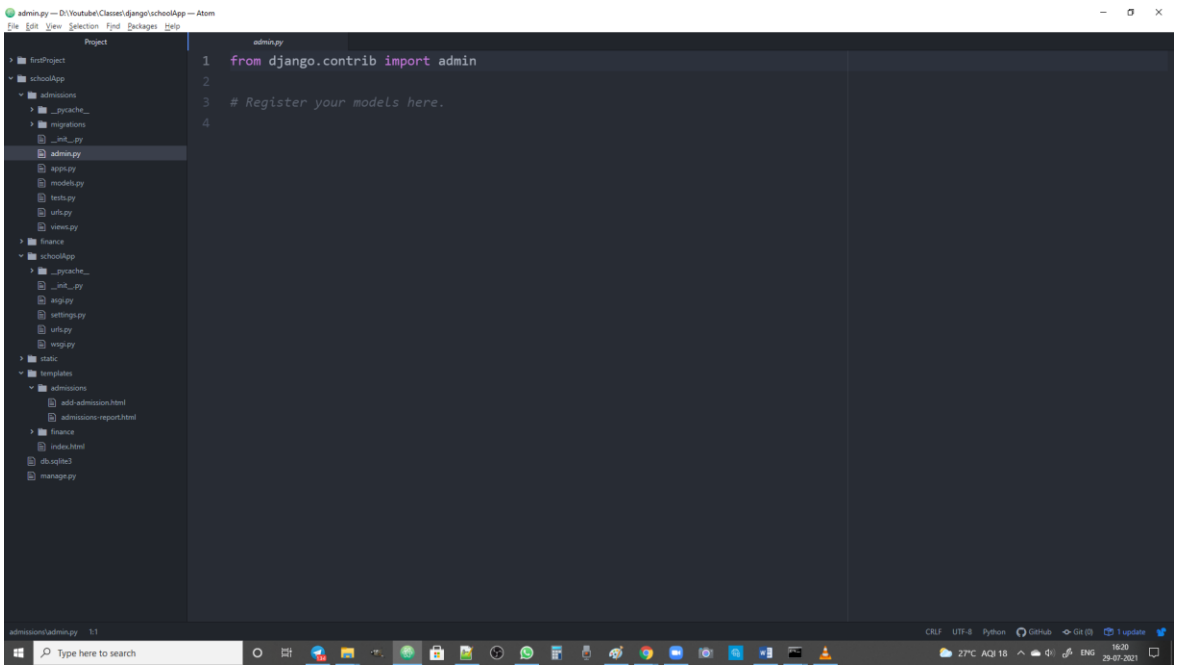
Now let us try to enter into the admin ui by using these credentials

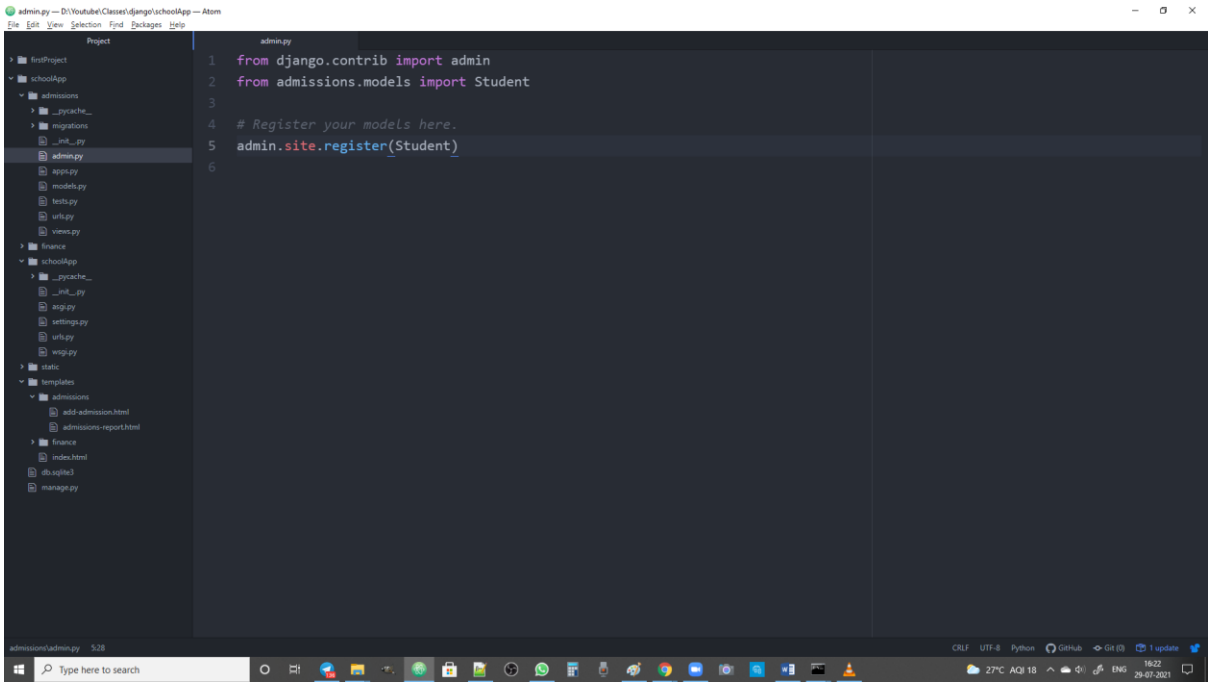


Observe how the admin user interface is automatically created by django.

Now let us add Student model to the admin UI

open admin.py in admission application

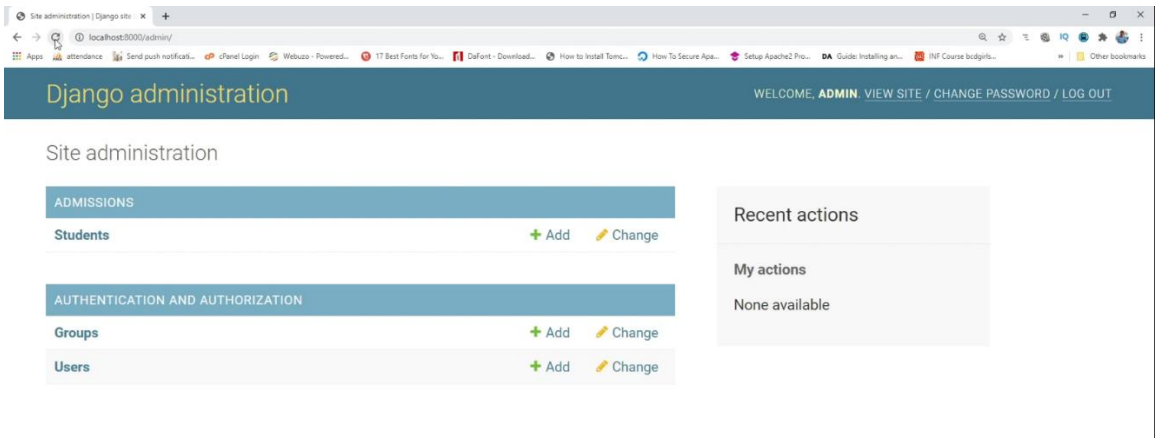




As shown above, We need to first import the model and register it into admin ui using

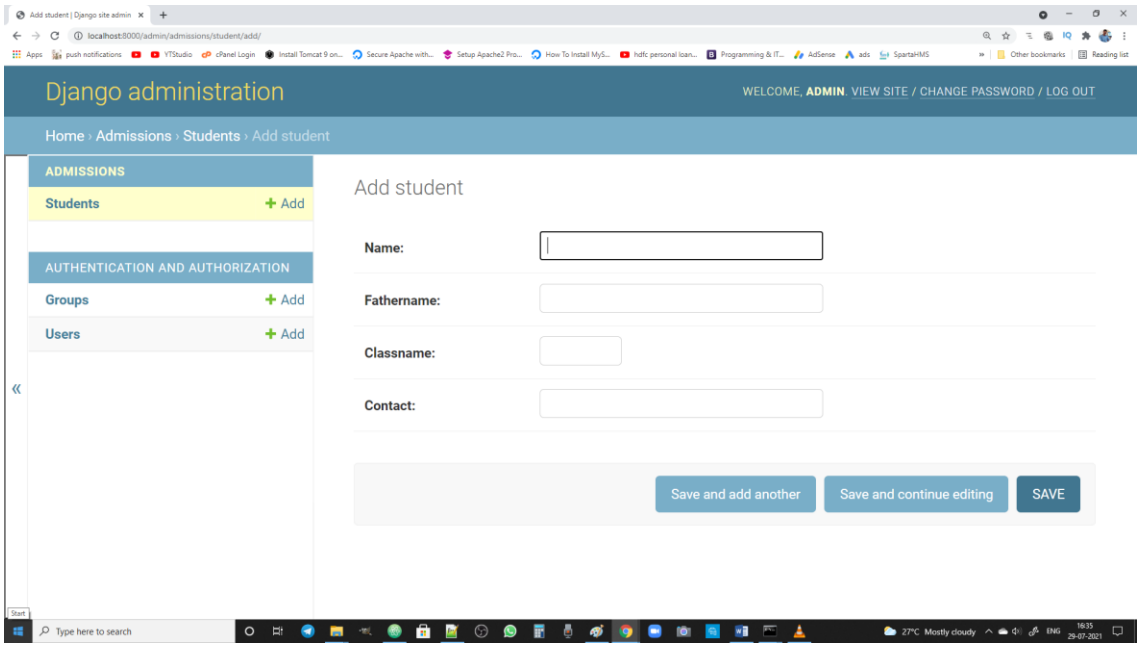
admin.site.register(modelname)

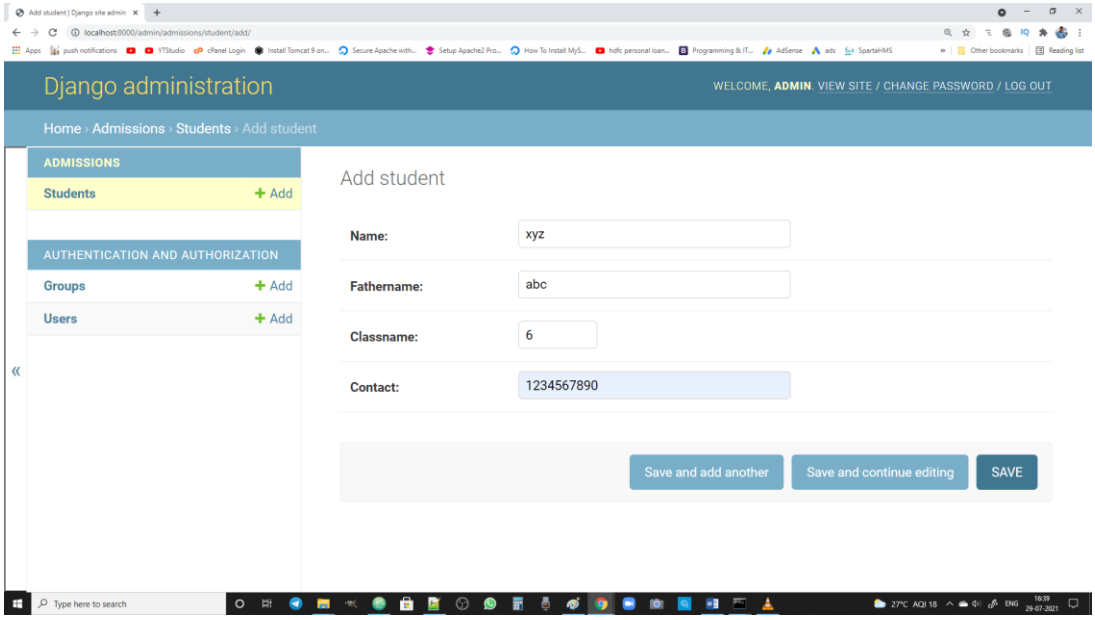
After this process, if we run our admin page again we can see the Student model in admin User interface.



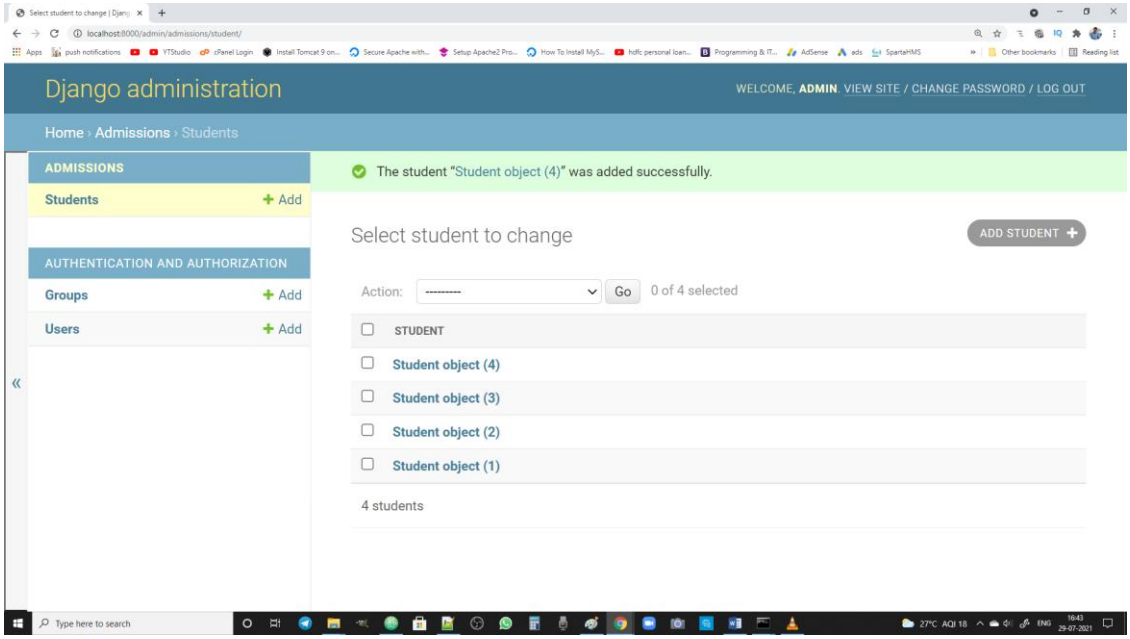
Observe, the registered model is displayed in admin ui interface

If we click on add button it will automatically creates a form and display it to enter new students data.



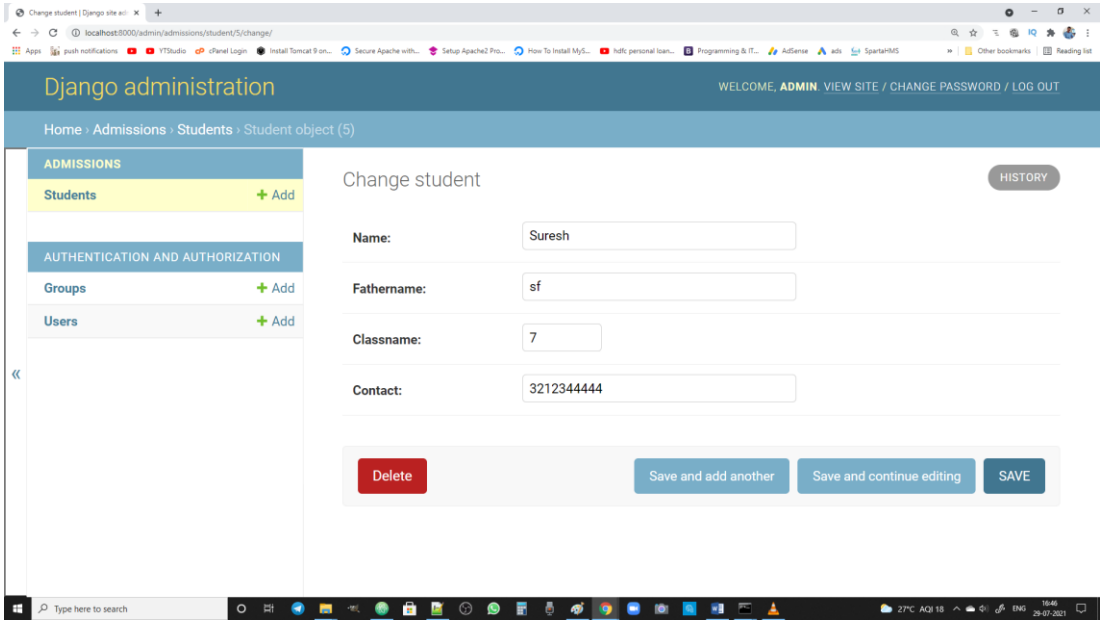


Once we fill data and click on save, it will automatically add it to the database

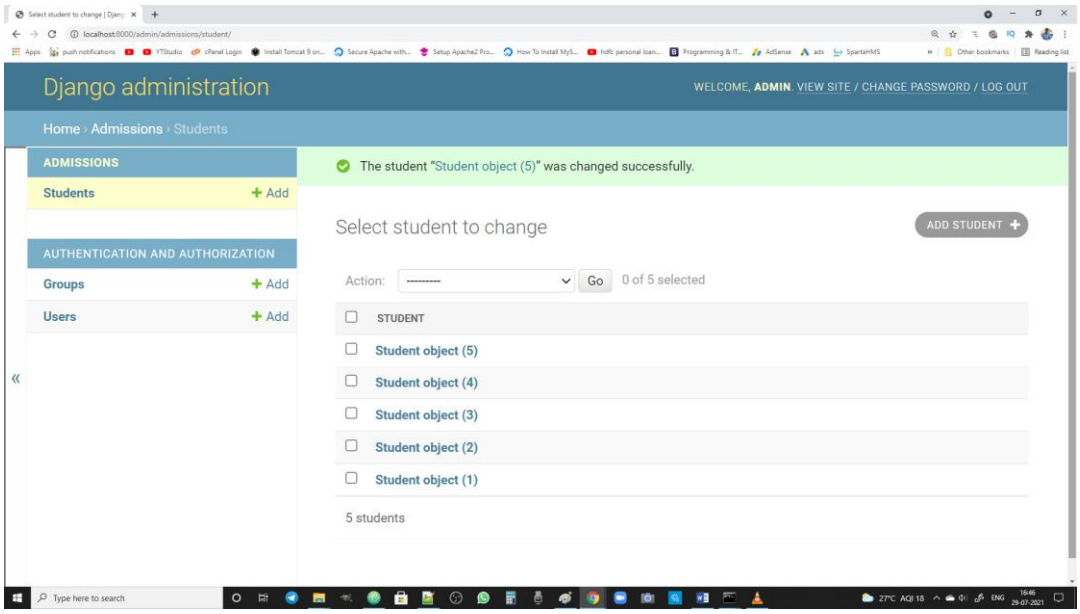


By default, it will display each student as an object. We added three students in above topic and now we added the forth student xyz, so it is showing 4 students.

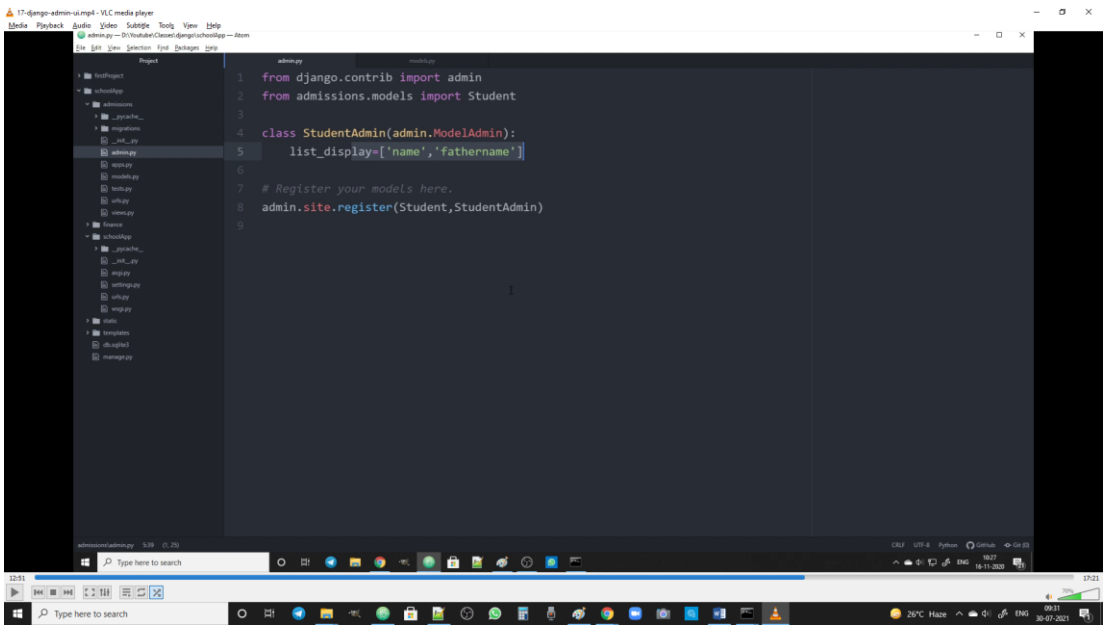
Let us try to add one more student now.



after click on save, it will now show 5 student objects

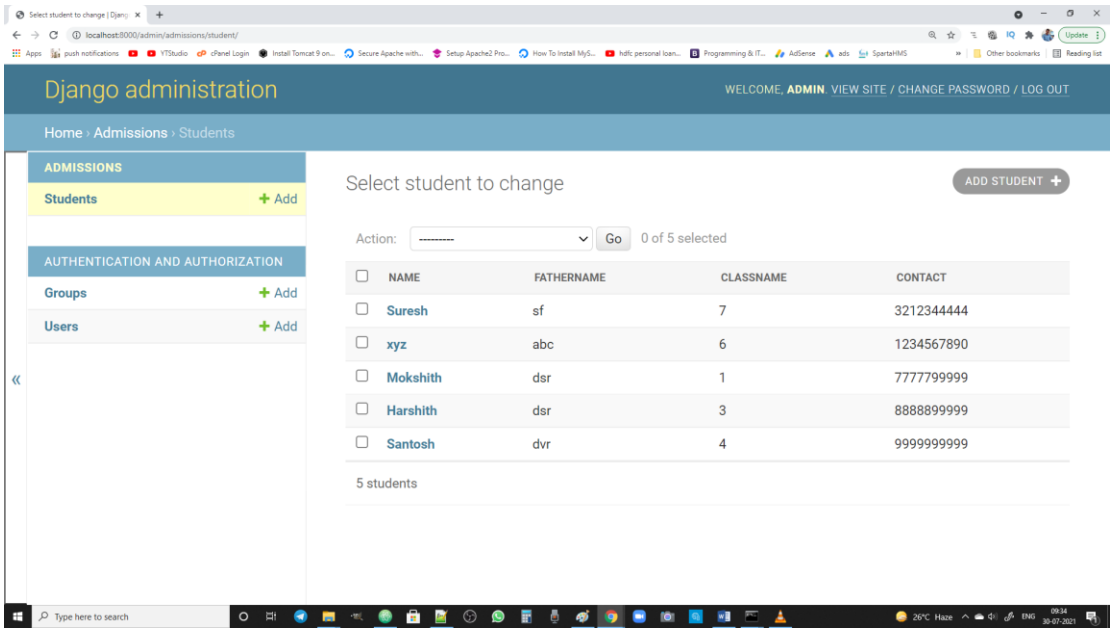


If we want to display these student objects as tables then we need to define an extra class in admin.py file (instead of registering the model directly).



Instead of directly registering our model with the admin, we need to create a class that inherits from ModelAdmin and we need to specify all the list of columns to be displayed in the admin ui by using list_display as shown above.

Now let us run the admin ui and check whether displaying student objects are replaced with these columns.



This is how django automatically provides a very beautiful admin interface.

DJANGO – ORM

The following are the various ways to implement various SQL queries through Django ORM concept. Following tables shows what's the query we want to write, How to implement it in django ORM way.

Selecting all columns

select * from admissions_student	qs = Student.objects.all() print (qs.query) for s in qs: print (s.name)
select * from admissions_student where id=2	s = Student.objects.get(id=2) print (s.name)

Selective columns(values_list or values or only)

select id,firstname from admissions_student	students = Student.objects.all().values_list('id', 'firstname')

filtering using (gt,gte,lt,lte,contains,icontains,in,startswith,endswith)

select * from admissions_student where id>2	students = Student.objects.filter(id__gt=2)

Logical Operators (And, Or, Not)

select * from admissions_student where id>2 or name LIKE '%sa%'	students = Student.objects.filter(id__gt=2) | Student.objects.filter(name__contains='sa')
select * from admissions_student where id>2 and id<5	students = Student.objects.filter(id__gt=2) & Student.objects.filter(id__lt=5)
select * from admissions_student where NOT(id>5)	students = Student.objects.exclude(id__gt=5)

Aggregate Functions (Avg, Sum, Max, Min, Count)

select id,firstname from admissions_student	from django.db.models import Avg,Sum,Max,Min,Count students = Student.objects.all().aggregate(Avg('id'))

inserting data into the database (create)

insert into admissions_student values(6,'abc','xyz',8,'5648562233')	1) s = Student(name='', fathername='',classname=,contact='') s.save() 2) Student.objects. create(name='', fathername='',classname=,contact='')

Bulk creating/inserting (bulk_create)

insert into admissions_student values(6,'abc','xyz',8,'5648562233')	Student.objects. bulk_create([Student(name='', fathername='',classname=,contact=''), Student(name='', fathername='',classname=,contact=''), Student(name='', fathername='',classname=,contact=''),])

Delete from tables

Delete from admissions_student where id=2	s = Student.objects.get(id=1) s.delete() qs = Student.objects.filter(id__gt5) qs.delete()

Updating tables

Update admissions_student set name='' where id=2	s = Student.objects.get(id=1) s.firstname='' s.save()

Sorting data in tables

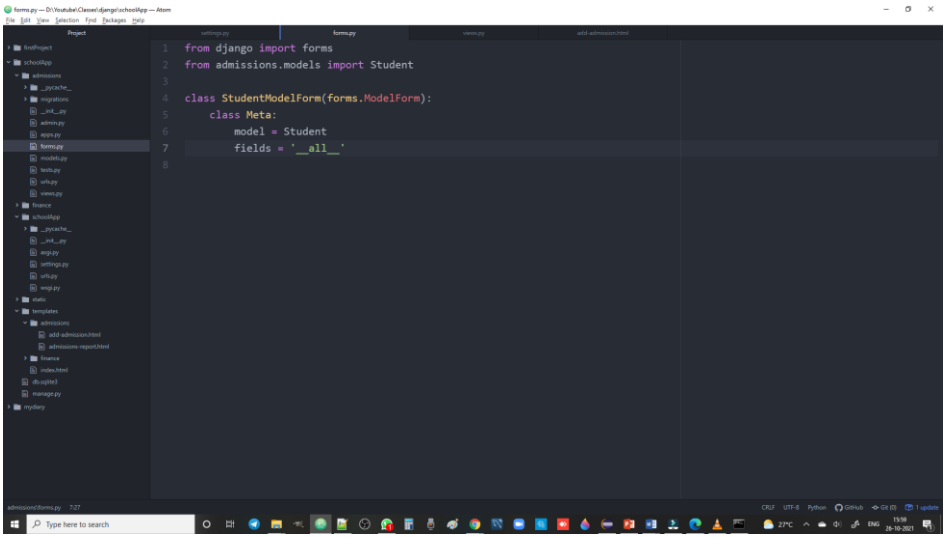
Select * from student order by firstname	s = Student.objects.all().order_by('firstname') s = Student.objects.all().order_by('-firstname') => reverse

DJANGO – MODEL FORMS

To create model forms

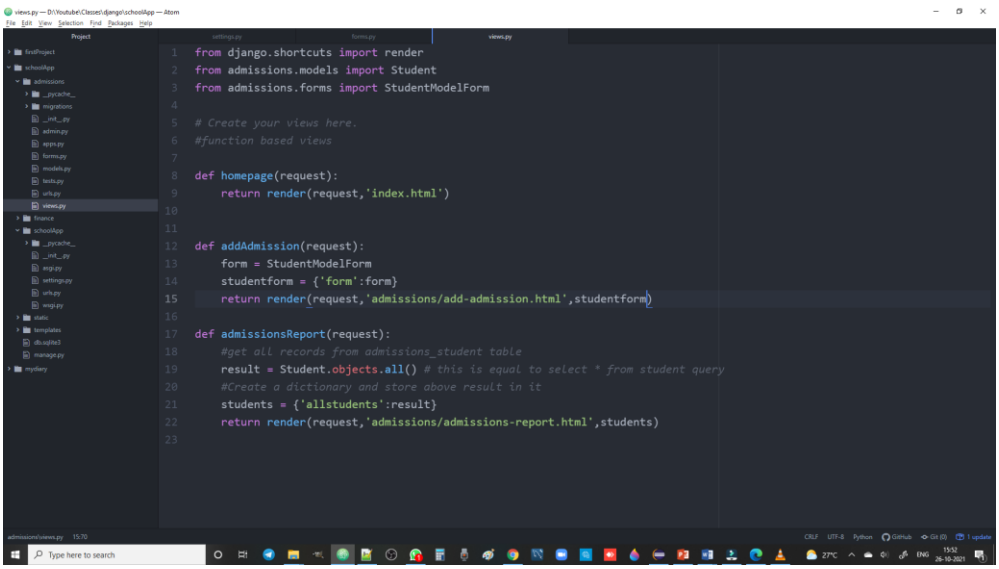
1) create a file forms.py in your application folder
2) Create a class that is inherited form forms.ModelForm
3) Create a class Meta

```
class classname(forms.ModelForm):
    class Meta:
        model = modelname
        fields = '__all__' (to import selected fields mention them as list)
```



4) In your view get the model form and send as dictionary to template (import model, model form first)

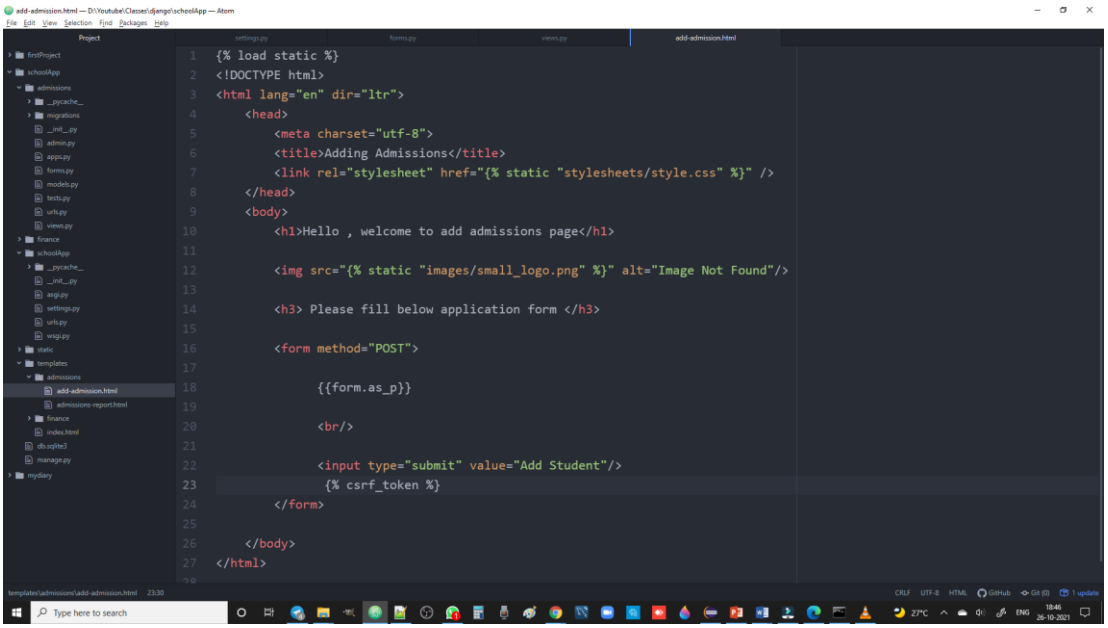
```
form  = modelformname() ---- creating object of modelform
return render (request, template name , dictionary)
```



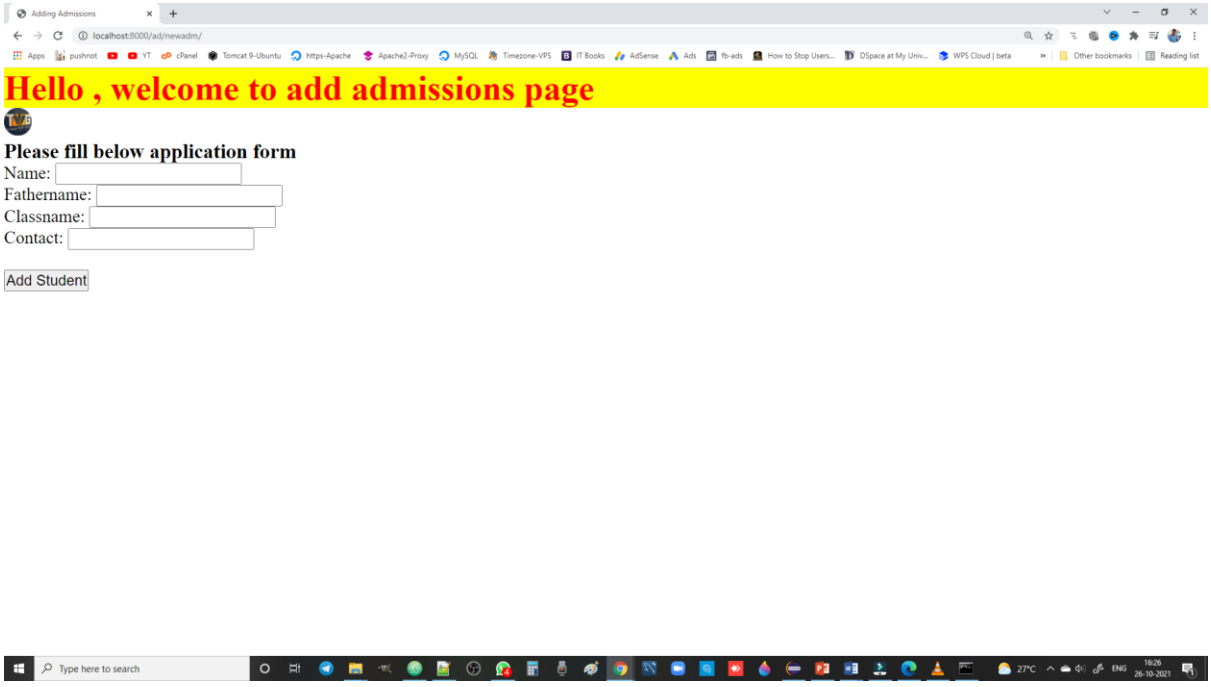
```python
from django.shortcuts import render
from admissions.models import Student
from admissions.forms import StudentModelForm

# Create your views here.
#function based views

def homepage(request):
    return render(request,'index.html')


def addAdmission(request):
    form = StudentModelForm
    studentform = {'form':form}
    return render(request,'admissions/add-admission.html',studentform)

def admissionsReport(request):
    #get all records from admissions_student table
    result = Student.objects.all() # this is equal to select * from student query
    #Create a dictionary and store above result in it
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students)
```

5) display the form in template

```
<form method="POST">
        {{form.as_p}}
        {% csrf_token %}
</form>
```



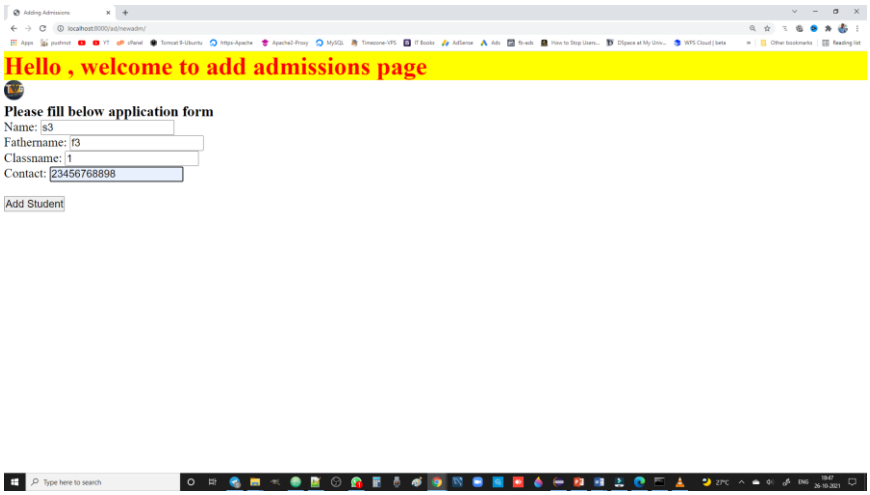
```python
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding Admissions</title>
        <link rel="stylesheet" href="{% static "stylesheets/style.css" %}" />
    </head>
    <body>
        <h1>Hello , welcome to add admissions page</h1>

        <img src="{% static "images/small_logo.png" %}" alt="Image Not Found"/>

        <h3> Please fill below application form </h3>

        <form method="POST">

            {{form.as_p}}

            <br/>

            <input type="submit" value="Add Student"/>
             {% csrf_token %}
        </form>

    </body>
</html>
```


Adding Admissions
localhost:8000/ad/newadm/
Hello , welcome to add admissions page
Please fill below application form
Name:
Fathername:
Classname:
Contact:
Add Student

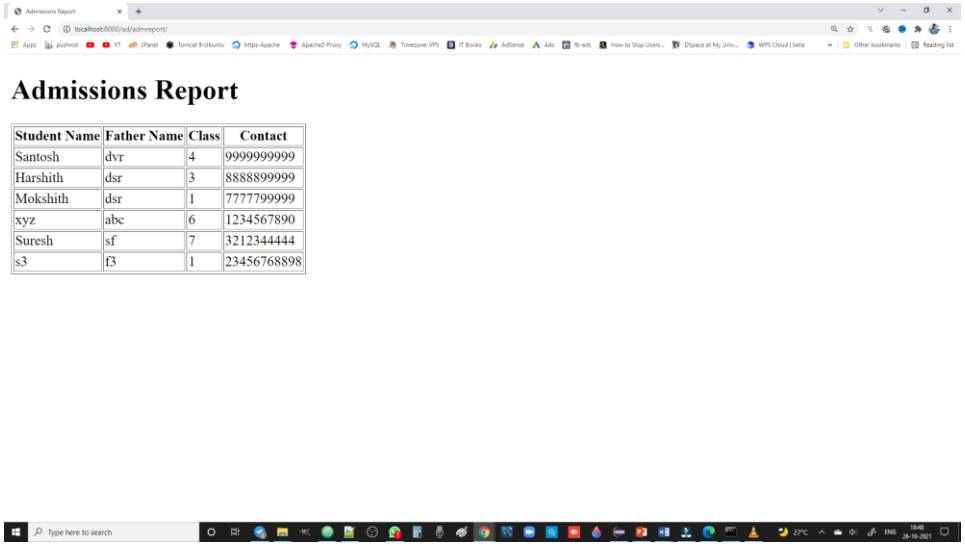
SAVING FORM DATA IN DJANGO

To get input from model forms and save in database we need to get the form data and check whether it is valid or not if valid we can save by using save method as follows.

```
form  = modelformname() ---- creating object of modelform
    if form.method=='POST':
        form = modelformname(request.POST)
        if form.is_valid():
            form.save()
        return indexpageviewname(request)
return render (request, template name ,  dictionary)
```



whenever we clicked on the button data will be saved into the database. if we check the admissions report then we can see this entry in the table.



Admissions Report

Student Name	Father Name	Class	Contact
Santosh	dvr	4	9999999999
Harshith	dsr	3	8888899999
Mokshith	dsr	1	7777799999
xyz	abc	6	1234567890
Suresh	sf	7	3212344444
s3	f3	1	23456768898

DJANGO FORMS

without using model & model form we can create the forms using django forms.

To create forms

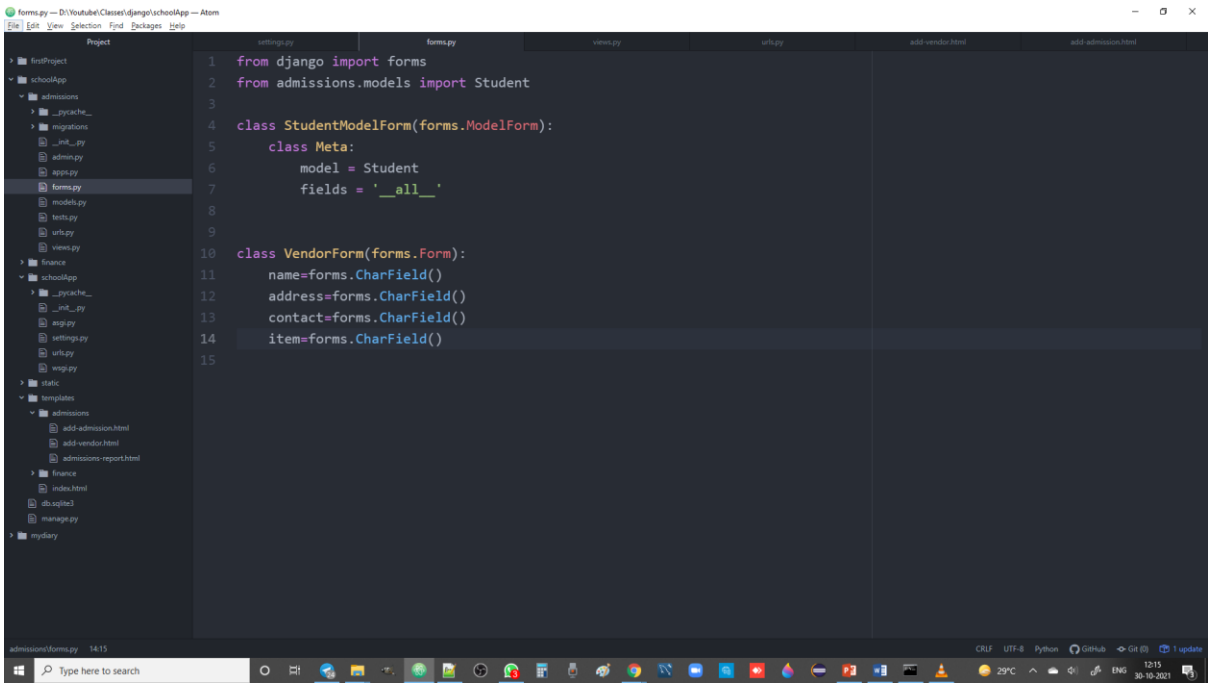
1) create a file forms.py in your application folder
2) Create a class that is inherited form forms.Form
3) define fields to be included in the form

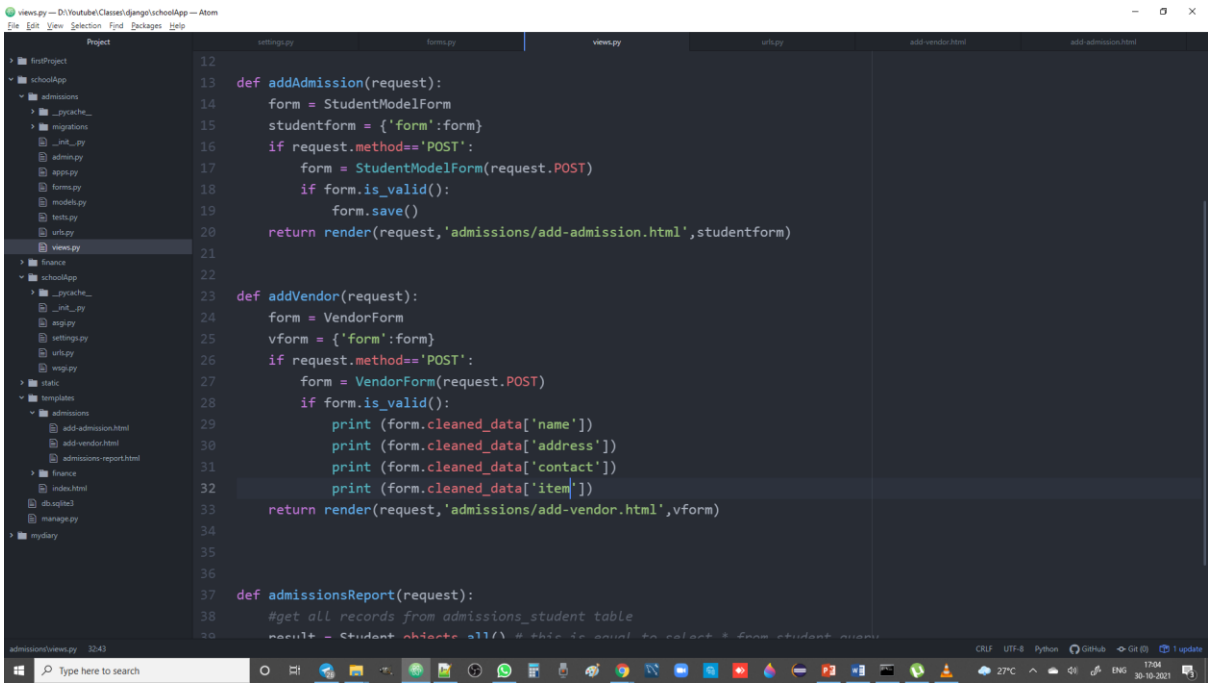
```
class VendorForm(forms.Form):
    name=form.CharField()
    address=form.CharField()
    contact=form.CharField()
    item=form.CharField()
```

4) Use Form in view

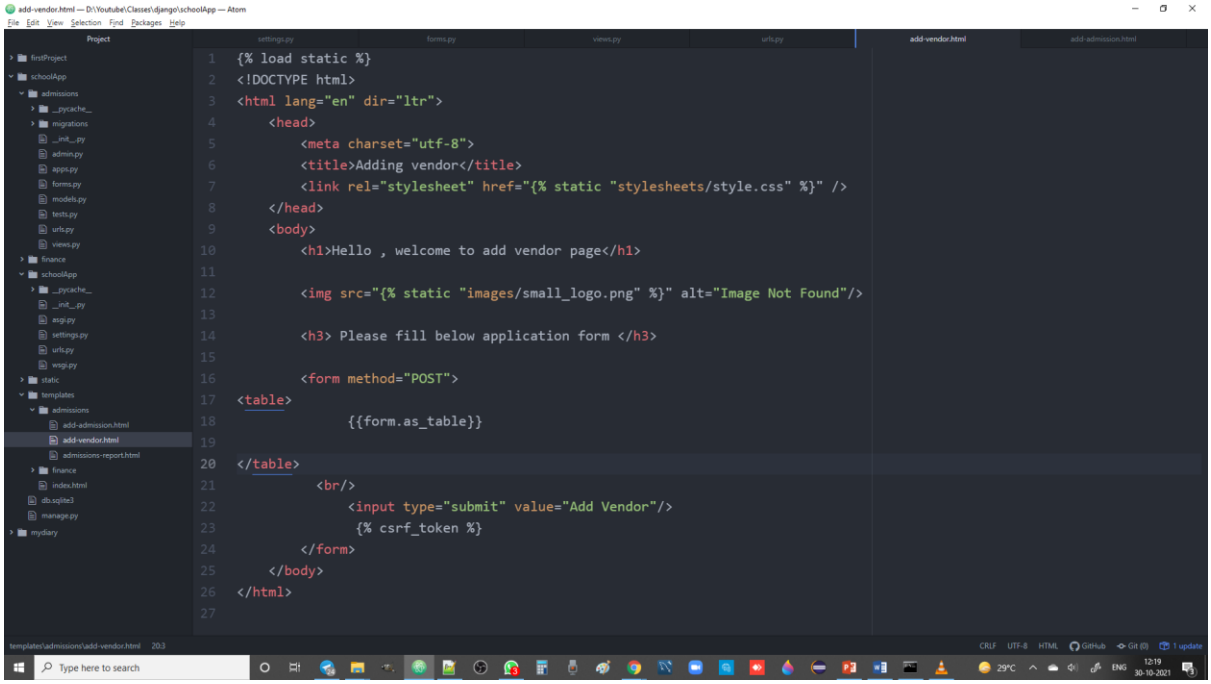
5) display the form in template

```
<form method="POST">
    {{form.as_p}}
    {% csrf_token %}
</form>
```

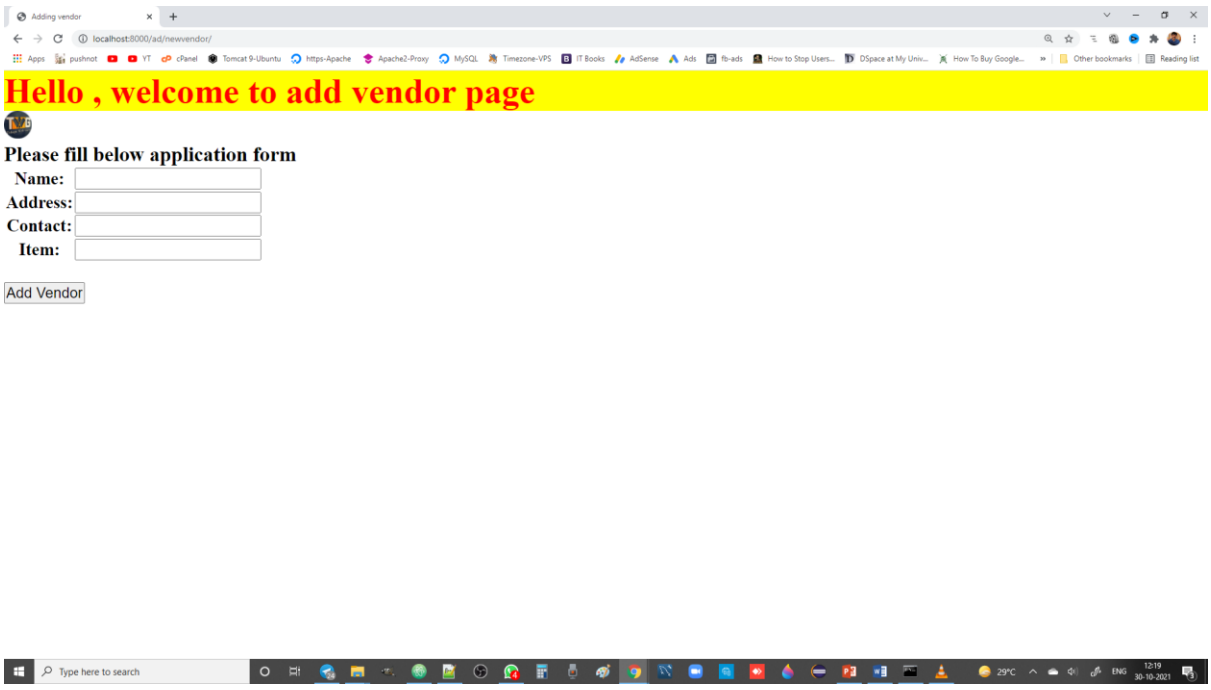


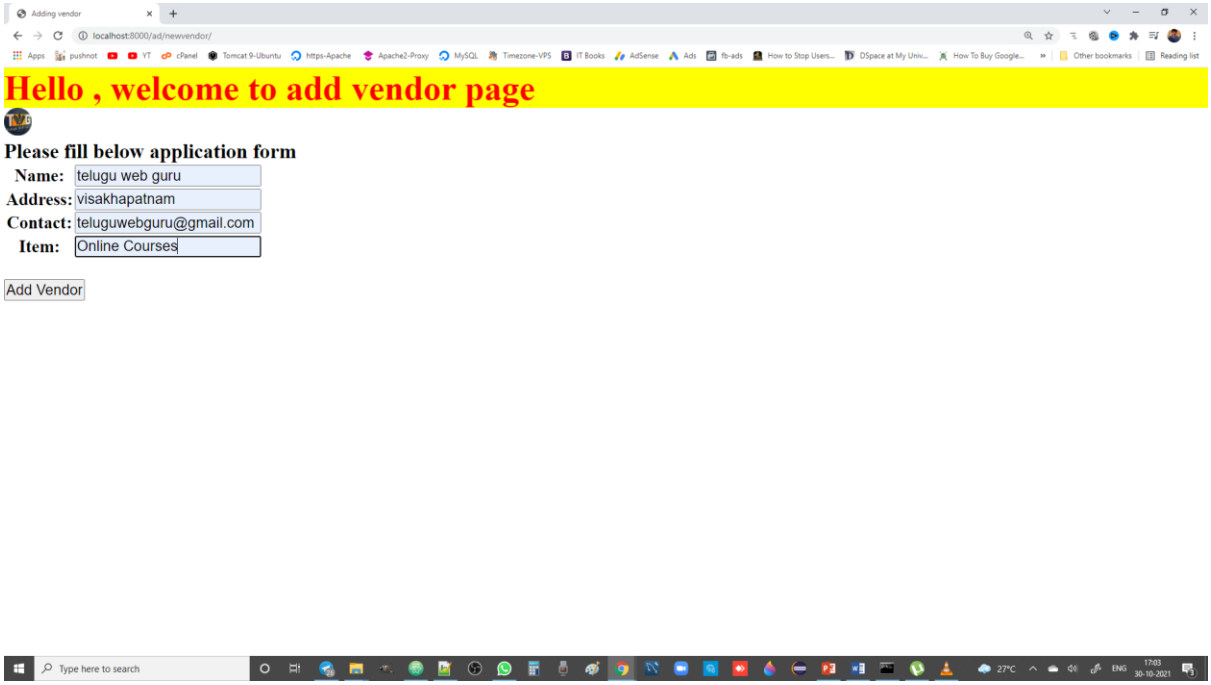


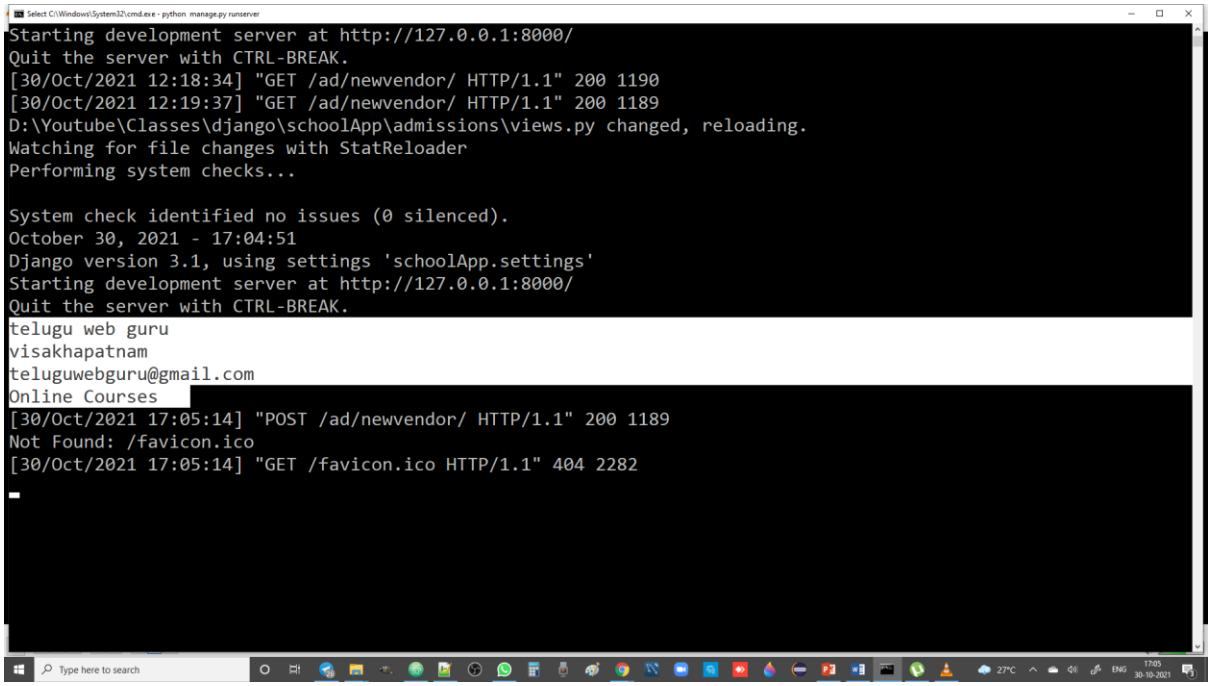
```
def addAdmission(request):
    form = StudentModelForm
    studentform = {'form':form}
    if request.method=='POST':
        form = StudentModelForm(request.POST)
        if form.is_valid():
            form.save()
    return render(request,'admissions/add-admission.html',studentform)


def addVendor(request):
    form = VendorForm
    vform = {'form':form}
    if request.method=='POST':
        form = VendorForm(request.POST)
        if form.is_valid():
            print (form.cleaned_data['name'])
            print (form.cleaned_data['address'])
            print (form.cleaned_data['contact'])
            print (form.cleaned_data['item'])
    return render(request,'admissions/add-vendor.html',vform)



def admissionsReport(request):
    #get all records from admissions_student table
```



```
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding vendor</title>
        <link rel="stylesheet" href="{% static "stylesheets/style.css" %}" />
    </head>
    <body>
        <h1>Hello , welcome to add vendor page</h1>

        <img src="{% static "images/small_logo.png" %}" alt="Image Not Found"/>

        <h3> Please fill below application form </h3>

        <form method="POST">
<table>
            {{form.as_table}}

</table>
         <br/>
            <input type="submit" value="Add Vendor"/>
             {% csrf_token %}
        </form>
    </body>
</html>
```


Adding vendor
localhost:8000/ad/newvendor/
Hello , welcome to add vendor page
Please fill below application form
Name:
Address:
Contact:
Item:
Add Vendor


Adding vendor
localhost:8000/ad/newvendor/
Hello , welcome to add vendor page
Please fill below application form
Name: telugu web guru
Address: visakhapatnam
Contact: teluguwebguru@gmail.com
Item: Online Courses
Add Vendor



observe that data entered in form is displayed here

CRUD OPERATIONS USING DJANGO

CRUD operations are Create, Read, Update, delete which we generally performed on data bases.

Now we will learn how to perform these crud operations.

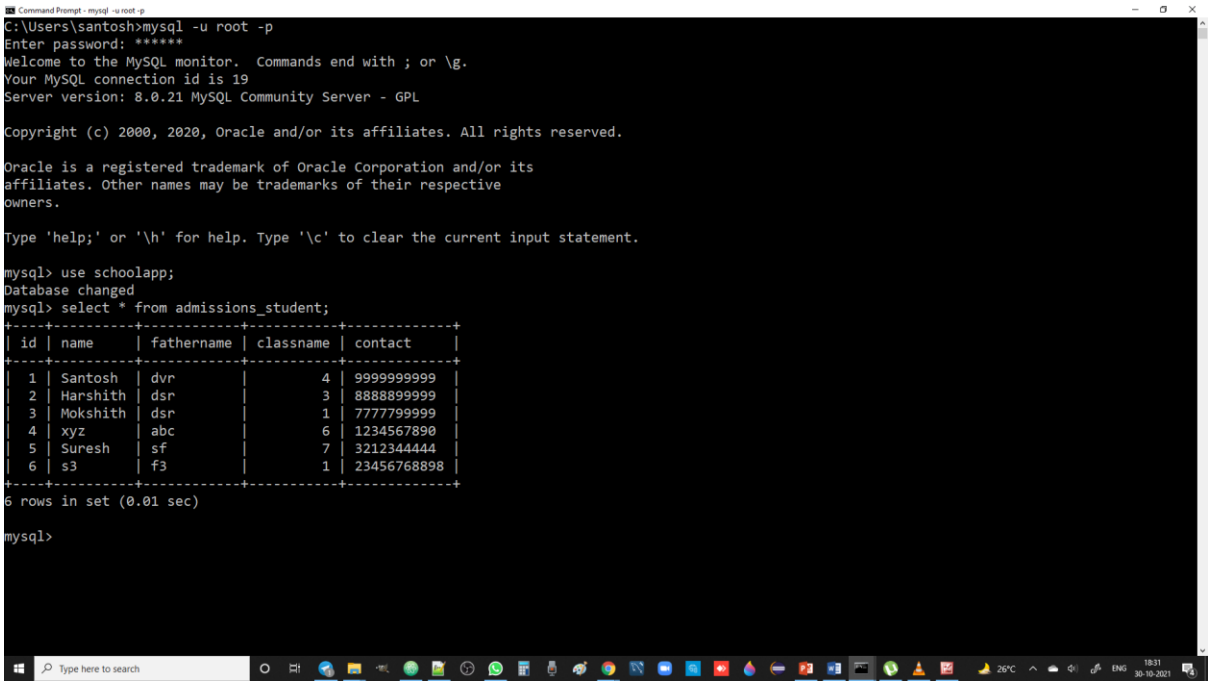
We already covered Create and Read in our previous lectures.

form.save() which we performed in above classes is Create operation

model.objects.all(), model.objects.get(..) these represents read operation.

Now let us learn how to update/delete some records based on id.

for this let us first check the existing table entries



There will be a slight difference between other URLs and update, delete. To update or delete records from the database we must specify the id of that record to uniquely identify and perform the actions.

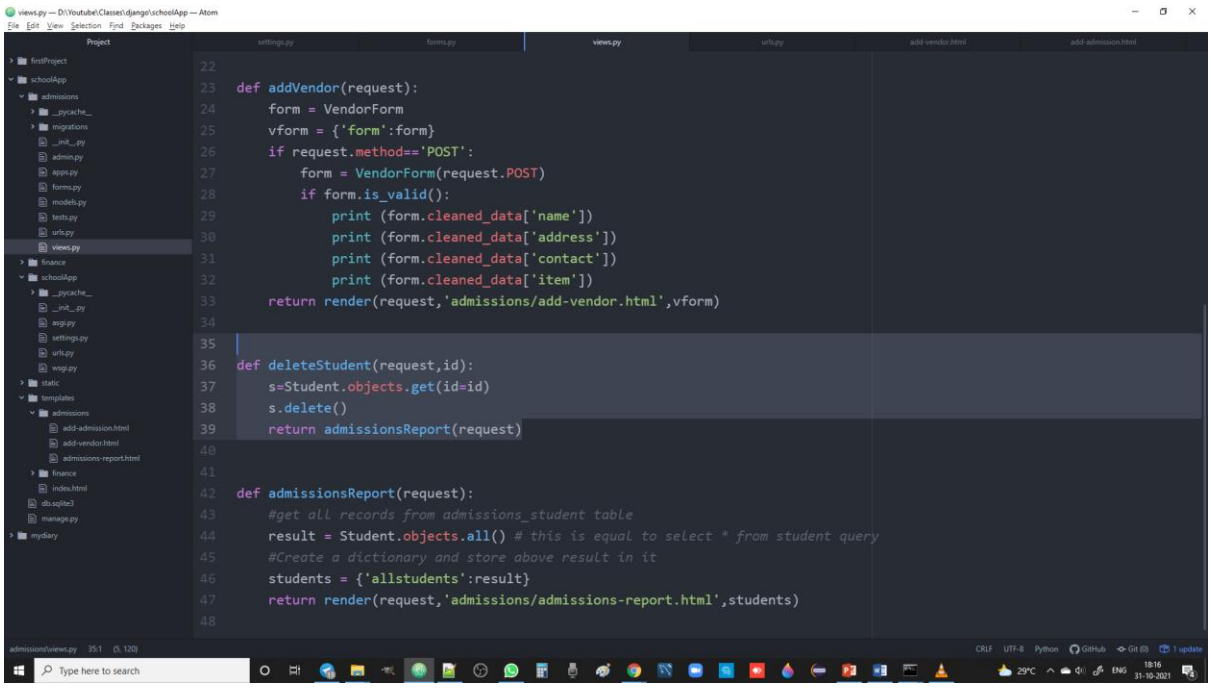
To delete a particular record with id 2 : First get the record and then perform delete operation.

```
def deleteStudent(request,id) :
    s = Student.objects.get(id = 2)
    s.delete()
```

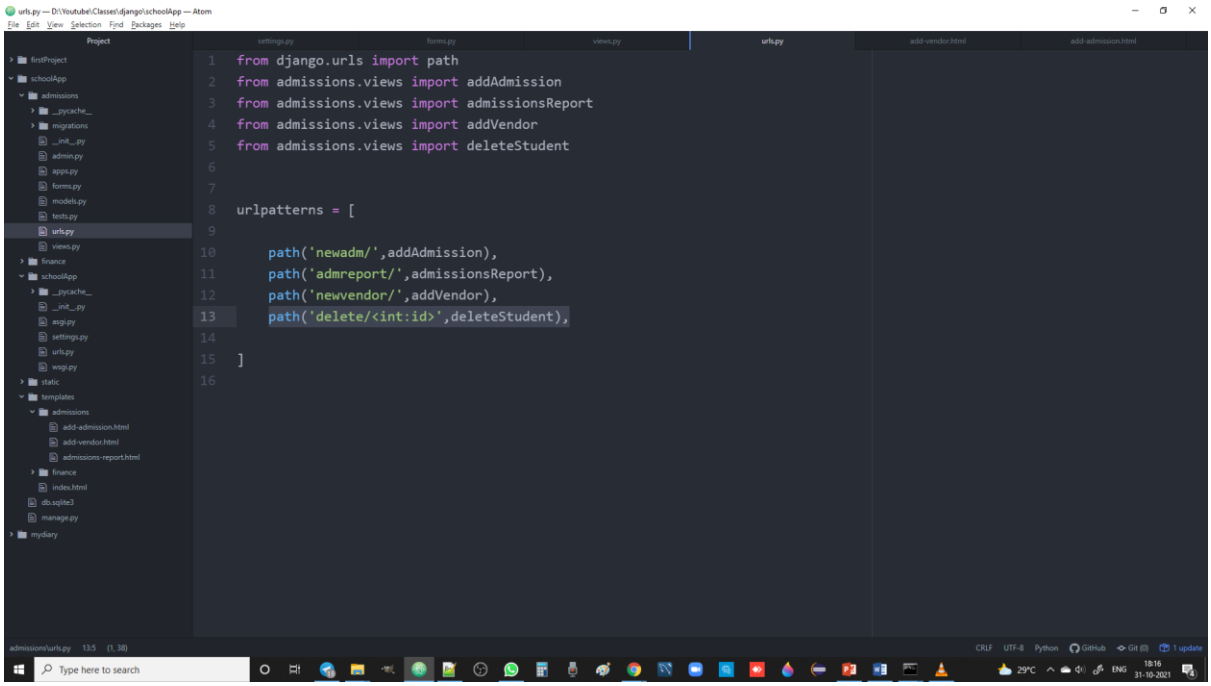
in url we must specify that an integer value will be passed.

path('delete/<int:id>', deleteStudent)

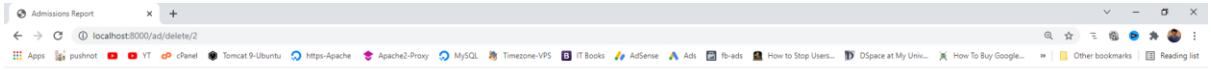
Now let us implement delete operation and try to delete the record with id 2 from the above table



```python
def addVendor(request):
    form = VendorForm
    vform = {'form':form}
    if request.method=='POST':
        form = VendorForm(request.POST)
        if form.is_valid():
            print (form.cleaned_data['name'])
            print (form.cleaned_data['address'])
            print (form.cleaned_data['contact'])
            print (form.cleaned_data['item'])
    return render(request,'admissions/add-vendor.html',vform)

def deleteStudent(request,id):
    s=Student.objects.get(id=id)
    s.delete()
    return admissionsReport(request)


def admissionsReport(request):
    #get all records from admissions_student table
    result = Student.objects.all() # this is equal to select * from student query
    #Create a dictionary and store above result in it
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students)
```



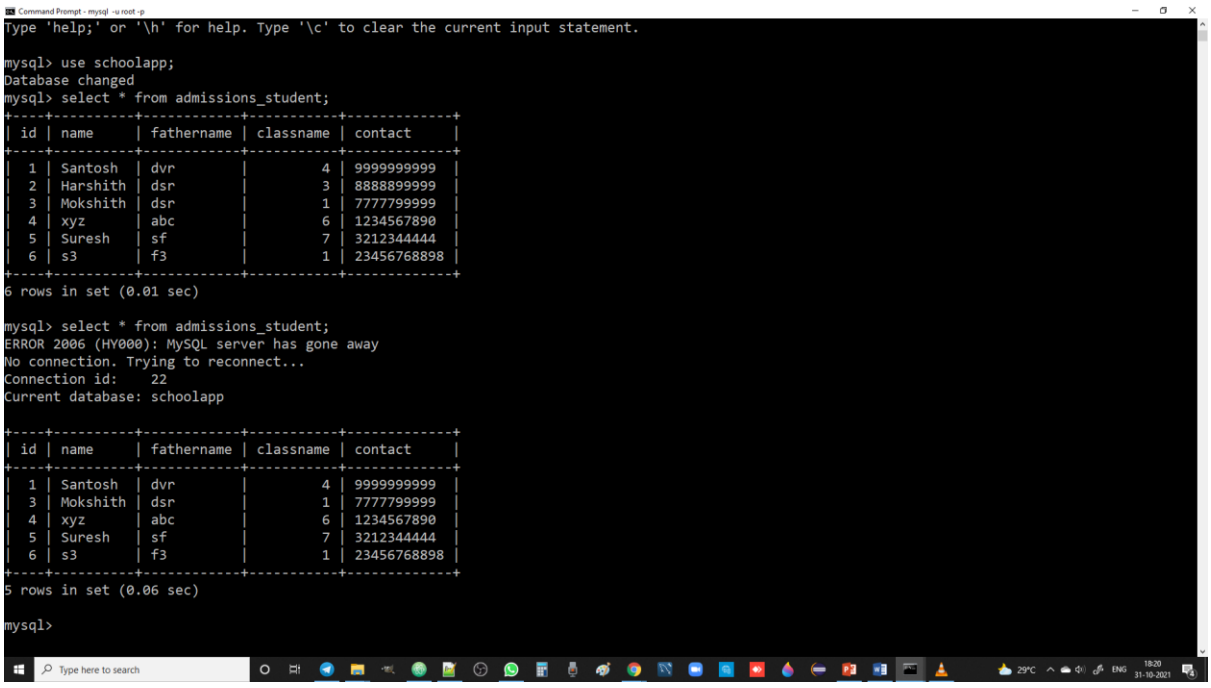
```python
from django.urls import path
from admissions.views import addAdmission
from admissions.views import admissionsReport
from admissions.views import addVendor
from admissions.views import deleteStudent


urlpatterns = [

    path('newadm/',addAdmission),
    path('admreport/',admissionsReport),
    path('newvendor/',addVendor),
    path('delete/<int:id>',deleteStudent),

]
```



Admissions Report

Student Name	Father Name	Class	Contact
Santosh	dvr	4	9999999999
Mokshith	dsr	1	7777799999
xyz	abc	6	1234567890
Suresh	sf	7	3212344444
s3	f3	1	23456768898

so, observe the database that the id=2 entry (Harshith) is deleted from the database.

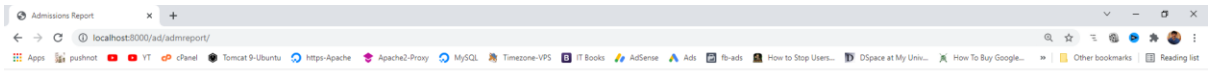


```
Command Prompt - mysql -u root -p
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use schoolapp;
Database changed
mysql> select * from admissions_student;
+----+----------+------------+-----------+-------------+
| id | name     | fathername | classname | contact     |
+----+----------+------------+-----------+-------------+
|  1 | Santosh  | dvr        |         4 | 9999999999  |
|  2 | Harshith | dsr        |         3 | 8888899999  |
|  3 | Mokshith | dsr        |         1 | 7777799999  |
|  4 | xyz      | abc        |         6 | 1234567890  |
|  5 | Suresh   | sf         |         7 | 3212344444  |
|  6 | s3       | f3         |         1 | 23456768898 |
+----+----------+------------+-----------+-------------+
6 rows in set (0.01 sec)

mysql> select * from admissions_student;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    22
Current database: schoolapp

+----+----------+------------+-----------+-------------+
| id | name     | fathername | classname | contact     |
+----+----------+------------+-----------+-------------+
|  1 | Santosh  | dvr        |         4 | 9999999999  |
|  3 | Mokshith | dsr        |         1 | 7777799999  |
|  4 | xyz      | abc        |         6 | 1234567890  |
|  5 | Suresh   | sf         |         7 | 3212344444  |
|  6 | s3       | f3         |         1 | 23456768898 |
+----+----------+------------+-----------+-------------+
5 rows in set (0.06 sec)

mysql>
```

Now let us change the admission report template to add delete option so that the page will become user friendly.

admissions-report.html — D:\Youtube\Classes\django\schoolApp — Atom

```
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Admissions Report</title>
    </head>
    <body>
        <h1>Admissions Report</h1>
        <table border="1">
            <tr>
                <th>Student Name</th>
                <th>Father Name</th>
                <th>Class</th>
                <th>Contact</th>
                <th>Action</th>
            </tr>
        {% for s in allstudents %}
        <tr>
            <td>{{s.name}}</td>
            <td>{{s.fathername}}</td>
            <td>{{s.classname}}</td>
            <td>{{s.contact}}</td>
            <td><a href="/ad/delete/{{s.id}}">delete</a></td>
        </tr>
        {% endfor %}
        </table>
    </body>
</html>
```



Admissions Report

localhost:8000/ad/admreport/

Admissions Report

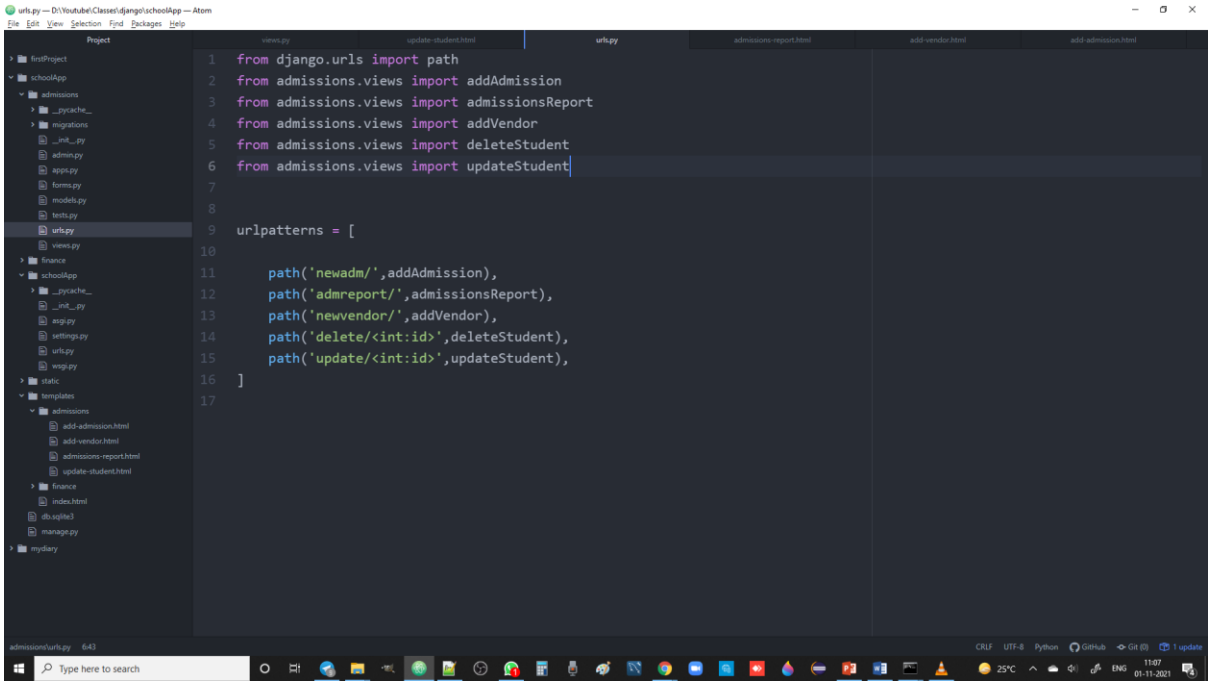
Student Name	Father Name	Class	Contact	Action
Santosh	dvr	4	9999999999	delete
Mokshith	dsr	1	7777799999	delete
xyz	abc	6	1234567890	delete
Suresh	sf	7	3212344444	delete
s3	f3	1	23456768898	delete

updating records :

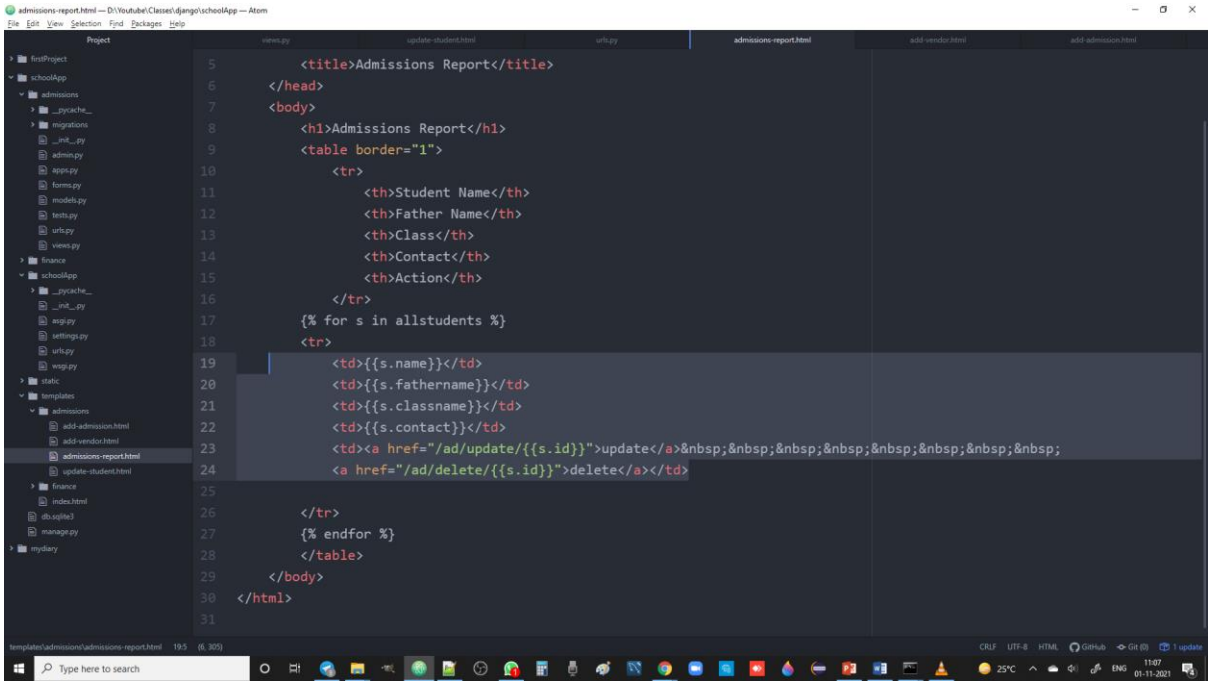
updating is also follows the same style. But to update values we need to show the form to the user with existing values. Then user will update the required values and show the results.

```python
def deleteStudent(request,id):
    s=Student.objects.get(id=id)
    s.delete()
    return admissionsReport(request)

def updateStudent(request,id):
    s=Student.objects.get(id=id)
    form = StudentModelForm(instance=s)
    dict = {'form':form}
    if request.method=='POST':
        form = StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
        return admissionsReport(request)

    return render(request,'admissions/update-student.html',dict)


def admissionsReport(request):
    #get all records from admissions_student table
    result = Student.objects.all() # this is equal to select * from student query
    #Create a dictionary and store above result in it
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students)
```

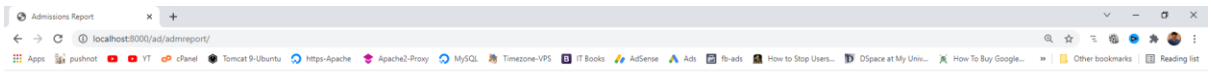
```html
{% load static %}
<!DOCTYPE html>
<html lang="en" dir="ltr">
    <head>
        <meta charset="utf-8">
        <title>Adding Admissions</title>
        <link rel="stylesheet" href="{% static "stylesheets/style.css" %}" />
    </head>
    <body>
        <h1>Hello , welcome to update student page</h1>

        <img src="{% static "images/small_logo.png" %}" alt="Image Not Found"/>

        <h3> Please fill below application form </h3>

        <form method="POST">

              {{form.as_p}}

              <br/>

              <input type="submit" value="Update Student"/>
               {% csrf_token %}
        </form>

    </body>
</html>
```



```
from django.urls import path
from admissions.views import addAdmission
from admissions.views import admissionsReport
from admissions.views import addVendor
from admissions.views import deleteStudent
from admissions.views import updateStudent


urlpatterns = [

    path('newadm/',addAdmission),
    path('admreport/',admissionsReport),
    path('newvendor/',addVendor),
    path('delete/<int:id>',deleteStudent),
    path('update/<int:id>',updateStudent),
]
```



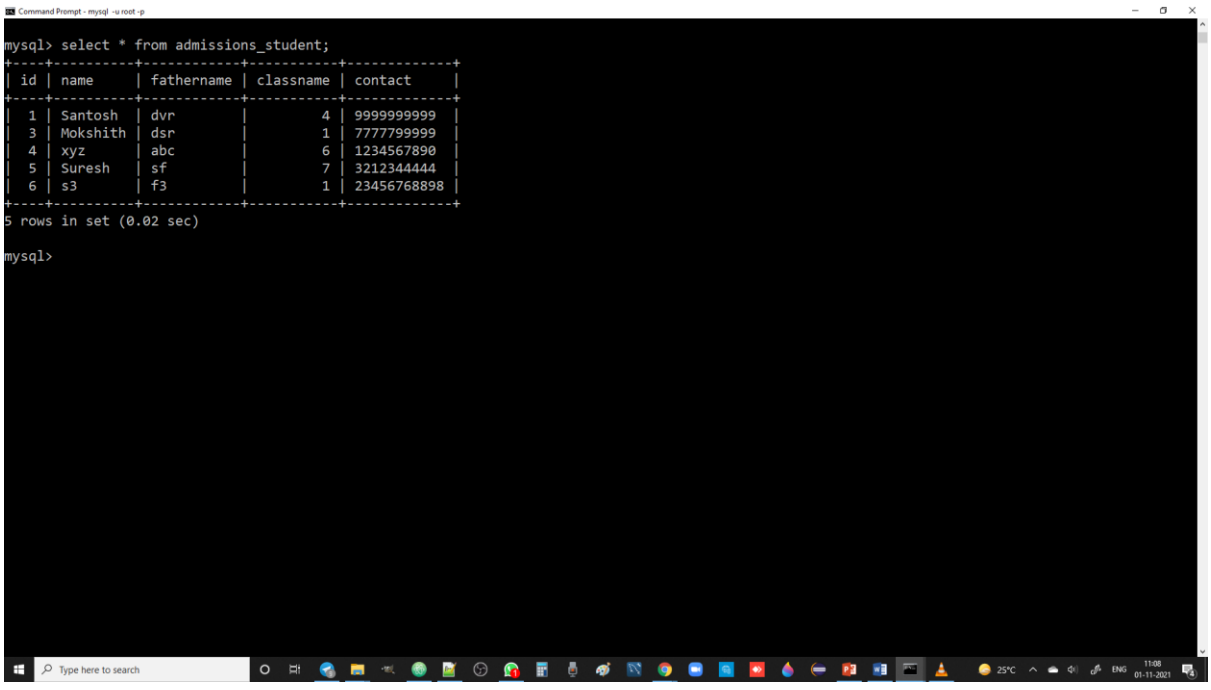
```
        <title>Admissions Report</title>
    </head>
    <body>
        <h1>Admissions Report</h1>
        <table border="1">
            <tr>
                <th>Student Name</th>
                <th>Father Name</th>
                <th>Class</th>
                <th>Contact</th>
                <th>Action</th>
            </tr>
        {% for s in allstudents %}
        <tr>
            <td>{{s.name}}</td>
            <td>{{s.fathername}}</td>
            <td>{{s.classname}}</td>
            <td>{{s.contact}}</td>
            <td><a href="/ad/update/{{s.id}}">update</a>        
            <a href="/ad/delete/{{s.id}}">delete</a></td>

        </tr>
        {% endfor %}
        </table>
    </body>
</html>
```



Admissions Report

Student Name	Father Name	Class	Contact	Action	
Santosh	dvr	4	9999999999	update	delete
Mokshith	dsr	1	7777799999	update	delete
xyz	abc	6	1234567890	update	delete
Suresh	sf	7	3212344444	update	delete
s3	f3	1	23456768898	update	delete

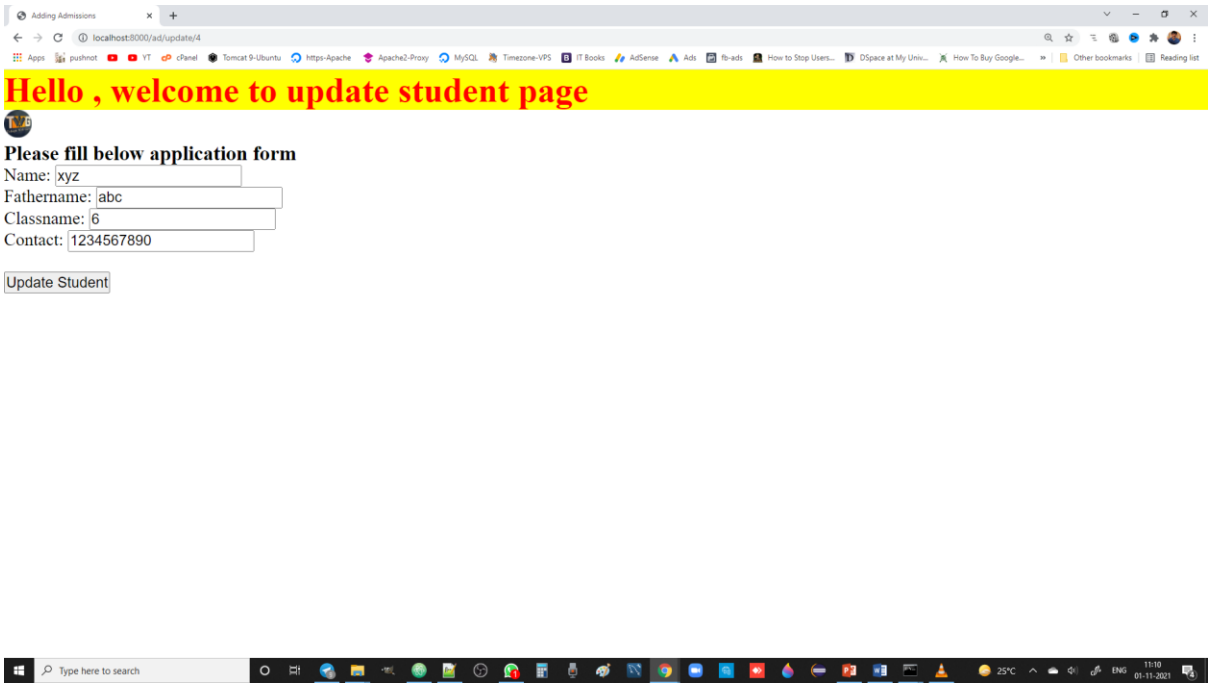
Now let us see the existing table data



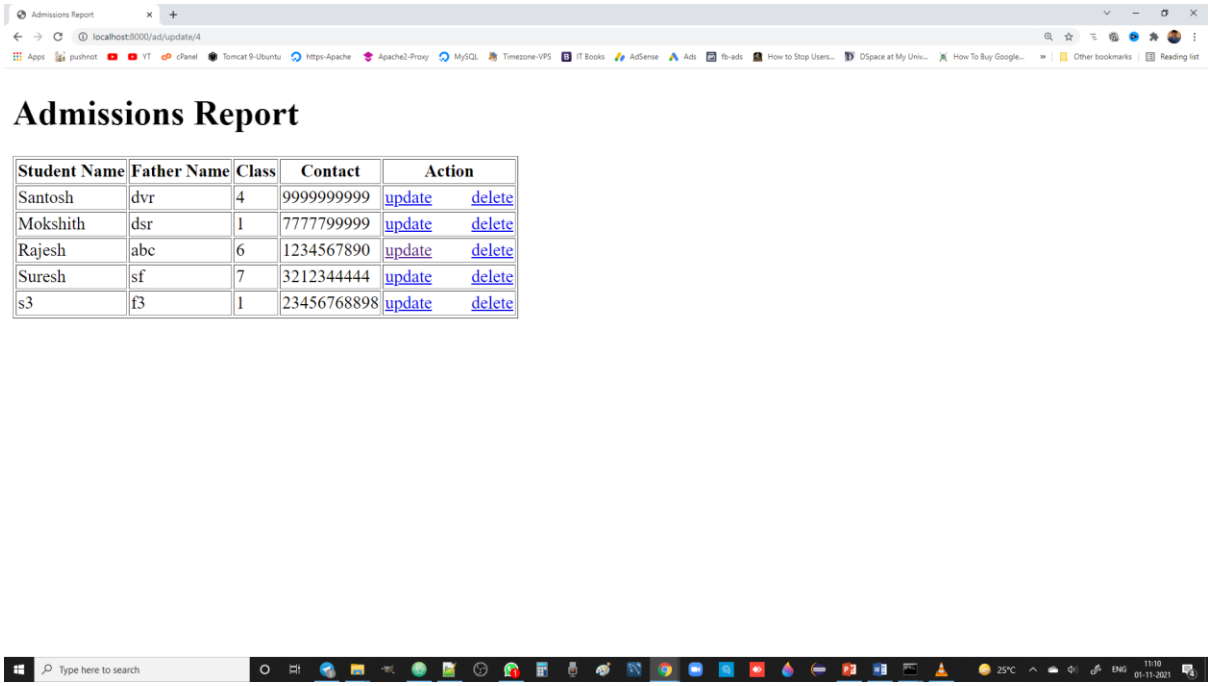
```
mysql> select * from admissions_student;
+----+----------+------------+-----------+-------------+
| id | name     | fathername | classname | contact     |
+----+----------+------------+-----------+-------------+
|  1 | Santosh  | dvr        |         4 | 9999999999  |
|  3 | Mokshith | dsr        |         1 | 7777799999  |
|  4 | xyz      | abc        |         6 | 1234567890  |
|  5 | Suresh   | sf         |         7 | 3212344444  |
|  6 | s3       | f3         |         1 | 23456768898 |
+----+----------+------------+-----------+-------------+
5 rows in set (0.02 sec)

mysql>
```

Now let us change the name of the student with id =4 from xyz to Rajesh

When we click on update button in admissionsreport page it will open a form for us



Now let us change the value of name to Rajesh and submit it.



Observe that student name is updated successfully.

Class Based Views in DJANGO

Class Based Views

We can create views by using classes also.

To create a class based views

1) we need to create a class that is inherited from django.views.generic.View

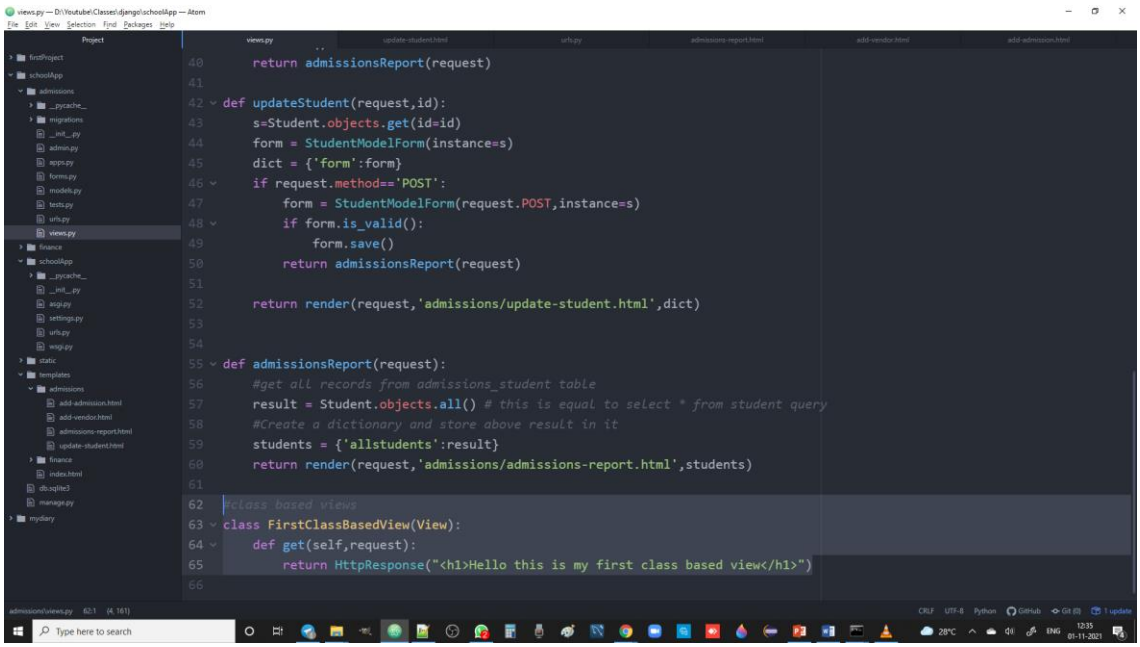
```
import django.views.generic import view
class exview(View):
    #class code
```

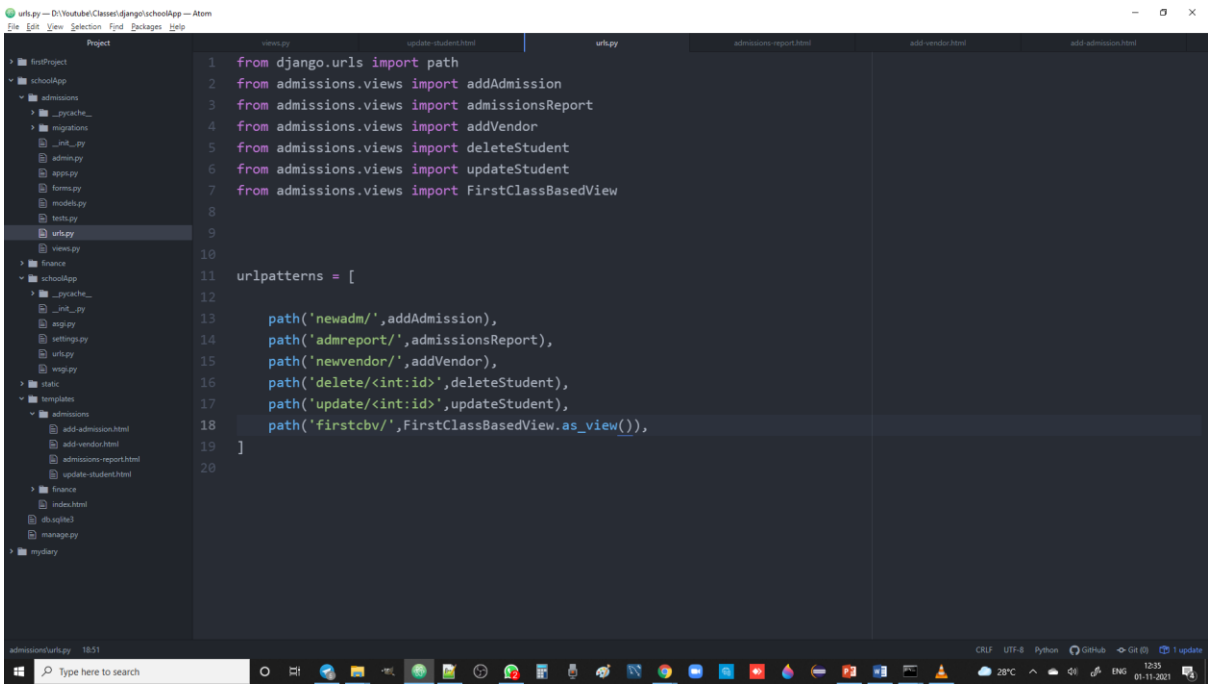
2) define separate methods for different http methods

```
def get(self,request):
    return HttpResponse("")
```

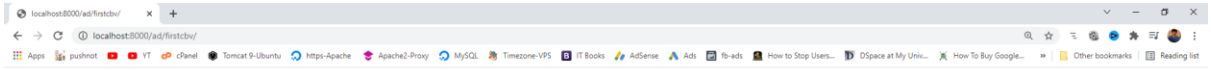
3) configure url in urls.py

```
path('',exview.as_view())
```





```python
from django.urls import path
from admissions.views import addAdmission
from admissions.views import admissionsReport
from admissions.views import addVendor
from admissions.views import deleteStudent
from admissions.views import updateStudent
from admissions.views import FirstClassBasedView



urlpatterns = [

    path('newadm/',addAdmission),
    path('admreport/',admissionsReport),
    path('newvendor/',addVendor),
    path('delete/<int:id>',deleteStudent),
    path('update/<int:id>',updateStudent),
    path('firstcbv/',FirstClassBasedView.as_view()),
]
```



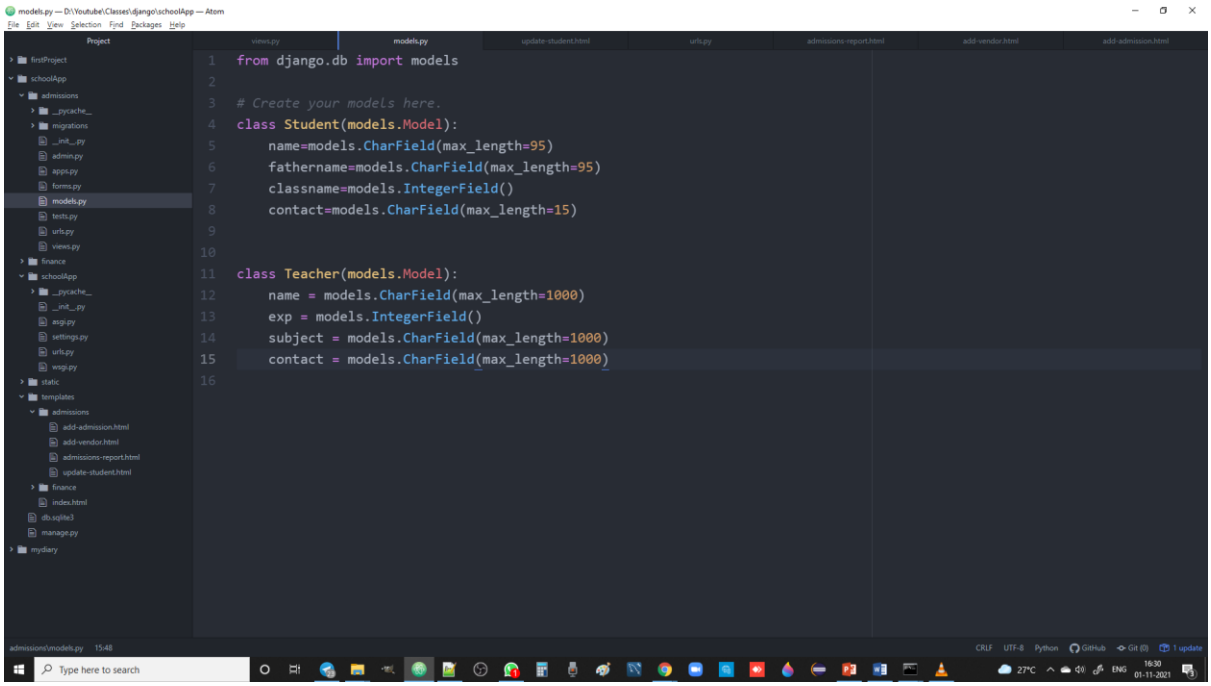
Hello this is my first class based view

CRUD Operations using Class Based Views in DJANGO

CRUD Operations using class Based Views

We can do CRUD operations by using various views available in django.views.generic : ListView, DetailView,UpdateView,DeleteView

First let us create a model Teacher and then perform crud operations using class based views.



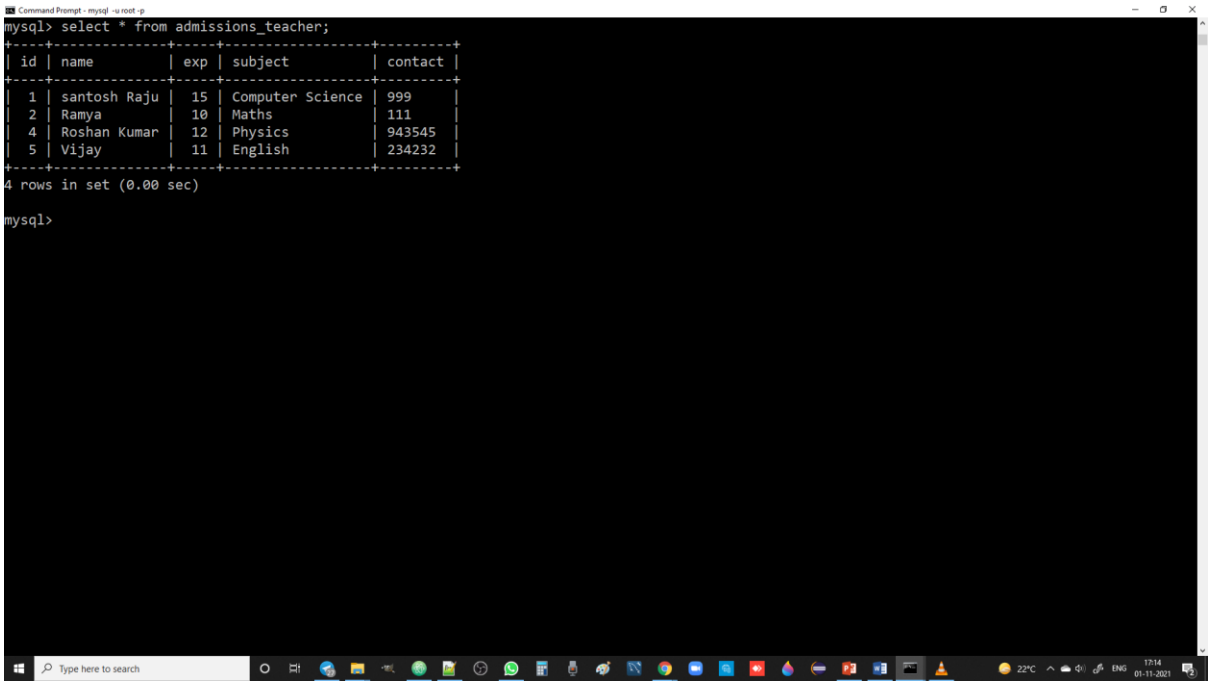
Now let us run makemigrations and migrate inorder to create teacher table automatically.

```
D:\Youtube\Classes\django\schoolApp>python manage.py makemigrations
Migrations for 'admissions':
  admissions\migrations\0002_teacher.py
    - Create model Teacher

D:\Youtube\Classes\django\schoolApp>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, admissions, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.

D:\Youtube\Classes\django\schoolApp>
```

```
| django_session             |
+----------------------------+
12 rows in set (0.03 sec)

mysql> desc admissions_teacher
    -> ;
+---------+---------------+------+-----+---------+----------------+
| Field   | Type          | Null | Key | Default | Extra          |
+---------+---------------+------+-----+---------+----------------+
| id      | int           | NO   | PRI | NULL    | auto_increment |
| name    | varchar(1000) | NO   |     | NULL    |                |
| exp     | int           | NO   |     | NULL    |                |
| subject | varchar(1000) | NO   |     | NULL    |                |
| contact | varchar(1000) | NO   |     | NULL    |                |
+---------+---------------+------+-----+---------+----------------+
5 rows in set (0.08 sec)

mysql>
```

Teacher table is created successfully.

Now let us insert data into the table and then perform crud operations on it.



Read Operations:

To perform Read operation, we need to use ListView. Steps are as follows

1) create a class based view that inherits from ListView
2) provide value to the model attribute (mandatory)
3) default template name is modelname_list.html you may pass your own template name by passing value to the template_name attribute.
4) default context object name (object that receives all the objects from ORM) is modelname_list. You can set your own context_object_name by passing the value to this attribute.
5) Create a template with name modelname_list.html and print the object's attributes from the list received (modelname_list).
6) Configure the url (classname.as_view())

Retrieve a single row using DetailView

To perform this, we need to use DetailView as follows.

1) create a class-based view that inherits from DetailView
2) provide value to the model attribute (mandatory)
3) default template name is modelname_detail.html you may pass your own template name by passing value to the template_name attribute.

4) default context object name (object that receives all the objects from ORM) is modelname. You can set your own context_object_name by passing the value to this attribute.
5) Create a template with name modelname_detail.html and print the object's attributes from the list received (modelname_list).
6) Configure the url - path('getdetails/<int:pk>/',classname.as_view())

Create Operations:

To perform this operation, we need to use CreateView. Steps are as follows

1) create a class based view that inherits from CreateView
2) provide value to the model attribute (mandatory)
3) provide value to fields (fields = (fields list separated by comma))
4) default template name is modelname_form.html you may pass your own template name by passing value to the template_name attribute.
5) create a method get_absolute_url in model class

```
get_absolute_url(self):
    return reverse('urlname',kwargs={'pk':self.pk})
```

6) Create a template with name modelname_ form.html and print the object's attributes from the list received (modelname_list).
7) Configure the url (classname.as_view())

Update Operations:

To perform Read operation, we need to use UpdateView. Steps are as follows

1) create a class based view that inherits from UpdateView
2) provide value to the model attribute (mandatory)
3) provide value to fields (fields = (fields list separated by comma))
4) default template name is modelname_form.html you may pass your own template name by passing value to the template_name attribute.
5) create a method get_absolute_url in model class

```
get_absolute_url(self):
    return reverse('urlname',kwargs={'pk':self.pk})
```

6) Create a template with name modelname_form.html and print the object's attributes from the list received (modelname_list).
7) Configure the url (classname.as_view())

Delete Operations:

To perform Read operation, we need to use DeleteView. Steps are as follows

1) create a class based view that inherits from DeleteView

2) provide value to the model attribute (mandatory)
3) provide value to success_url = reverse_lazy('urlname')
4) create model_confirm_delete.html to which django forward us and waits for our confirmation.
5) Configure the url (classname.as_view())

Security in Django:

Django supports authentication and authorization automatically.

Authentication Steps:

1) Add the auth urls
2) Create the login form
3) Secure the views
4) Create the users
5) Test it

Add the auth urls:

We will use include from django.urls to implement this.

1) Add a path in urls.py that points to accounts

   path ('accounts/',include('django.contrib.auth.urls'))

   This step will link all the urls related to auth to accounts. now we can use signin , signout by using /accounts/signin, /accounts/signout so on.,

Create the login form:

create a folder registration under templates

under this create a file login.html

```
<form method="POST">
    {%csrf_token%}
    {{form.as_p}}
    <button type="submit" value="Login">
</form>
```

Secure the views:

In the views.py apply the @login_required decorator to each view

from django.contrib.auth.decorators import login_required

now add @login_required above each view definition

Create the users:

Create user through admin form with the admin credentials that we created in earlier classes

Now we can login with the newly created user credentials and access the application.

To Logout from the application use the url /accounts/logout

To forward the users to a particular page after logout just include the following line at the end of the settings.py

LOGOUT_REDIRECT_URL = '/'

To implement a custom logout page then create a view with template and give that url in settings.py

Authorization:

Enter into admin login and click the user that we just created and there in the user page we can select and give selected authorizations to the particular user.

import permission_required as well along with login_required

Now add @permission_required('appname.actionname_modelname') with each view

---

You can add a group from admin login and assign the required authorizations to that group also so that all the users added into this group will automatically get those authorizations.

Template Inheritance:

create a template with common code by using block template tag

```
<html>
<head></head>
<body>
        {%block body_block%}


        {%endblock%}
</body>
</html>
```

In other Html Files just include the following statement

```
        {%extends 'appname/commontemplatename.html'%}
        {%block body_block%}
```

SESSION MANAGEMENT IN DJANGO

HTTP is a stateless protocol. That means server does not store the state of the application anywhere. whenever a new request comes then it losts all the temporary information (state) that is stored as part of processing previous request and it just start storing current request's state.

We can solve this problem by using one of the following techniques

1) Hidden Variables
2) Cookies
3) Sessions

Hidden Variables:

Example Program: views.py

```
from django.shortcuts import render

from admissions.models import Student

from admissions.forms import StudentModelForm

from admissions.forms import VendorForm

from django.views.generic import View,ListView,DetailView,CreateView,UpdateView,DeleteView

from django.http import HttpResponse

from admissions.models import Teacher

from django.urls import reverse_lazy

from django.contrib.auth.decorators import login_required,permission_required



# Create your views here.




#function based views
```

```
@login_required
def homepage(request):
    return render(request,'index.html',{'user':request.user})


def logoutUser(request):
    return render(request,'logout.html')


@login_required
def addAdmission(request):
    #retrieve values from hidden variables
    n = request.GET.get('nm')
    a = request.GET.get('addr')
    c = request.GET.get('cont')
    i = request.GET.get('it')
    dict = {"name":n,"address":a,"contact":c,"item":i}


    form = StudentModelForm
    studentform = {'form':form,"name":n,"address":a,"contact":c,"item":i}


    if request.method=='POST':
        form = StudentModelForm(request.POST)
        if form.is_valid():
            form.save()
```

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



    return render(request,'admissions/add-admission.html',studentform);




@login_required
@permission_required('admissions.view_student')
def admissionsReport(request):
    #get all the records from the table
    result = Student.objects.all(); #SELECT * FROM students
    #store it in dictionary students
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students);


@login_required
@permission_required('admissions.delete_student')
def deleteStudent(request,id):
    s = Student.objects.get(id=id) # select * from admissions_student where id=idvalue
    s.delete()
    return admissionsReport(request)



@login_required
```

```
@permission_required('admissions.change_student') #add delete change view
def updateStudent(request,id):
    s = Student.objects.get(id=id)
    form = StudentModelForm(instance=s)
    dict = {'form':form}

    if request.method=='POST':
        form = StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
        return admissionsReport(request)

    return render(request,'admissions/update-admission.html',dict)

@login_required
def addVendor(request):
    form = VendorForm
    vform = {'form':form}

    if request.method=='POST':
```

```
        form = VendorForm(request.POST)
        if form.is_valid():
            n = form.cleaned_data['name']
            a = form.cleaned_data['address']
            c = form.cleaned_data['contact']
            i = form.cleaned_data['item']
            dict = {"name":n,"address":a,"contact":c,"item":i}
        return render(request,'index.html',dict);

    return render(request,'admissions/add-vendor.html',vform);



#class based view


class FirstClassBasedView(View):
    def get(self,request):
        return HttpResponse("<h1>Hello ... this is my first class based view</h1>")




class TeacherRead(ListView):
    model = Teacher
```

```
class GetTeacher(DetailView):

    model = Teacher




class AddTeacher(CreateView):

    model = Teacher

    fields = ('name','subject','exp','contact')




class UpdateTeacher(UpdateView):

    model = Teacher

    fields = ('name','contact')




class DeleteTeacher(DeleteView):

    model = Teacher

    success_url = reverse_lazy('listteachers')
```

index.html

```
{% load static %}

<html>

<head>

<title>add admissions page</title>

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

</head>
```

```
<body>

<h2>

<img src="{% static "images/small_logo.png"  %}" width="100" height="100" alt="Image Not Found"/>

 telugu web guru academy

</h2>

<br/>

<hr/>

<br/>

<br/><br/><br/>



<div class="module">

<h1>ADMISSIONS</h1><br/>



<form action="/ad/newadm">

    {% csrf_token %}

<button type="submit" name="button">Add Admission</button></a><br/><br/>

<input type="hidden" name="nm" value="{{name}}">

<input type="hidden" name="addr" value="{{address}}">

<input type="hidden" name="cont" value="{{contact}}">

<input type="hidden" name="it" value="{{item}}">

</form>



<a href="/ad/admreport"><button type="button" name="button">Admission Report</button></a><br/><br/>
```

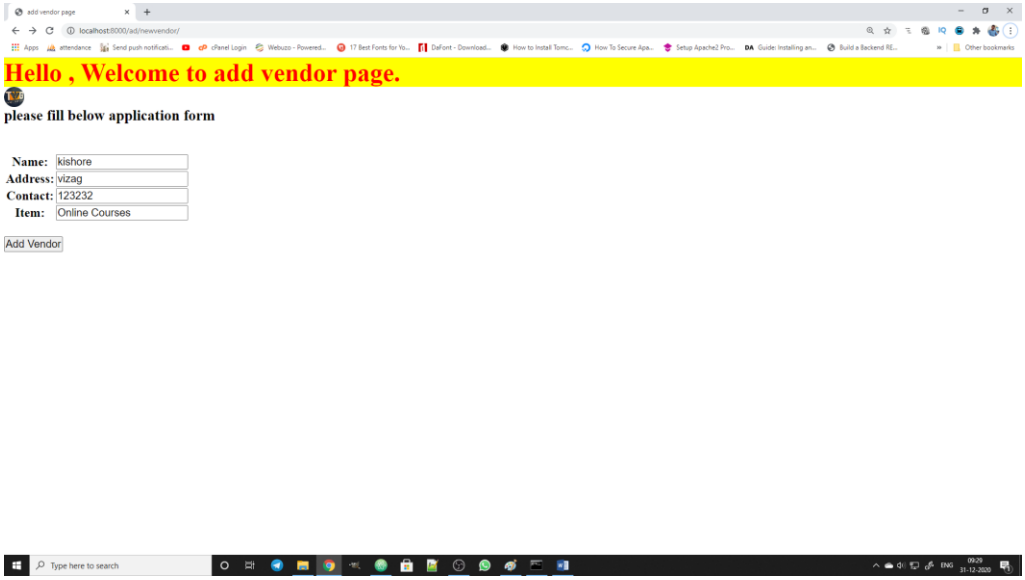
```
</div>



<div class="module">

<h1>FINANCE</h1><br/>

<a href="/fin/feecoll"><button type="button" name="button">Fee
Collection</button></a><br/><br/>

<a href="/fin/collectionreport"><button type="button" name="button">Fee Collection
Report</button></a><br/><br/>

<a href="/fin/duesreport"><button type="button" name="button">Fee Dues
Report</button></a>

</div>




<br/><br/>



<a href="/accounts/logout">Logout</a>




<br/><br/>



{{name}}<br/>

{{address}}<br/>

{{contact}}<br/>

{{item}}<br/>



</body>
```

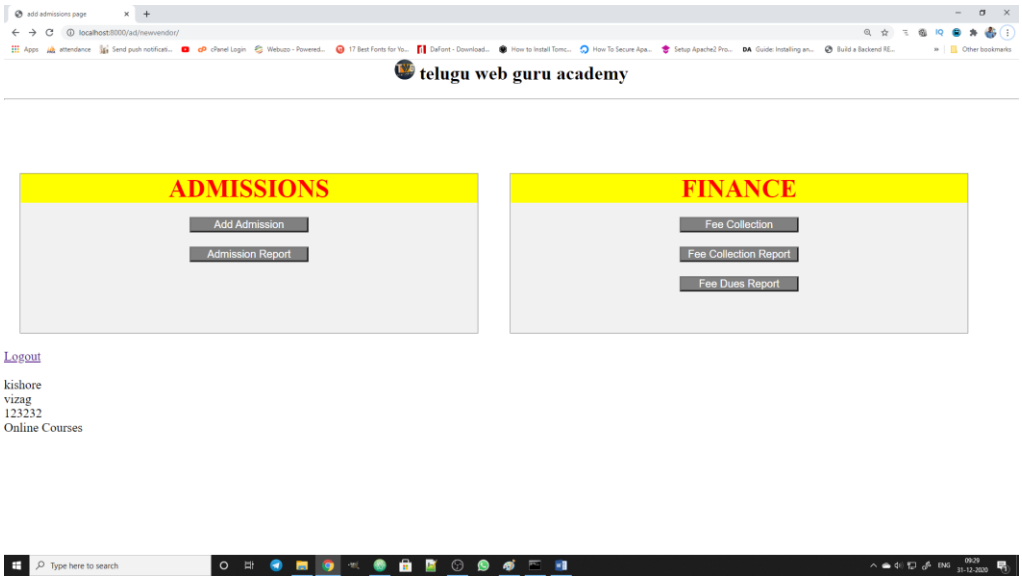
```
</html>
```

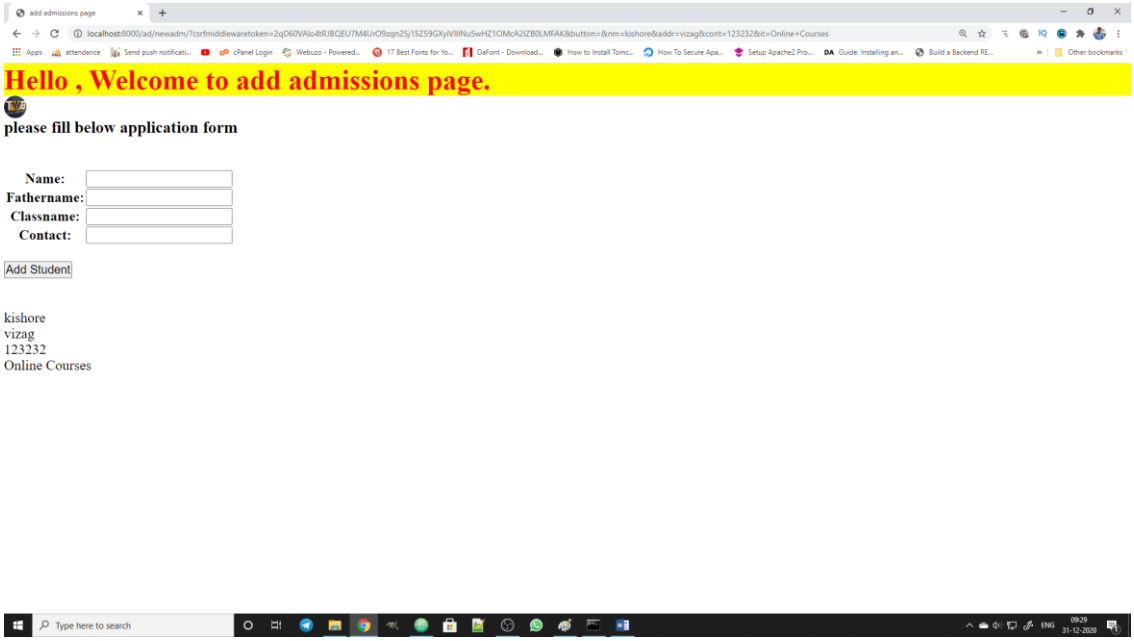
add-admission.html

```
{% load static %}

<html>

<head>

<title>add admissions page</title>

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

</head>

<body>




<h1>Hello , Welcome to add admissions page.</h1>



<img src="{% static "images/small_logo.png"  %}" width="100" height="100" alt="Image Not
Found"/>



<h3>please fill below application form</h3><br/><br/>



<form method="POST">

<table>

  {{form.as_table}}

</table>
```

```
<br/>

<input type="submit" value="Add Student">


{% csrf_token %}

</form>


<br/><br/>


{{name}}<br/>

{{address}}<br/>

{{contact}}<br/>

{{item}}<br/>


</body>


</html>
```


Hello , Welcome to add vendor page.
please fill below application form
Name: kishore
Address: vizag
Contact: 123232
Item: Online Courses
Add Vendor


telugu web guru academy
ADMISSIONS
Add Admission
Admission Report
FINANCE
Fee Collection
Fee Collection Report
Fee Dues Report
Logout
kishore
vizag
123232
Online Courses



cookie :

The following are used to set and retrieve the cookies in django

response.set_cookie(name,value,expirytime)

requests.COOKIES

The following methods are used to check whether cookies are enabled in client browser or not

request.session.set_test_cookie()

request.session.test_cookie_worked()

request.session.delete_test_cookie()

```
from django.shortcuts import render

from admissions.models import Student

from admissions.forms import StudentModelForm

from admissions.forms import VendorForm

from django.views.generic import View,ListView,DetailView,CreateView,UpdateView,DeleteView

from django.http import HttpResponse

from admissions.models import Teacher
```

```
from django.urls import reverse_lazy

from django.contrib.auth.decorators import login_required,permission_required



# Create your views here.




#function based views



@login_required

def homepage(request):

    return render(request,'index.html',{'user':request.user})



def logoutUser(request):

    return render(request,'logout.html')



@login_required

def addAdmission(request):

    form = StudentModelForm

    studentform = {'form':form}



    if request.method=='POST':

        form = StudentModelForm(request.POST)

        if form.is_valid():

            form.save()
```

```
        return homepage(request)


    return render(request,'admissions/add-admission.html',studentform);




@login_required
@permission_required('admissions.view_student')
def admissionsReport(request):
    #get all the records from the table
    result = Student.objects.all(); #SELECT * FROM students
    #store it in dictionary students
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students);


@login_required
@permission_required('admissions.delete_student')
def deleteStudent(request,id):
    s = Student.objects.get(id=id) # select * from admissions_student where id=idvalue
    s.delete()
    return admissionsReport(request)




@login_required
```

```
@permission_required('admissions.change_student') #add delete change view
def updateStudent(request,id):
    s = Student.objects.get(id=id)
    form = StudentModelForm(instance=s)
    dict = {'form':form}

    if request.method=='POST':
        form = StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
        return admissionsReport(request)

    return render(request,'admissions/update-admission.html',dict)

@login_required
def addVendor(request):
    form = VendorForm
    vform = {'form':form}

    if request.method=='POST':
```

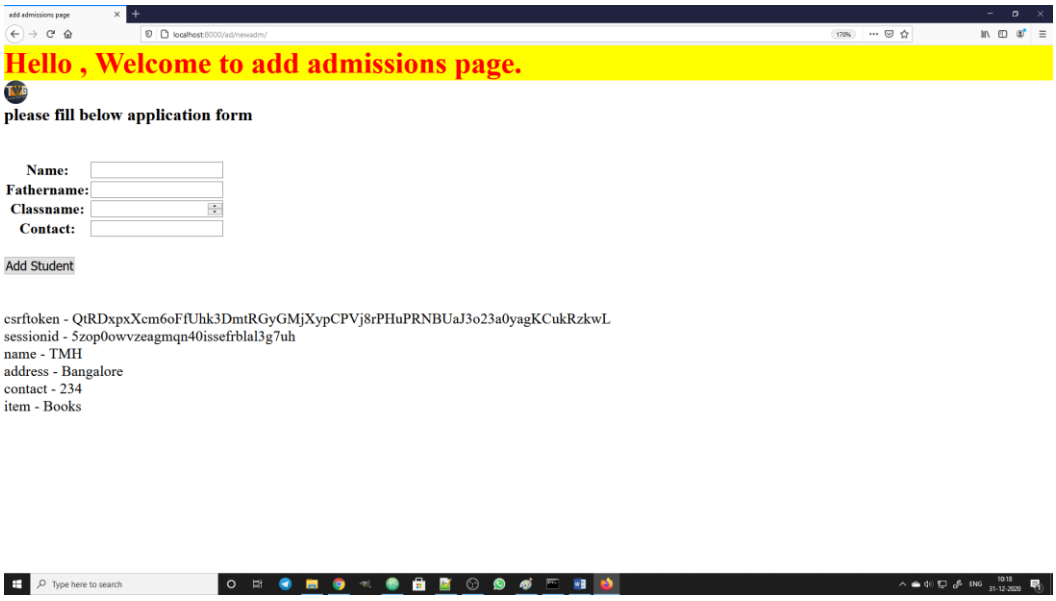
```
        form = VendorForm(request.POST)
        if form.is_valid():
            n = form.cleaned_data['name']
            a = form.cleaned_data['address']
            c = form.cleaned_data['contact']
            i = form.cleaned_data['item']

            response = render(request,'index.html');
            response.set_cookie("name",n)
            response.set_cookie("address",a)
            response.set_cookie("contact",c)
            response.set_cookie("item",i)

        return response

    return render(request,'admissions/add-vendor.html',vform);



#class based view

class FirstClassBasedView(View):
    def get(self,request):
        return HttpResponse("<h1>Hello ... this is my first class based view</h1>")
```

```
class TeacherRead(ListView):
    model = Teacher



class GetTeacher(DetailView):
    model = Teacher



class AddTeacher(CreateView):
    model = Teacher
    fields = ('name','subject','exp','contact')



class UpdateTeacher(UpdateView):
    model = Teacher
    fields = ('name','contact')



class DeleteTeacher(DeleteView):
    model = Teacher
    success_url = reverse_lazy('listteachers')
```

add-admission.html

```
{% load static %}

<html>

<head>

<title>add admissions page</title>

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

</head>

<body>




<h1>Hello , Welcome to add admissions page.</h1>



<img src="{% static "images/small_logo.png"  %}" width="100" height="100" alt="Image Not
Found"/>



<h3>please fill below application form</h3><br/><br/>



<form method="POST">

<table>

  {{form.as_table}}

</table>




<br/>

<input type="submit" value="Add Student">
```

```
{% csrf_token %}

</form>


<br/><br/>

{% for key,value in request.COOKIES.items %}

{{key}} - {{value}}<br/>

{% endfor %}

</body>


</html>
```



Session :

Session is the time between a user login and user logout

or

Sequence of actions performed by a user

request.session.get(session-name,default-value)

request.session[name]=value

```
from django.shortcuts import render

from admissions.models import Student

from admissions.forms import StudentModelForm

from admissions.forms import VendorForm

from django.views.generic import View,ListView,DetailView,CreateView,UpdateView,DeleteView

from django.http import HttpResponse

from admissions.models import Teacher

from django.urls import reverse_lazy

from django.contrib.auth.decorators import login_required,permission_required



# Create your views here.




#function based views


@login_required

def homepage(request):

    return render(request,'index.html',{'user':request.user})


def logoutUser(request):

    return render(request,'logout.html')


@login_required
```

```
def addAdmission(request):
    form = StudentModelForm
    studentform = {'form':form}

    if request.method=='POST':
        form = StudentModelForm(request.POST)
        if form.is_valid():
            form.save()
        return homepage(request)

    return render(request,'admissions/add-admission.html',studentform);



@login_required
@permission_required('admissions.view_student')
def admissionsReport(request):
    #get all the records from the table
    result = Student.objects.all(); #SELECT * FROM students
    #store it in dictionary students
    students = {'allstudents':result}
    return render(request,'admissions/admissions-report.html',students);

@login_required
```

```
@permission_required('admissions.delete_student')
def deleteStudent(request,id):
    s = Student.objects.get(id=id) # select * from admissions_student where id=idvalue
    s.delete()
    return admissionsReport(request)



@login_required
@permission_required('admissions.change_student') #add delete change view
def updateStudent(request,id):
    s = Student.objects.get(id=id)
    form = StudentModelForm(instance=s)
    dict = {'form':form}

    if request.method=='POST':
        form = StudentModelForm(request.POST,instance=s)
        if form.is_valid():
            form.save()
        return admissionsReport(request)







    return render(request,'admissions/update-admission.html',dict)
```

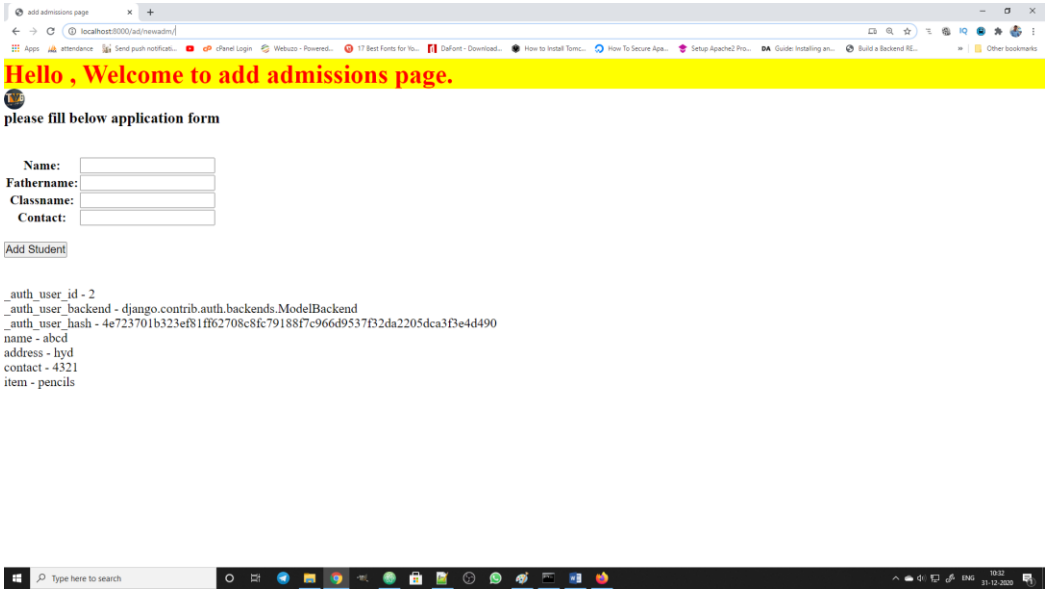
```
@login_required
def addVendor(request):
    form = VendorForm
    vform = {'form':form}

    if request.method=='POST':
        form = VendorForm(request.POST)
        if form.is_valid():
            n = form.cleaned_data['name']
            a = form.cleaned_data['address']
            c = form.cleaned_data['contact']
            i = form.cleaned_data['item']

            response = render(request,'index.html');

            request.session['name']=n;
            request.session['address']=a;
            request.session['contact']=c;
            request.session['item']=i;

        return response
```

```
    return render(request,'admissions/add-vendor.html',vform);




#class based view


class FirstClassBasedView(View):

    def get(self,request):

        return HttpResponse("<h1>Hello ... this is my first class based view</h1>")




class TeacherRead(ListView):

    model = Teacher



class GetTeacher(DetailView):

    model = Teacher



class AddTeacher(CreateView):

    model = Teacher

    fields = ('name','subject','exp','contact')
```

```
class UpdateTeacher(UpdateView):

    model = Teacher

    fields = ('name','contact')




class DeleteTeacher(DeleteView):

    model = Teacher

    success_url = reverse_lazy('listteachers')
```

```
{% load static %}

<html>

<head>

<title>add admissions page</title>

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

</head>

<body>




<h1>Hello , Welcome to add admissions page.</h1>



<img src="{% static "images/small_logo.png"  %}" width="100" height="100" alt="Image Not
Found"/>



<h3>please fill below application form</h3><br/><br/>
```

```
<form method="POST">
<table>
  {{form.as_table}}
</table>



<br/>
<input type="submit" value="Add Student">


{% csrf_token %}
</form>


<br/><br/>
{% for key,value in request.session.items %}
{{key}} - {{value}}<br/>
{% endfor %}
</body>


</html>
```


add admissions page
localhost:8000/ad/newadm/
Hello , Welcome to add admissions page.
please fill below application form
Name:
Fathername:
Classname:
Contact:
Add Student
_auth_user_id - 2
_auth_user_backend - django.contrib.auth.backends.ModelBackend
_auth_user_hash - 4e723701b323ef81ff62708c8fc79188f7c966d9537f32da2205dca3f3e4d490
name - abcd
address - hyd
contact - 4321
item - pencils
Type here to search