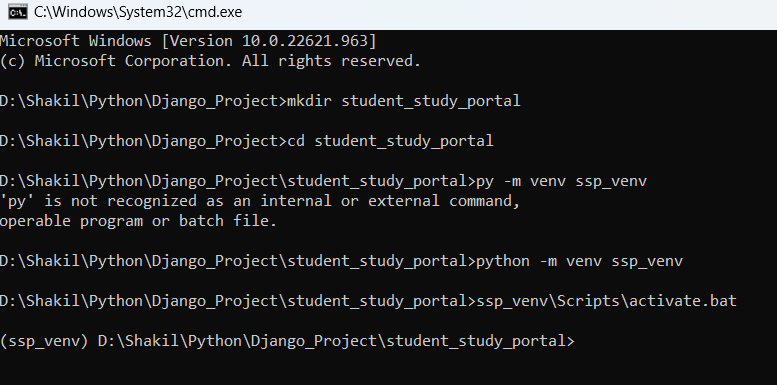
# **1. Installation and Settings**

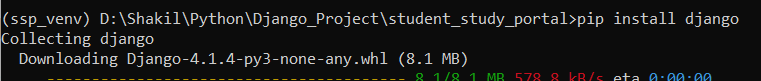
* Virtual environment setup

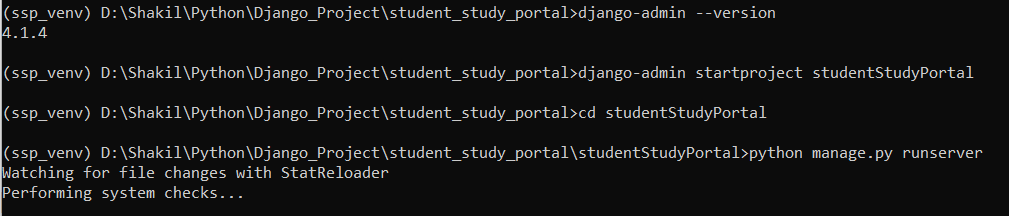
1. mkdir student\_study\_portal [folder create]
2. cd student\_study\_portal [enter into the folder]
3. python -m venv ssp\_venv [create environment]
4. ssp\_venv\Scripts\activate.bat [activate]
5. cd..\ [one foldr back] and cd ..\..\ [two folder back]



* Django install, create and run project

1. pip install Django
2. django-admin startproject studentStudyPortal
3. python manage.py runserver



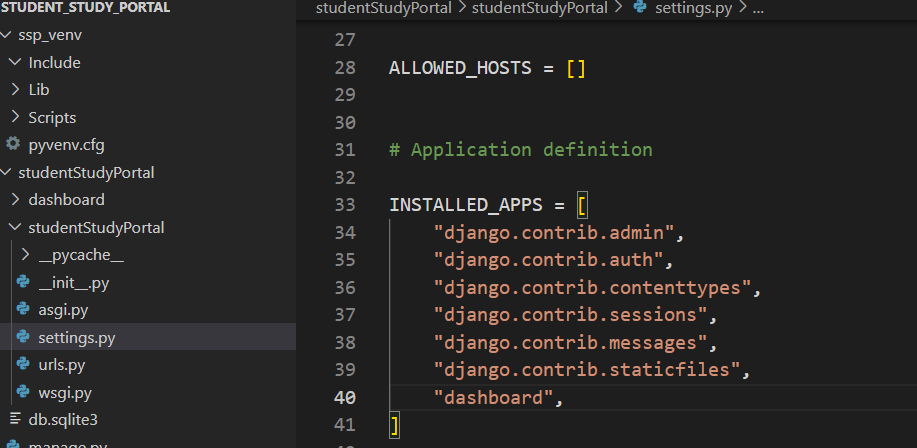


* **Open root folder in visual code [student\_study\_portal where Django installed]**
* **Create app**

1. python manage.py startapp dashboard



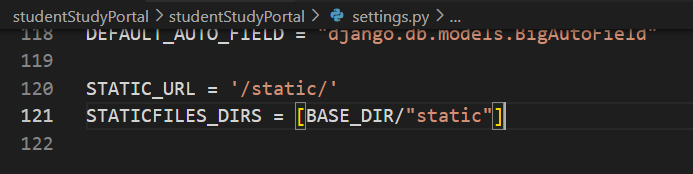
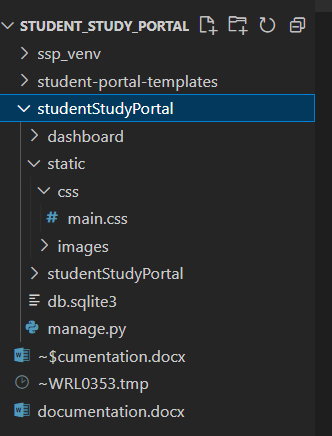
* Put app name in project [settings.py]



# **2. Static files and Template Mastering**

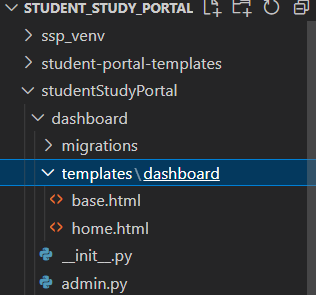
* Move static css and images into project

1. Create a folder name static in project root
2. Put static urls and directory in project settings.py



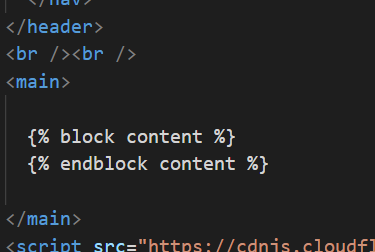
* Templates folder in app

1. Create templates->dashboard->copy home.html and base.html [folder in app]



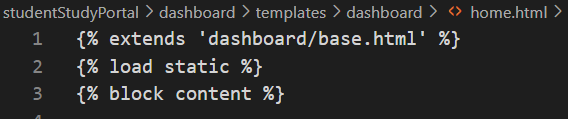
* Html file organize

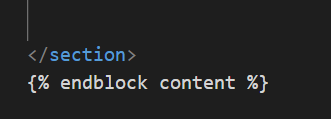
1. {% load static %}
2. <link rel="stylesheet" href="{% static 'css/main.css' %}" />
3. Main content binding with header and nav



4. home.html [only content without header nav]

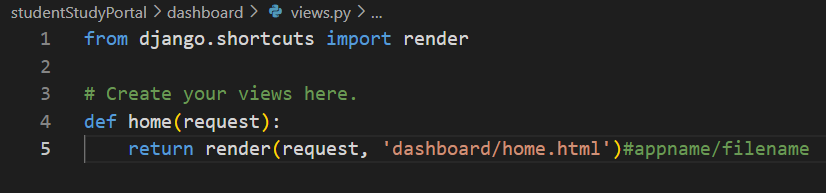
5. extends header file





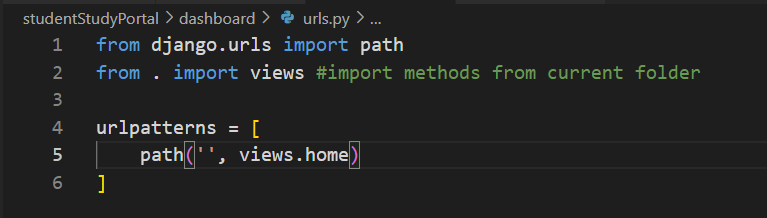
## 2.1. Display Static Files and Url settings

* Create methods in views.py [app folder]



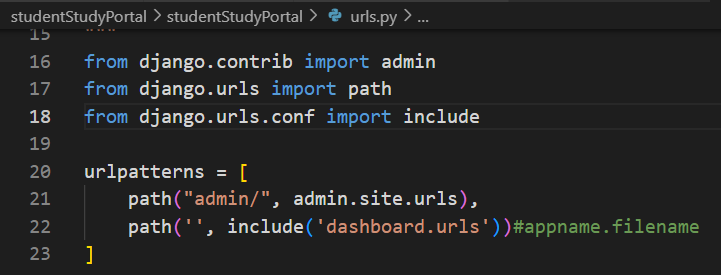
* Create urls.py in dashboard [app folder]

1. from django.urls import path
2. from . import views [import defined method]



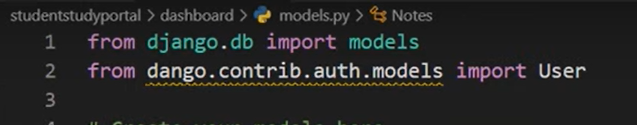
* Include this urls.py into project urls.py

1. path('', include('dashboard.urls'))#appname.filename
2. from django.urls.conf import include [import]



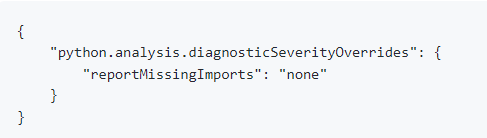
## 2.3. Report Missing Imports

* Report Missing Imports: warning in yellow



Solution: Go Files>preferences>Settings>Extension>Edit in Json

Reference: <https://github.com/microsoft/pylance-release/issues/52>

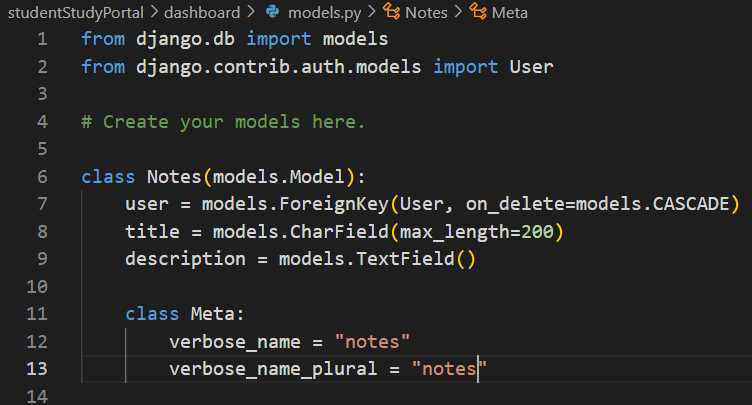


# 3. Models Settings [Notes Model]

## 3.1. Create Model and add data in table

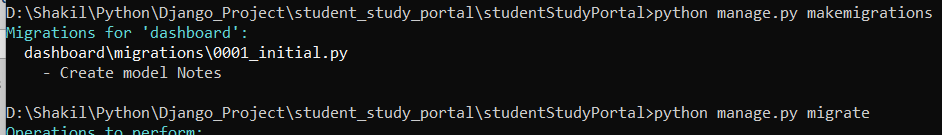
* Create Models[notes]:

1. from django.contrib.auth.models import User
2. On\_delete=models.cascade [if parent table user is removed then child table user is automatically removed]
3. Verbose\_name\_plural is used for admin section “notess” remove extra s



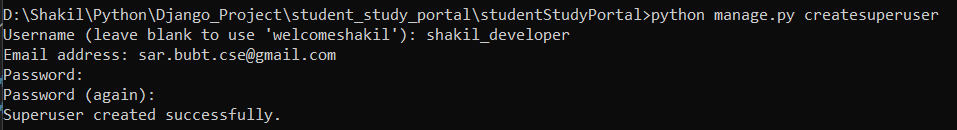
* Migrations:

1. Python manage.py makemigrations
2. Python manage.py migrate

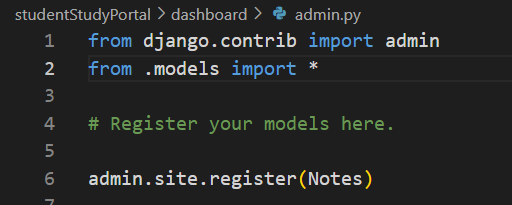


* Create super user:

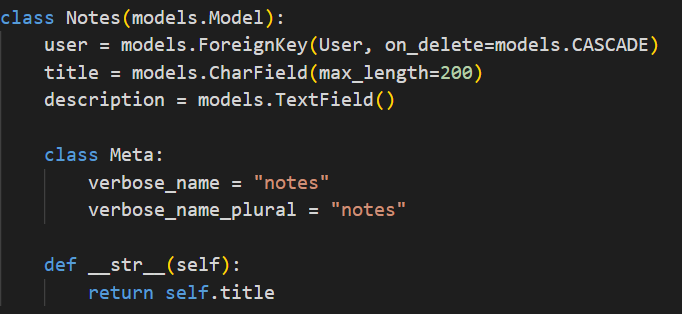
1. Python manage.py createsuperuser
2. Shakil\_developer pass: dev665580



* Register Model in admin section [admin.py]

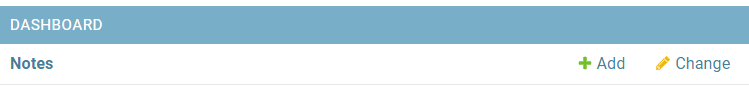
1. Import \* or model class [Notes]

* Class meta for model[Notes]



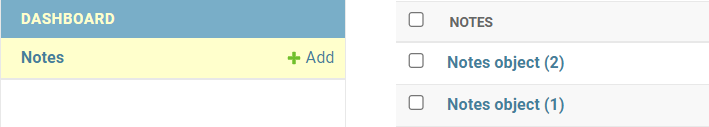


After verbose\_name\_plural

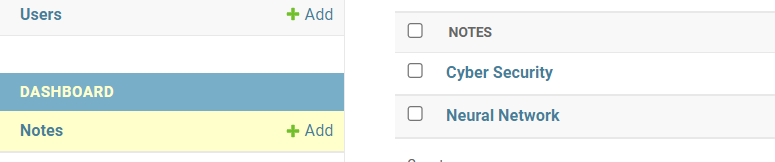


* Def \_\_str\_\_(self) for model[Notes]

1. Before adding



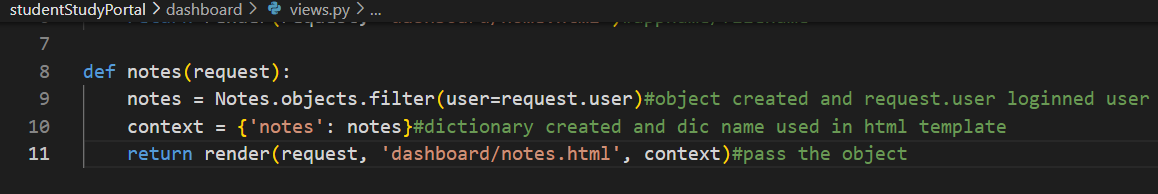
1. After adding return title



## 3.2. Show data from Model [Table]

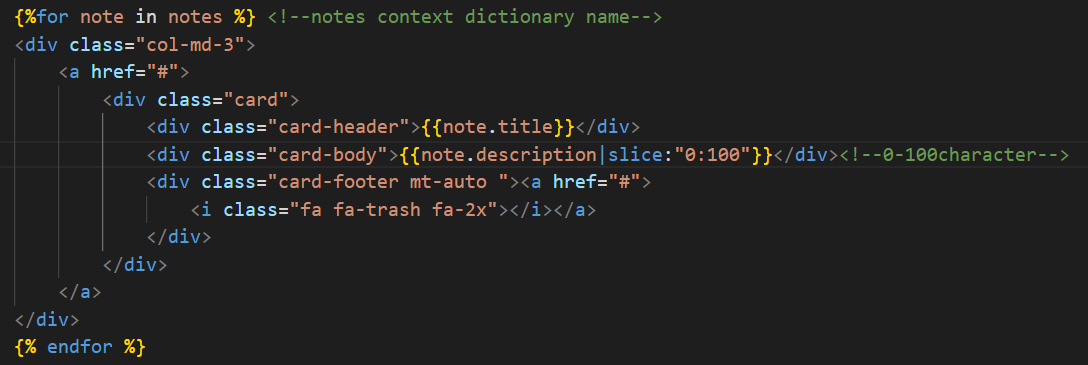
* Import model and created context in views.py

1. from .models import Notes



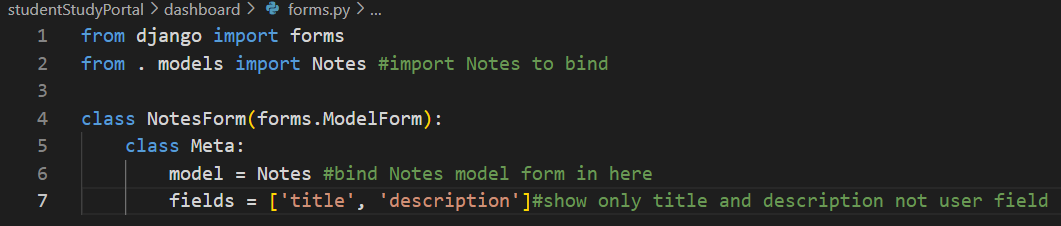
* Print in html template

1. {%for note in notes%}……..{%endfor%}
2. {{note.title}}
3. {{note.description|slice:”0:100”}} [show 100 character]

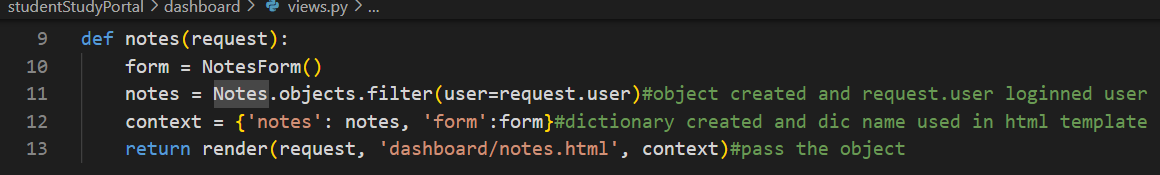


## 3.3. Models table bind with crispy form

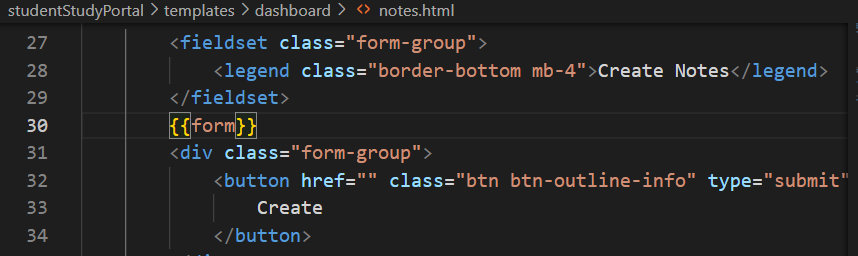
1. Create forms.py in app [dashboard]
2. Use title and description field from created model Notes table and bind with form



3. import form in views.py



4. show in template using dictionary name {{form}}



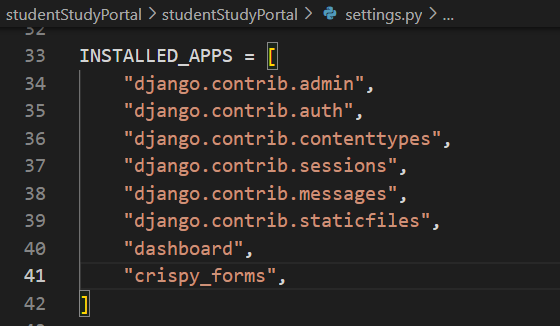
## 3.4. Installing Crispy Form app to beautify Form

* Implementing bootstrap into the form

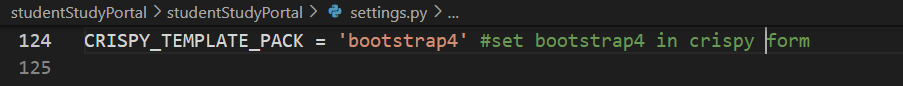
1. Pip install Django-crispy-forms [make sure venv activated]

Reference - <https://django-crispy-forms.readthedocs.io/en/latest/>

1. Put in project settings.py

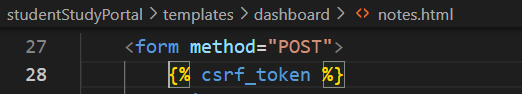


1. Put bootstrap4 in settings.py



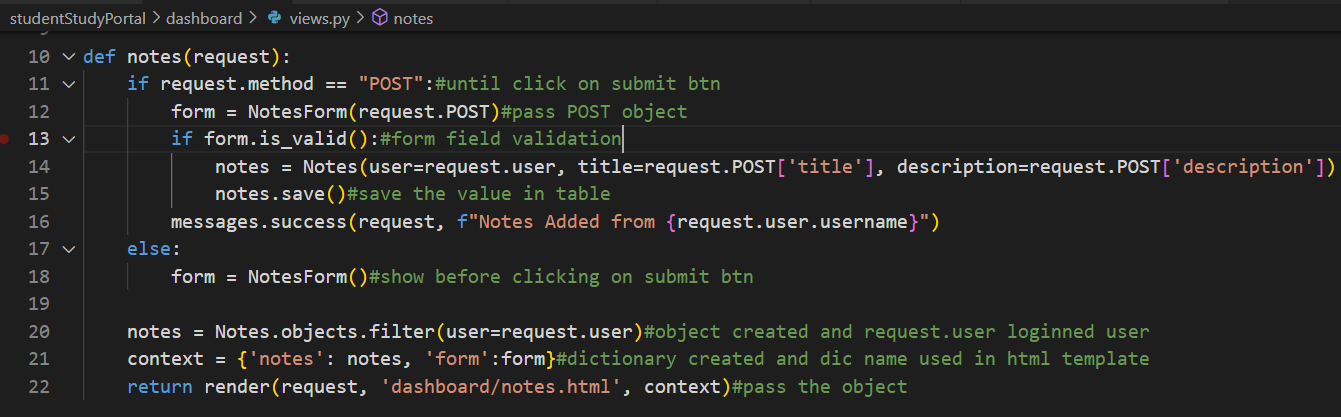
## 3.5. Save Form Data into Database

* Csrf\_token in form [html file]



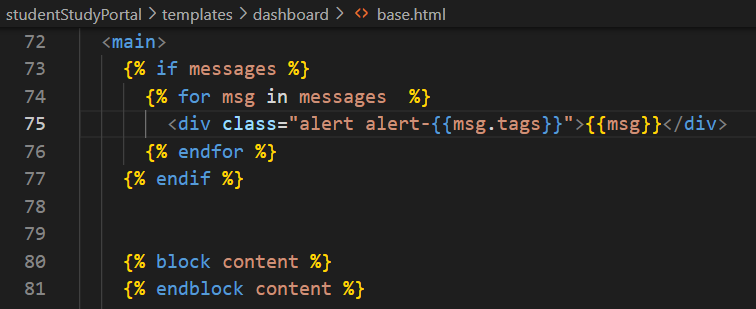
* Implement methods in views.py

1. from django.contrib import messages [show messages]
2. Logic: pass post object with crispy form when clicked on the submit button otherwise show only crispy form and show the objects in html through context



* Show messages in template file

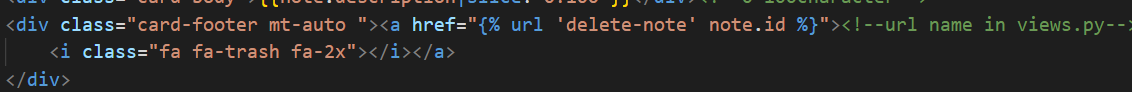
1. Show the msg in master file [base.html]
2. Msg.tags dynamically show success, warning, etc



## 3.6. Delete Data from Html Template and database

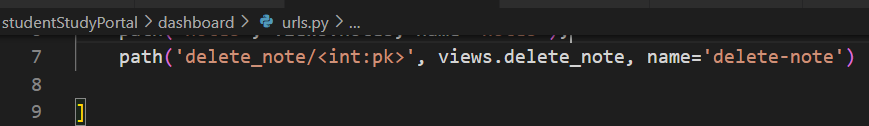
* Set urls in html template

1. href="{% url 'delete-note' note.id %}" [delete-note is url name from views.py url and pass id of user]



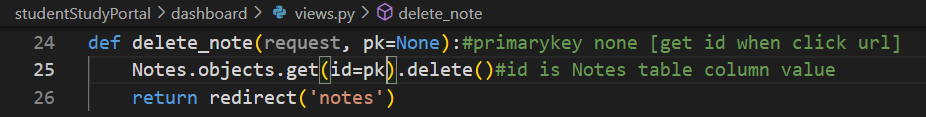
* set urls in urls.py

1. int:pk [name = primary key] variable will be pass in the function [pk is note.id]



* method in views.py

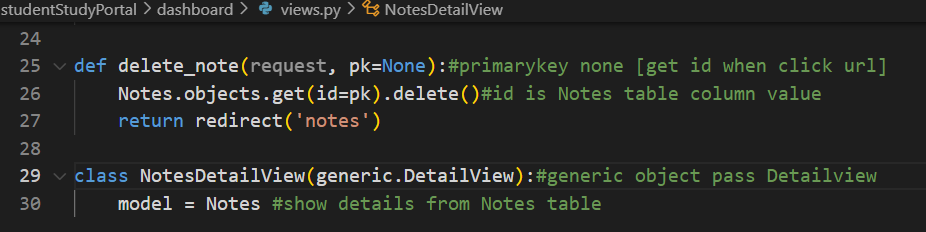
1. from django.shortcuts import render, redirect[import redirect]



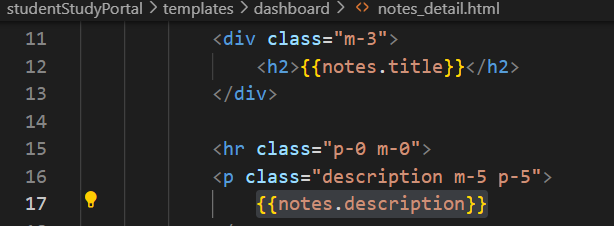
## 3.7. Generic DetialView of item

* create class in views.py

1. from django.views import generic
2. define class



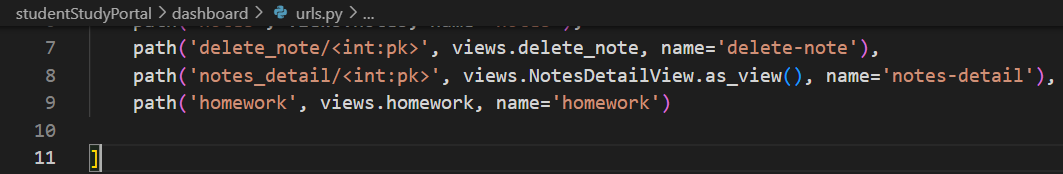
1. copy paste notes\_detail.html file to templates
2. set {{notes.title}} and {{notes.description}} [notes is context dict name]



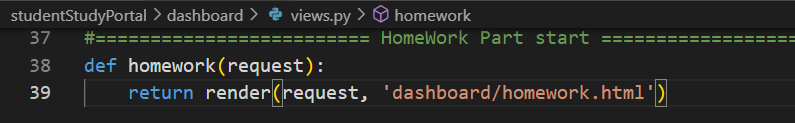
# 4. Model Settings [HomeWork Model]

## 4.1. Create Homework Model & add data in table

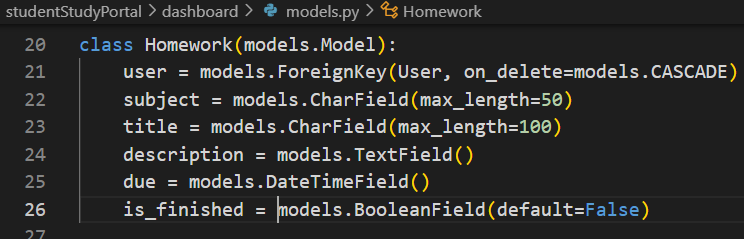
* set path in urls.py



* define method homework



* Create Model in models.py

1. On\_delete CASCADE is if delete parent table data then child table also automatically delete. booleanField default unchecked

* Migrations:

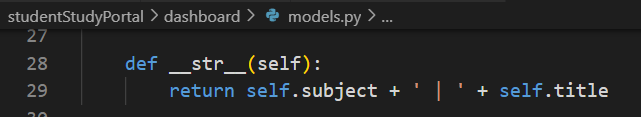
1. Python manage.py makemigrations
2. Python manage.py migrate

* Register Model in admin.py



* Def \_\_str\_\_(self):

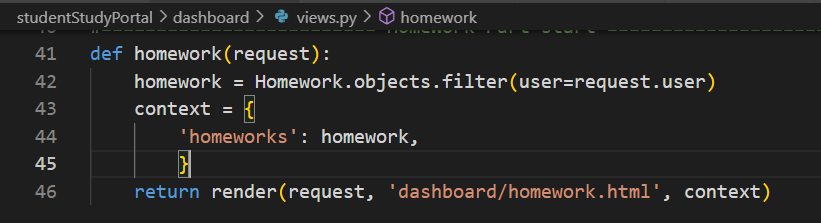
1. Add data into homework table
2. Return subject | title



## 4.2. Show Data from Model or Table

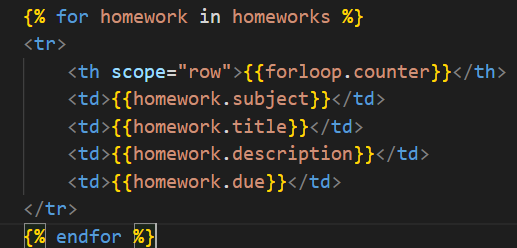
* Edit on def homework in views.py

1. Create object of model homework
2. Pass the object through context dictionary key

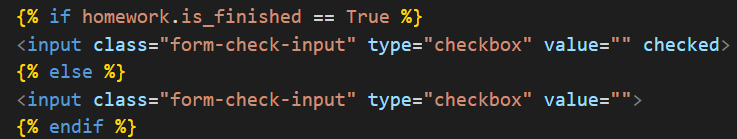


* Update in homework.html

1. {{forloop.counter}} [used counter 1 to 1++]

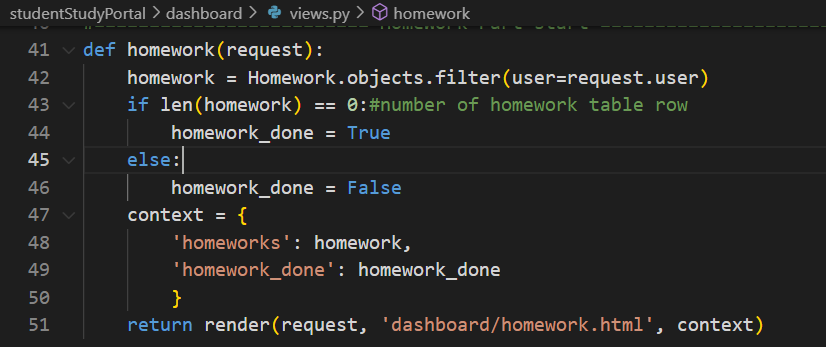


1. Checkbox condition [is\_finished is a Boolean value]

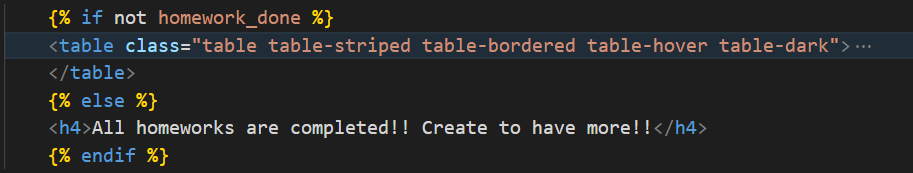


* All homeworks are completed!! Create to have more!!

1. Logic is if all homeworks completed [no row in table] show the msg without showing table of homework and if not completed show the table which are completed or not
2. Len(homework) is the row number of table if no data that means homework\_done is true



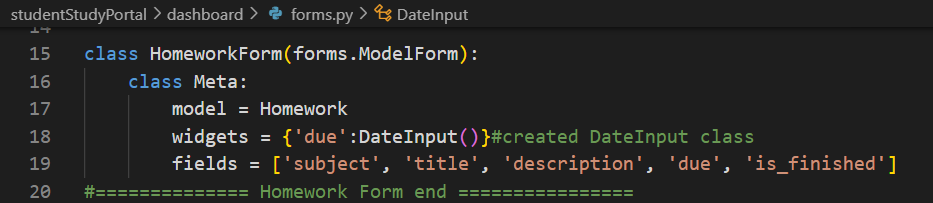
1. Show in html [not homework\_done means homework\_done is no value or not empty]



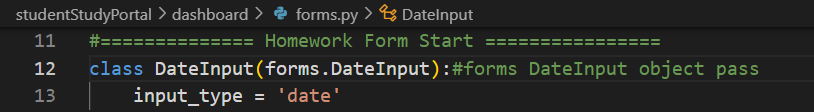
## 4.3. Model table bind with created crispy form

* Created Form in forms.py

1. Form created from model Homework fields
2. Widget new class call

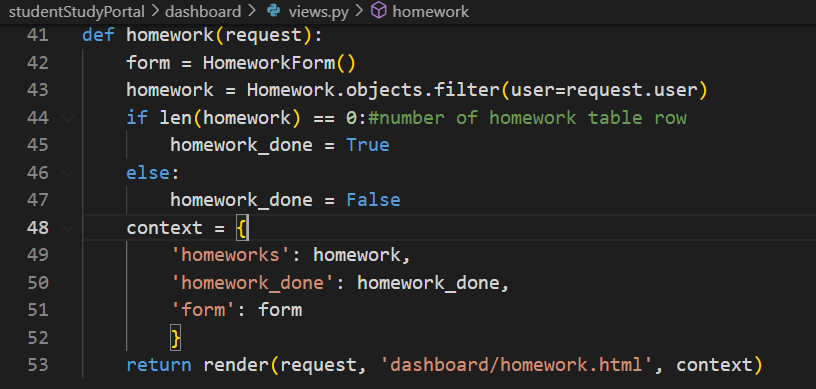


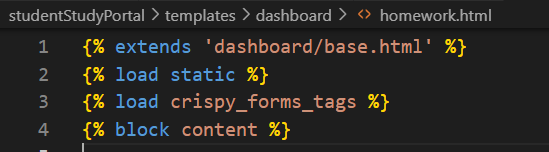
1. Create new class DateInput for widget

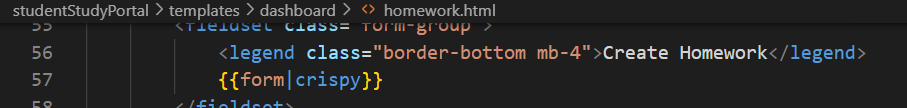


* Call The object in views.py

1. form = HomeworkForm()
2. Pass the object through context for showing in html
3. Call the form in html by {{form|crispy}} [load {% load crispy\_forms\_tags %}]



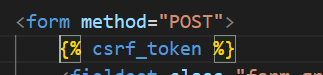




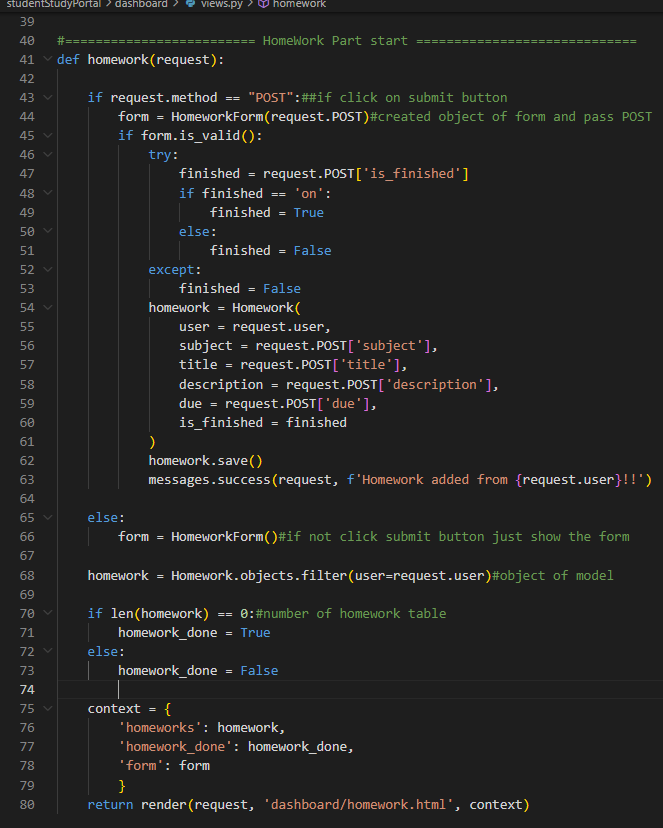
## 4.4. Save Form Data into database

* Update homework method in views.py

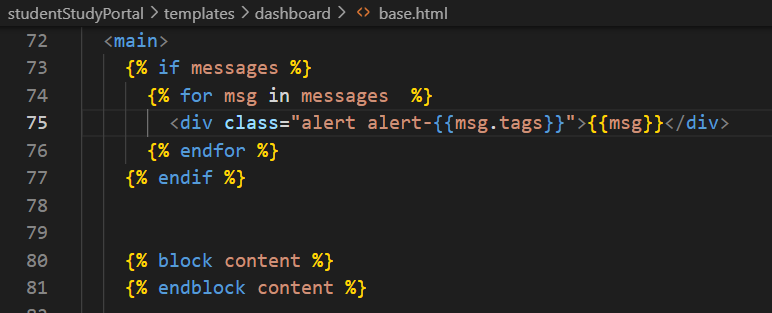
1. Put csrf\_token below form



1. Logic is when submit form create the object of form and pass POST object otherwise create only object of the form
2. Checkbox on/off try except [execute try block if get error then execute except block]
3. Homework\_done is for showing msg if there is no data in the table and show msg all homeworks is completed



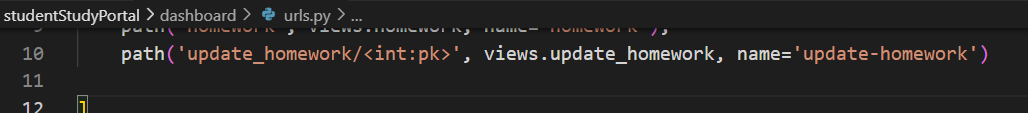
1. Show msg of saved data in base.html [master template]



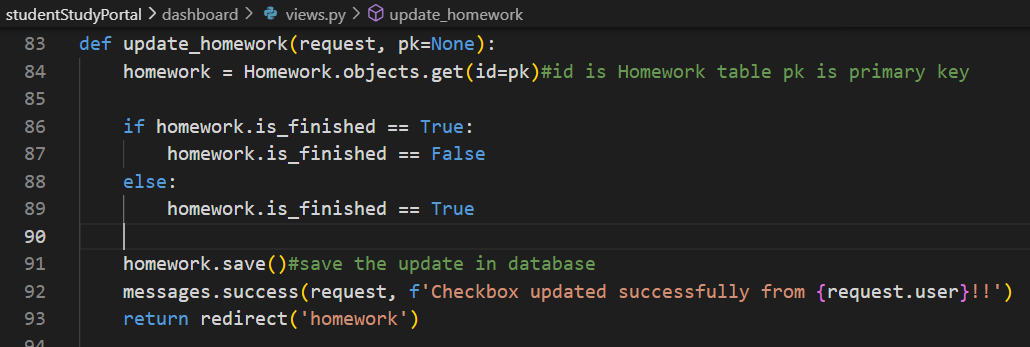
## 4.5. Update checkbox checked or unchecked

* Update

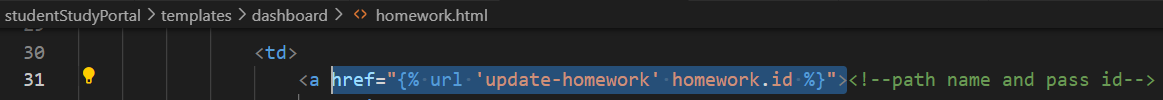
1. Update path in urls.py



1. Define method in views.py

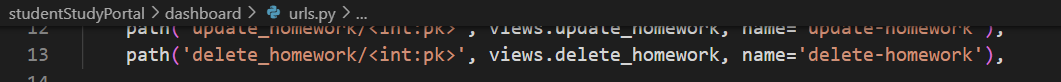


1. href="{% url 'update-homework' homework.id %}">

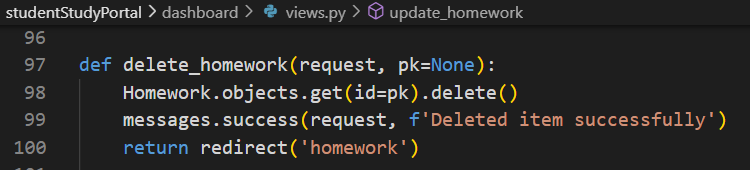


## 4.6. Delete item from model

* update path in views.py



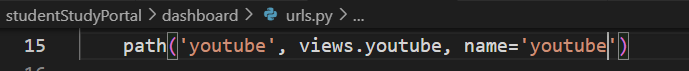
* create delete\_homework method in views.py



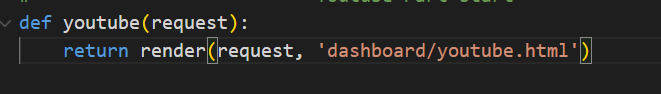
# 5. Common Search Form Settings

## 5.1. Youtube Search Form API

* Path Create in urls.py

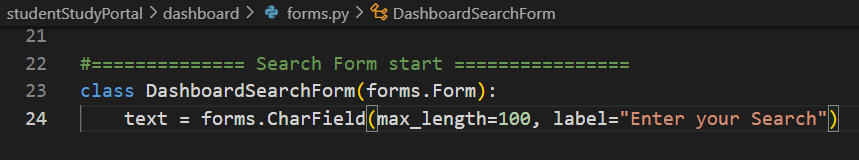


* Define method youtube in views.py

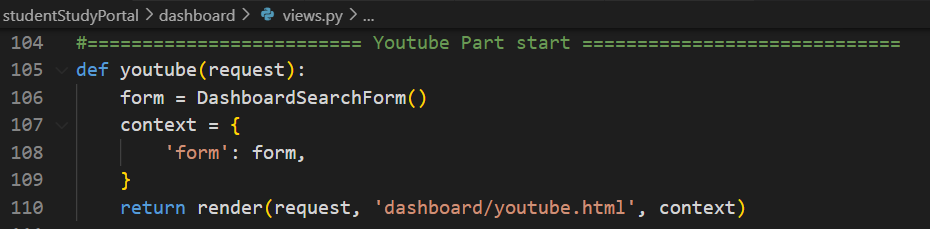


* Copy paste the html file in templates/dashboard folder
* Form creation in forms.py [create forms.py in app]

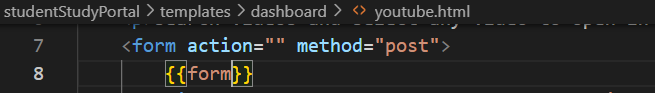
1. Common class for youtube search, Wikipedia search
2. Text can be any name for forms charfield



1. Update method in views.py[from . forms import DashboardSearchForm]



1. Update youtube.html



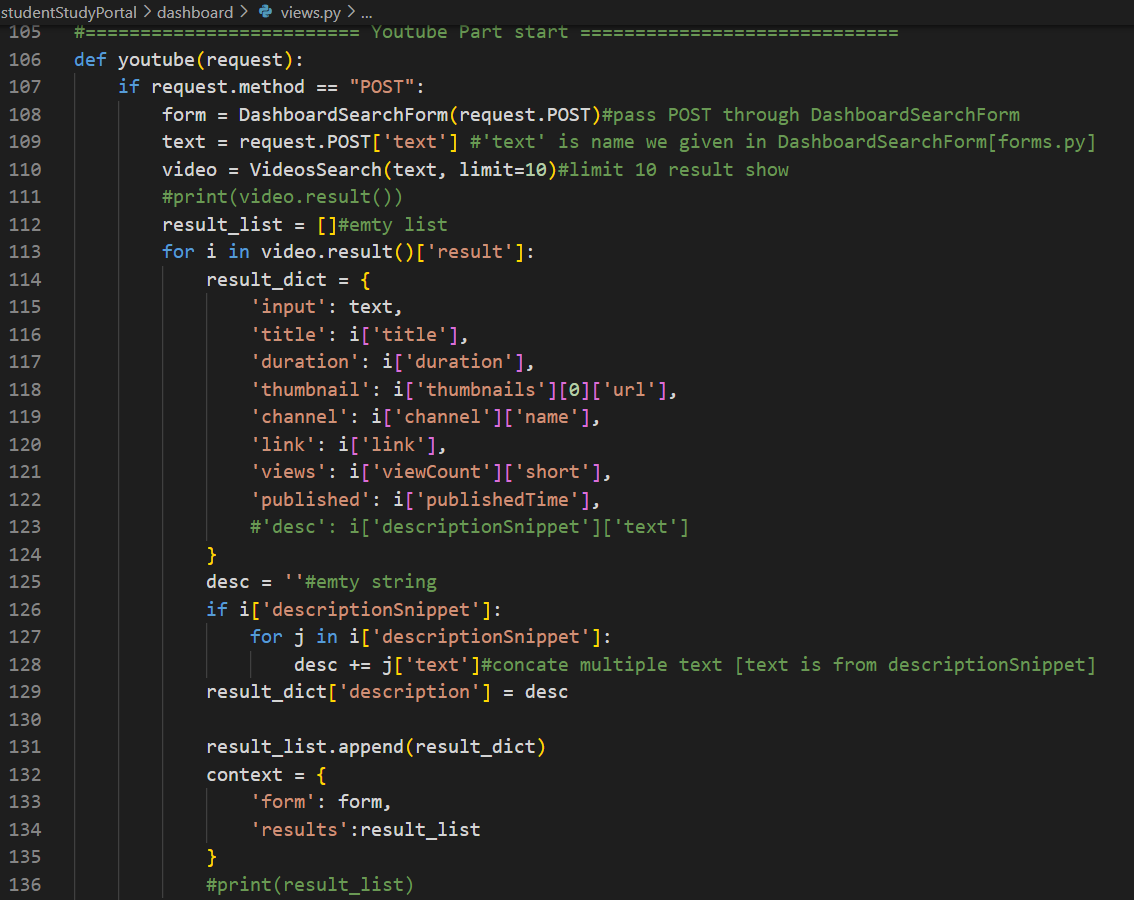
### 5.1.1. Install youtube-search-python api library

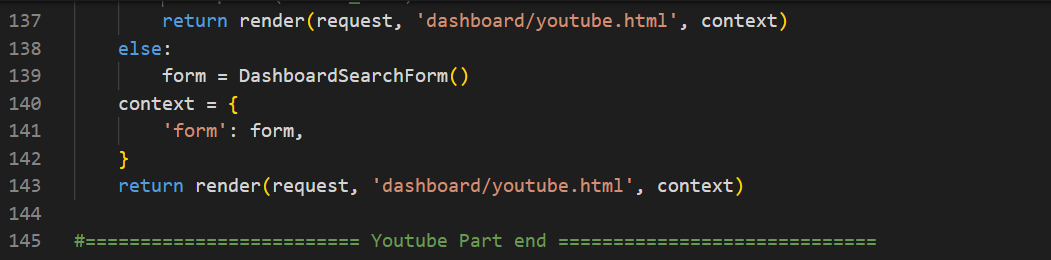
1. csrf\_token in form in youtube.html [befor submit search btn]
2. Before install packages, select venv directory and activate venb
3. pip install youtube-search-python [install library]

reference: <https://pypi.org/project/youtube-search-python/>



1. import in views.py [from youtubesearchpython import VideosSearch]
2. update method youtube in views.py
3. all api data in [video.result()['result']]
4. collect data using loop and set the data in a dictionary
5. for description index of dictionary, concate multiple text in an empty string,
6. return all value outside the loop through context





1. shows in youtube.html

{{result.link}} = [youtube link]

{{result.title}}

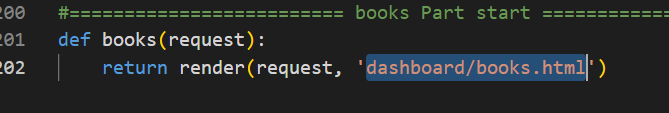
{{result.thumbnail}} etc tag in html file

## 5.2. Requests API to access any url to get data

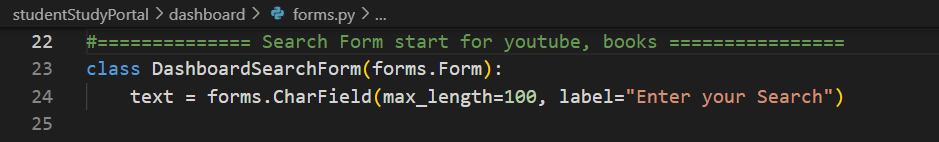
* Path created in urls.py



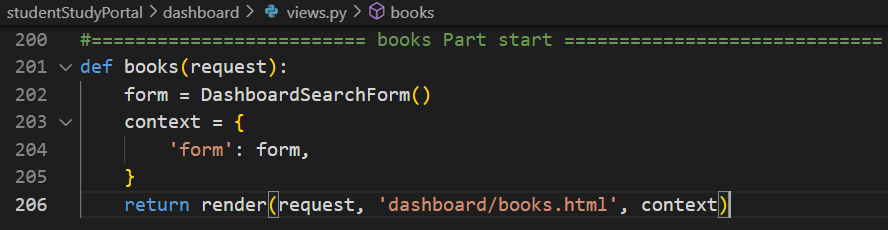
* Books method created in views.py



* Copy the books.html file into template\dashboard folder
* Create forms.py file in app folder [dashboard]
* Create a common form of class DashboardSearchForm for youtube, books, Wikipedia [already created]



* Call the form object in views.py [from . forms import DashboardSearchForm]



* Set {{form}} in books.html file

### 5.2.1. install requests API [ pip install requests ]

References:

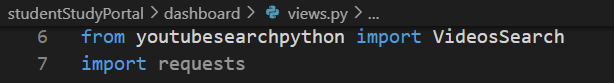
<https://pypi.org/project/requests/>

<https://developers.google.com/books/docs/v1/using>

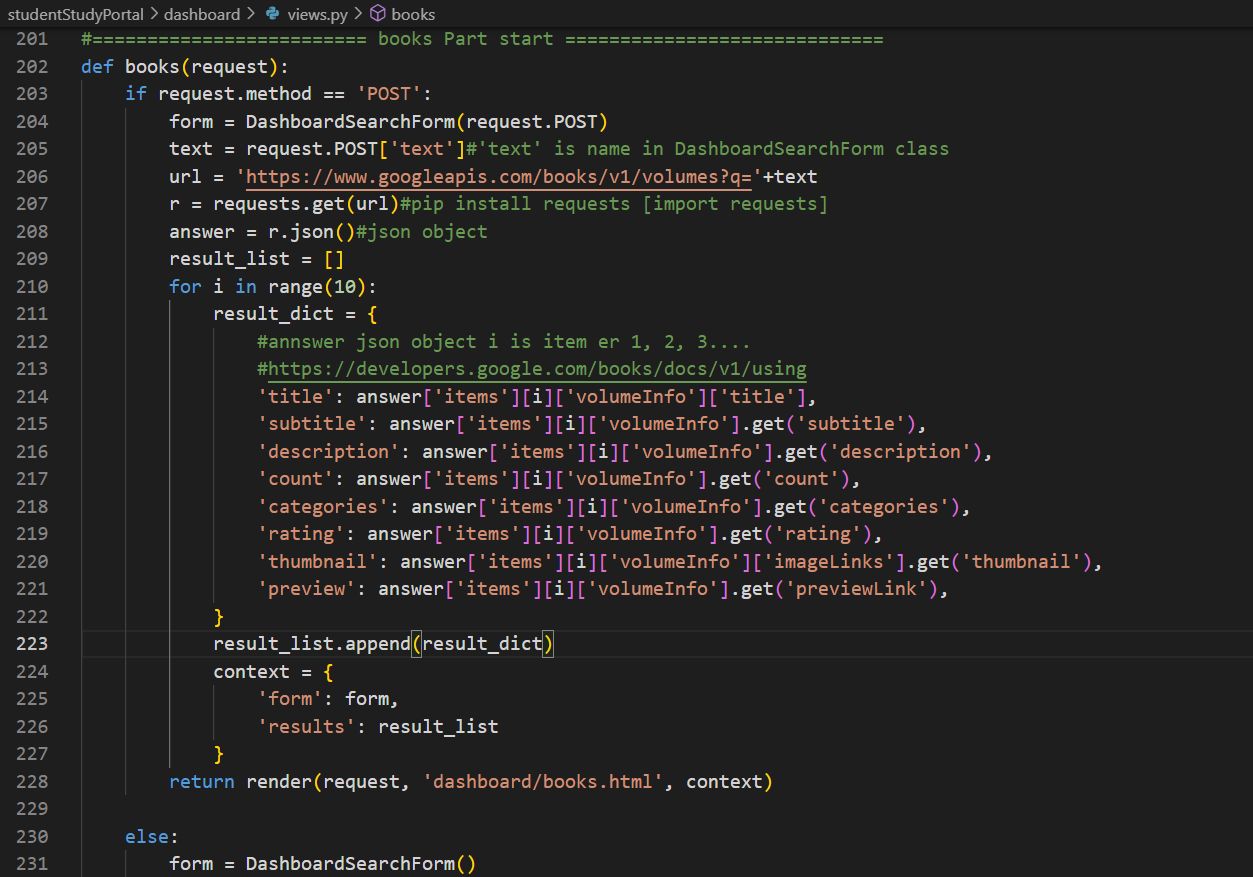
* Install in the root folder [make sure virtual environment is activated]



* Import in views.py [import requests]

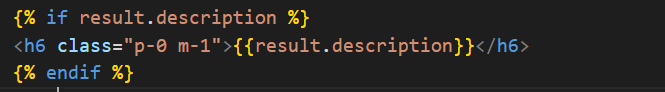


* {% csrf\_token %} set in books.html form
* Update books method in views.py
* Answer is object from rquests api [items from google books api]
* Multiple items that’s why ‘i’ count first item, second item
* We can access by indexing or get function of requests api





* Update in books.html

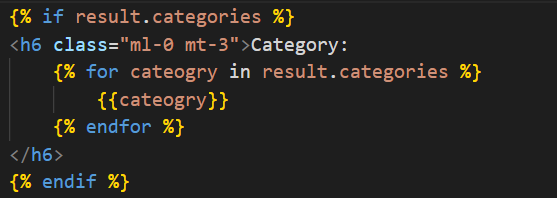




* Image src path



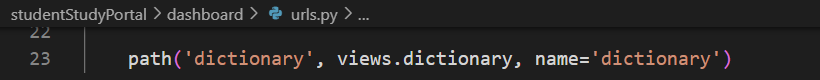
* Categories forloop to show category individually



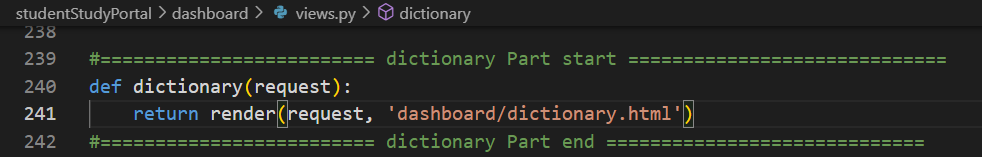
## 5.3. Dictionary API

### 5.3.1. Path, method, template

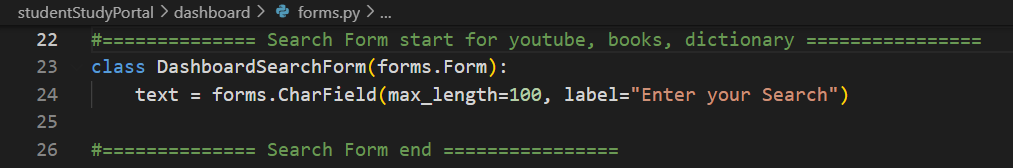
* Create path in urls.py

]

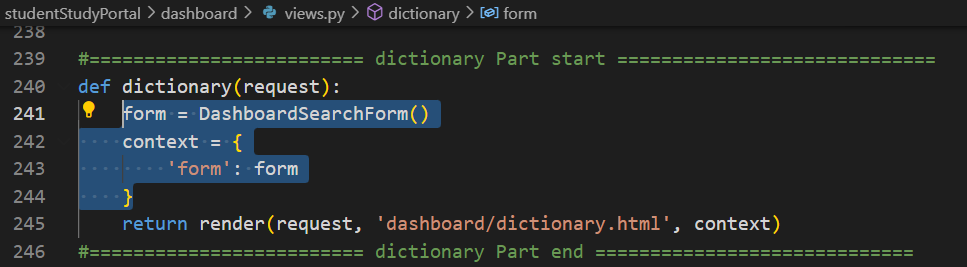
* Create dictionary method in views.py



* Copy dictionary.html and paste into templates/dashboard folder
* Create forms.py in dashboard[app folder]
* Create class DashboardSearchForm [from django import forms]



* Update dictionary method in views.py



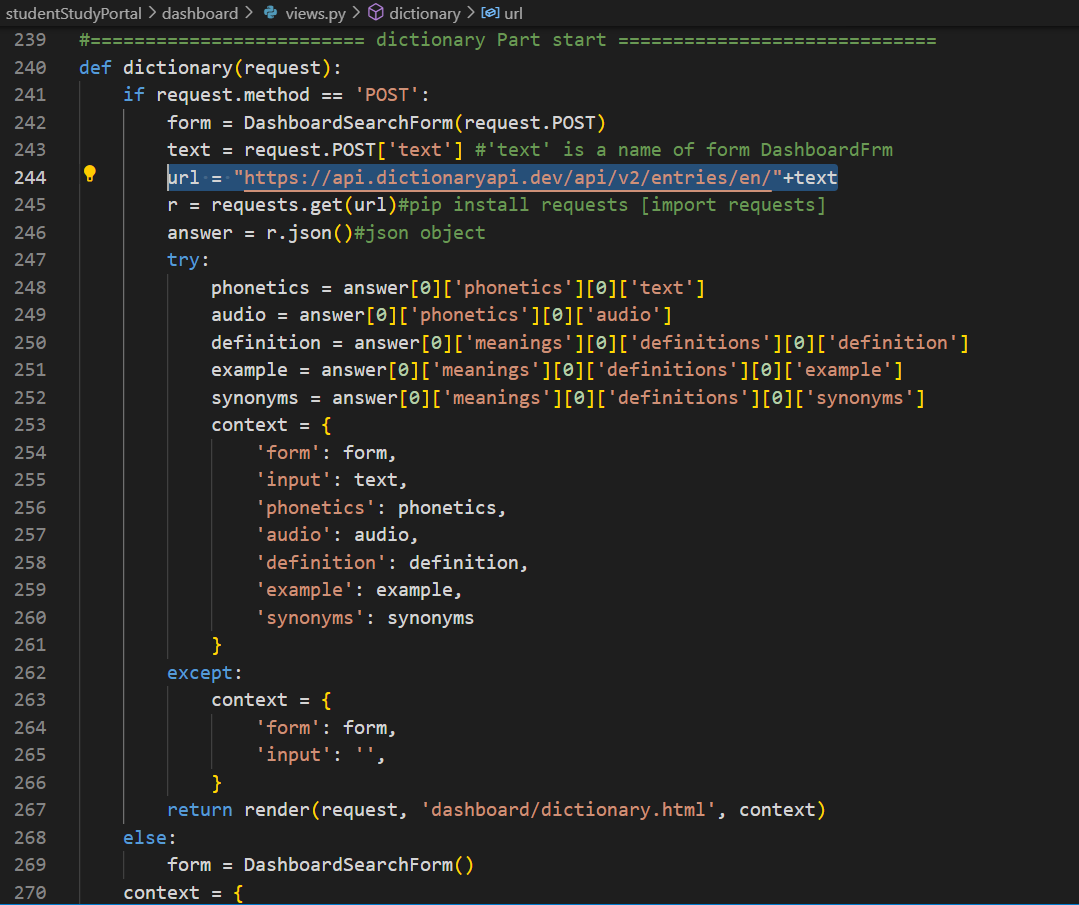
* {{form}} set the form in dictionary.html [after form tag]

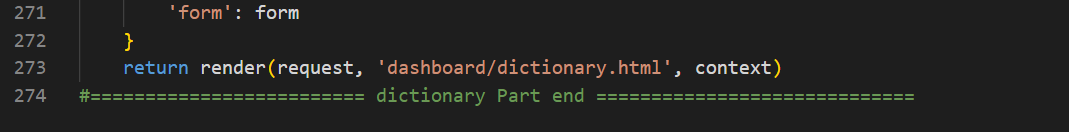
### 5.3.2. Implement Dictionary API

* References:

<https://dictionaryapi.dev/>

* Csrf\_token for submit the form
* No need to install library
* url = "https://api.dictionaryapi.dev/api/v2/entries/en/"+text
* r = requests.get(url)#pip install requests [import requests]
* answer = r.json()#json object
* update dictionary method in views.py





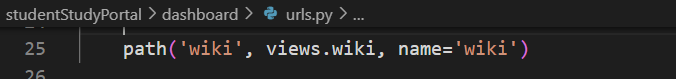
* update in dictionary.html



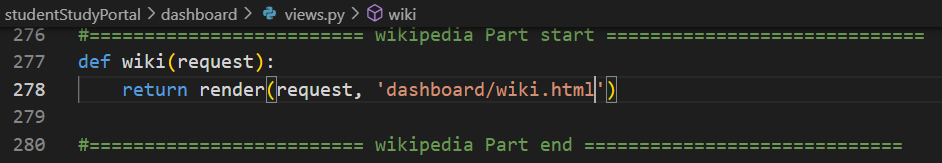
## 5.4. Wikipedia API

### 5.4.1. Path, method, template

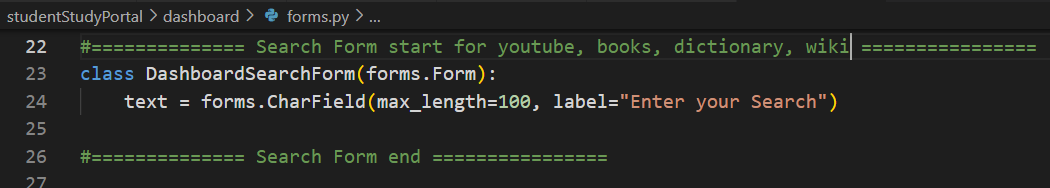
* path created in urls.py



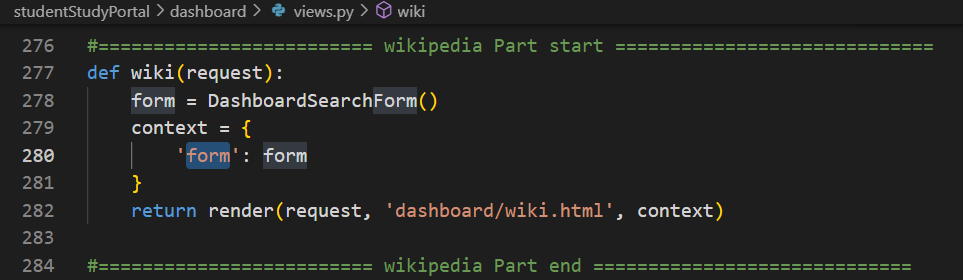
* wiki method created in views.py



* Copy wiki.html and paste into templates/dashboard folder
* Create forms.py in dashboard[app folder]
* Create class DashboardSearchForm [from django import forms]



* Update wiki method in views.py



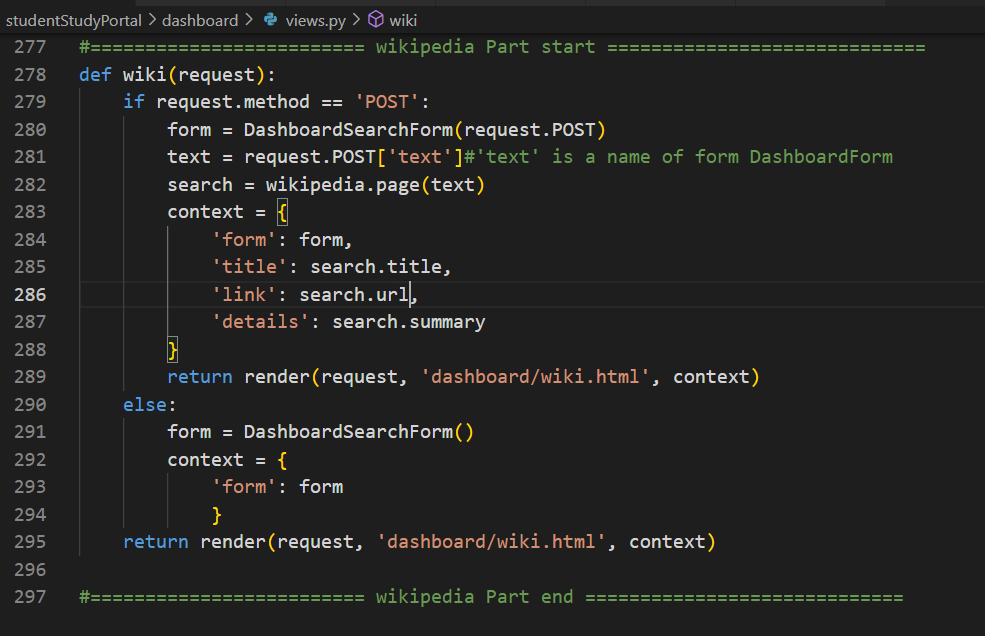
* {{form}} set the form in dictionary.html [after form tag]

### 5.4.2. Implement wiki API

* References:

<https://pypi.org/project/wikipedia/>

* Csrf\_token for submit the form
* pip install Wikipedia [import wikipedia]
* update wiki method in views.py

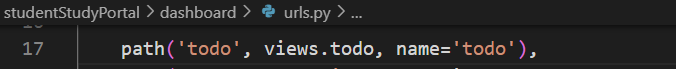


* update wiki.html
* {{title}} {{link}} {{details}}

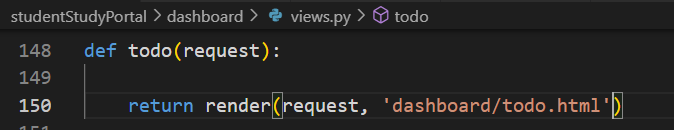
# 6. Model Settings [Todo Model similar homework]

## 6.1. Path and method Call

* Create path in urls.py



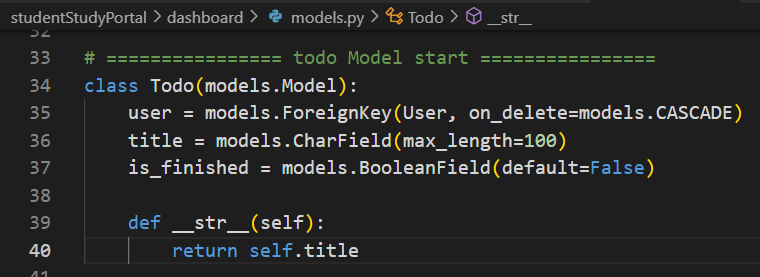
* Create method todo in views.py



## 6.2. Create Todo Model and add data in table

* Create Todo in models.py

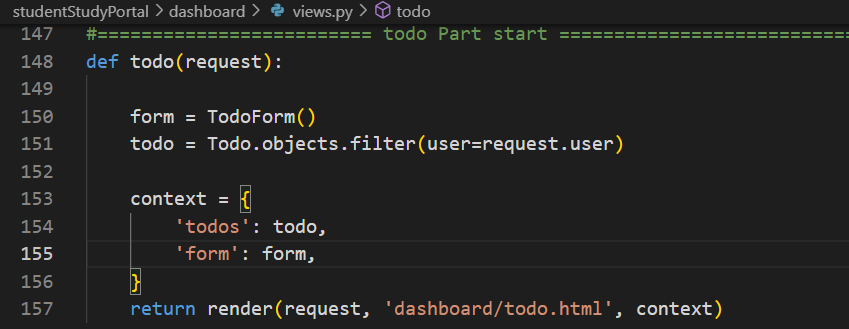
1. from django.db import models
2. from django.contrib.auth.models import User
3. Return self.title for showing title in admin panel



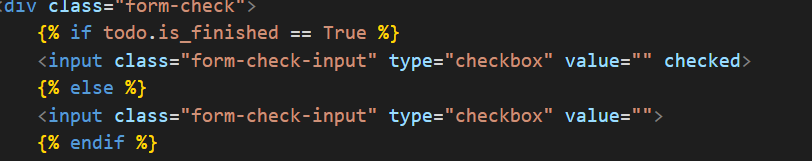
## 6.3. Showing data in html file

* Update todo method in views.py

1. Import and create object of the model and pass through context



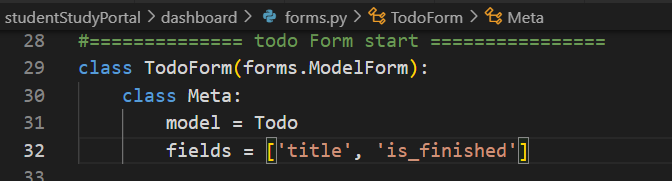
1. Counter [{{forloop.counter}}] and title is {{todo.title}}
2. Checkbox Condition in html



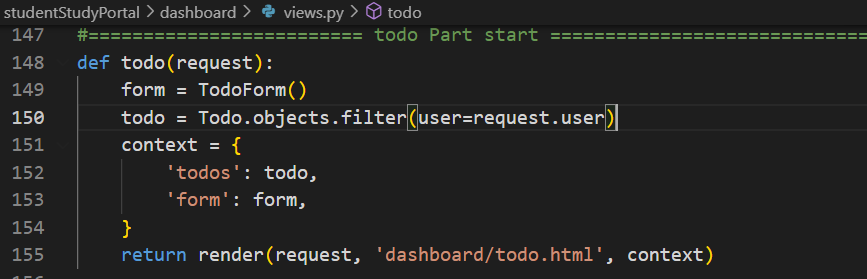
## 6.4. Create Form binding model in forms.py

* Create Form

1. Create file forms.py in app director
2. Import the model Todo
3. from django import forms [import and pass object ModelForm]
4. Create form with the fields I want to show



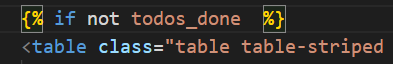
1. Update in views.py and import TodoForm object and pass through context

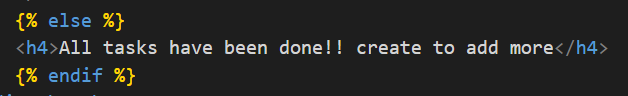


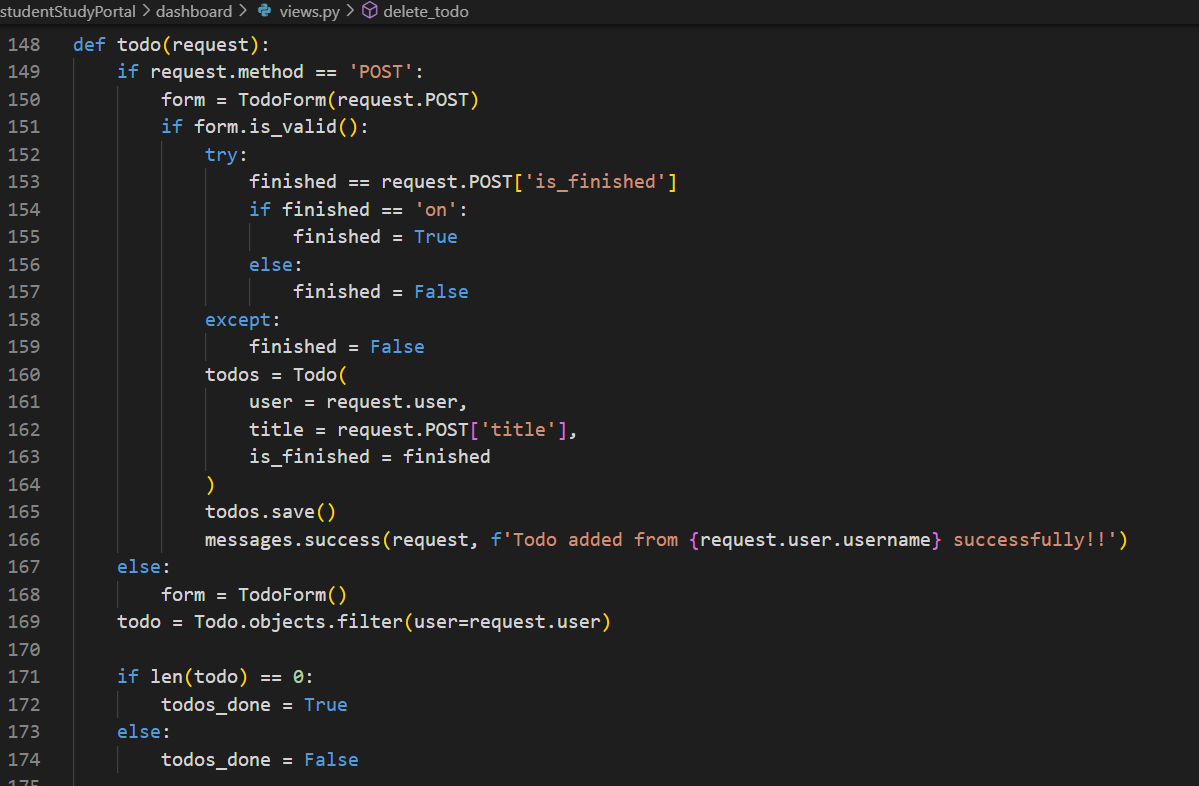
1. In html file {% load crispy\_forms\_tags %} [load in header]
2. {{form|crispy}}

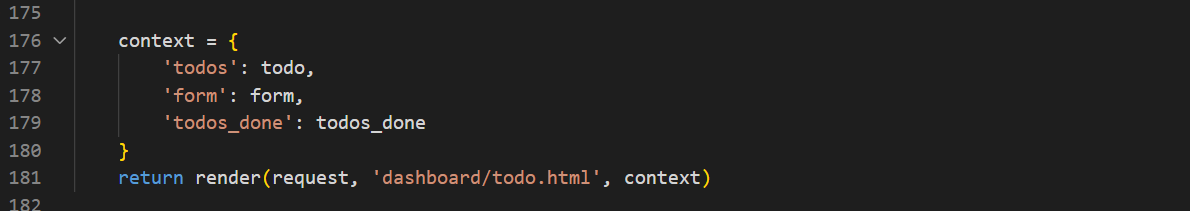
## 6.5. Save submitted form data into table

* Csrf\_token in html form
* Update todo method in views.py
* If is\_finished checkbox checked that mean it is on otherwise off
* Checkbox on/off try except [execute try block if get error then execute except block]
* Todo\_done is for showing msg if there is no data in the table and show msg all todos is completed otherwise show the table







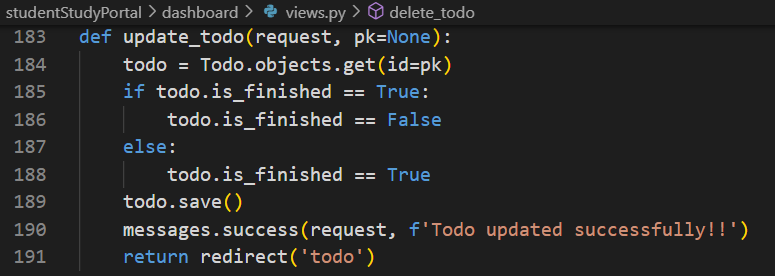


## 6.6. Update checkbox checked or unchecked

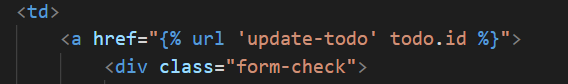
* Path create in urls.py



* Update\_todo Method create in views.py



* url setting in html

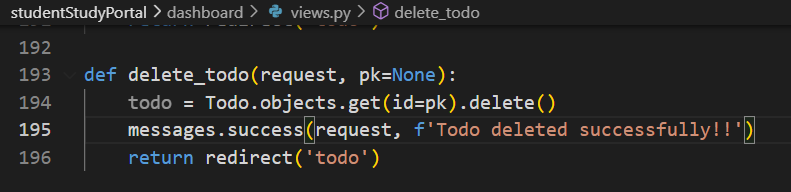


## 6.7. Delete item from database and update in html

* path created in urls.py



* delete\_todo method create in views.py



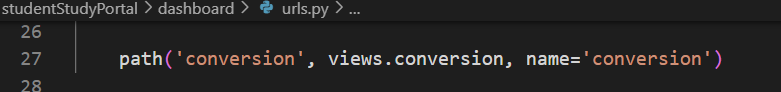
* url setting in html



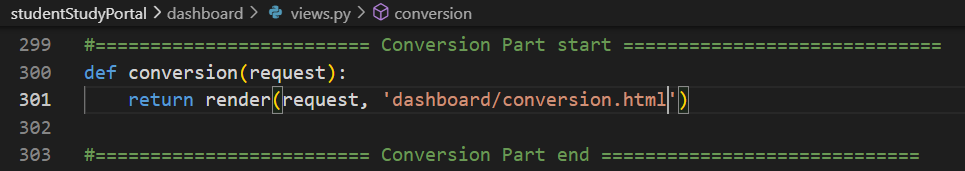
# 7. Conversion pages [class Created from Forms.py]

## 7.1. Path, Method, Template

* path created in urls.py



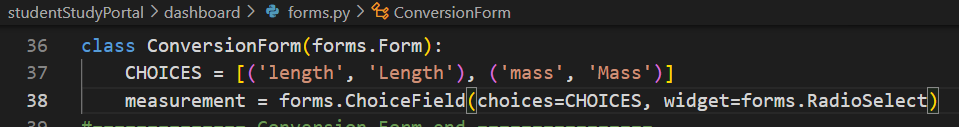
* conversion method created in views.py



## 7.2. Create a radio select class in forms.py

* Create class [forms.py created in app]

1. Two choices
2. Measurement [RadioSelect option with two choices]

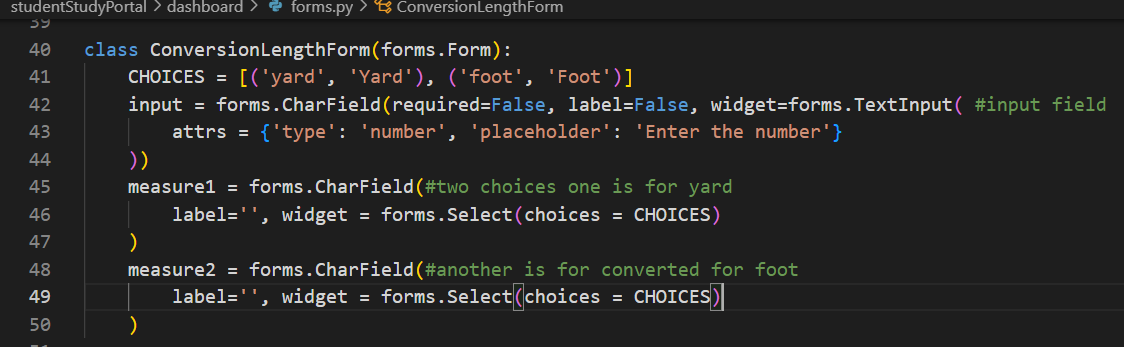


* Update conversion method in views.py

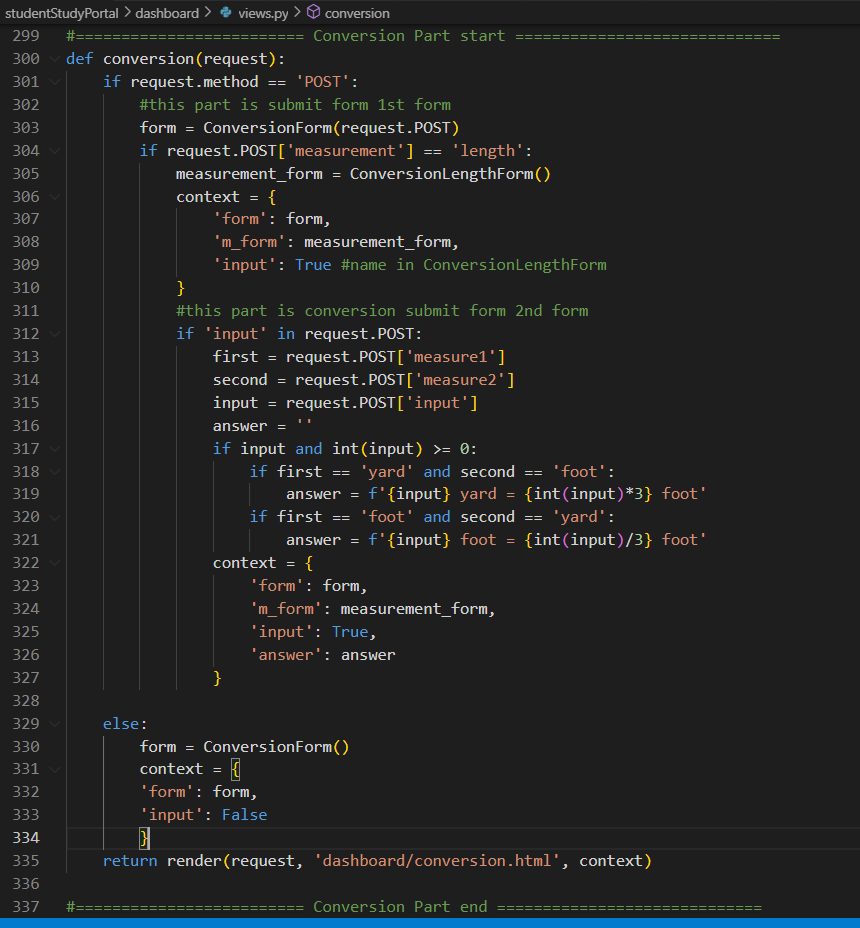


## 7.3. Length Conversion Yard to Foot

* Create ConversionLengthForm class in forms.py

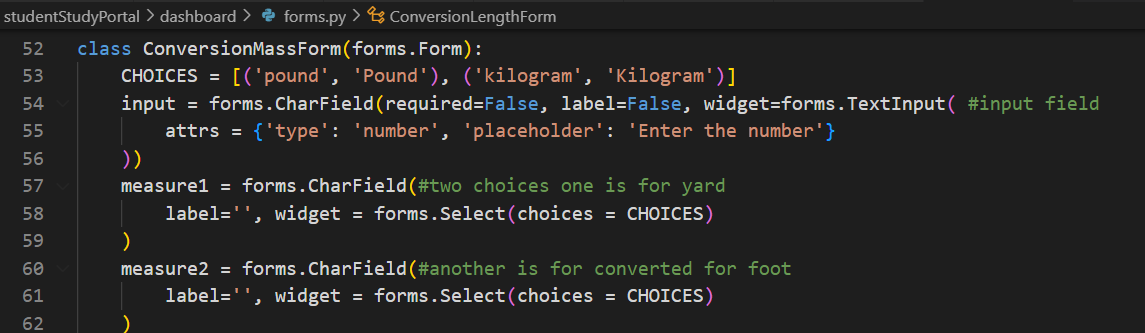


* Update in views.py

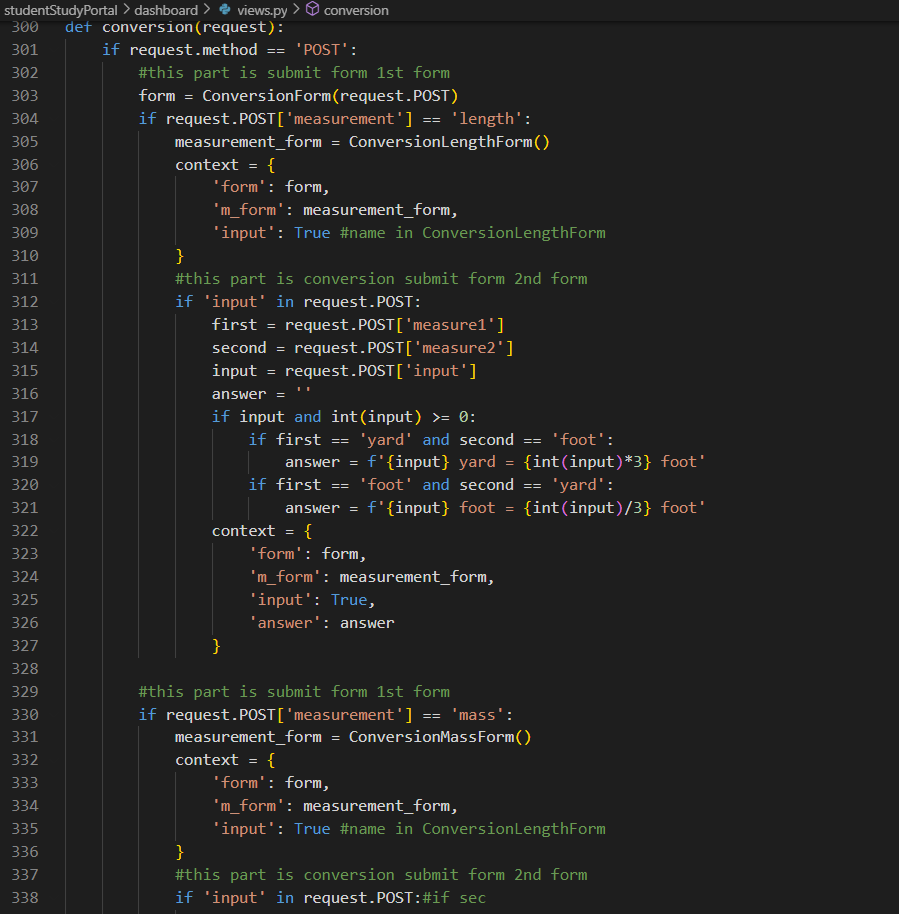


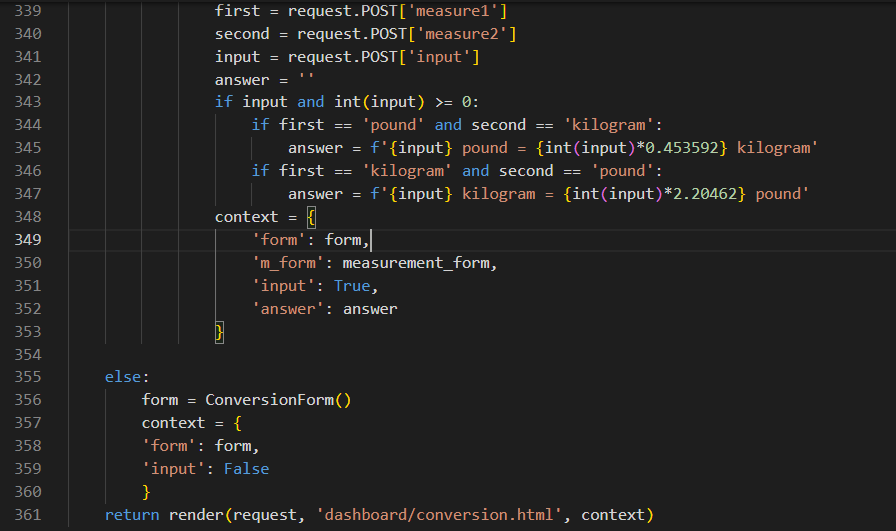
## 7.4. Mass Conversion Kilo to Pound

* Create ConversionMassForm class in forms.py



* Update views.py

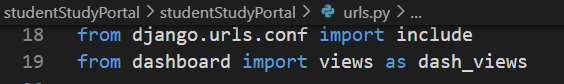




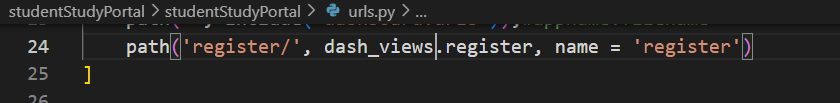
# 8. Registration, Login, Profile, Logout

## 8.1. Registration

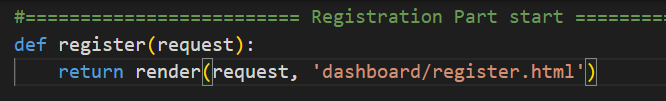
* All registration, logins url in project urls.py not apps url.py
* Import views.py from app folder



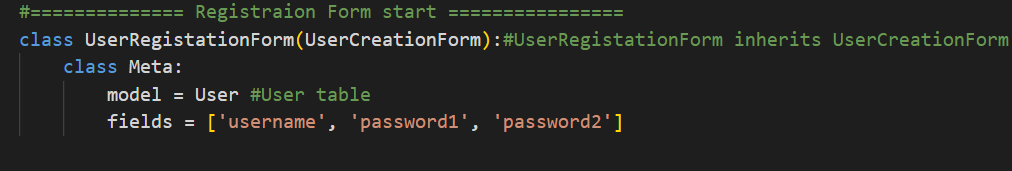
* Path create in urls.py [project url]



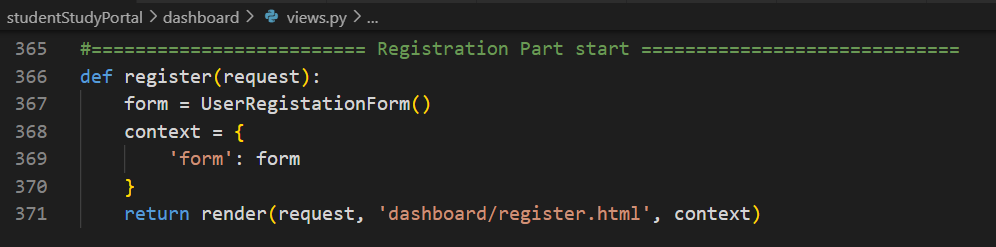
* Register method created in views.py



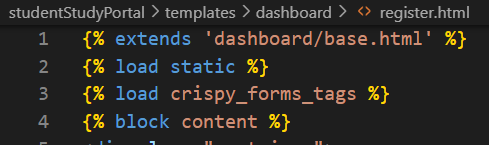
* Created forms.py in app folder
* Import from django.contrib.auth.forms import UserCreationForm [forms.py created]
* Create class UserRegistationForm [from django.contrib.auth.models import User]



* Update in views.py



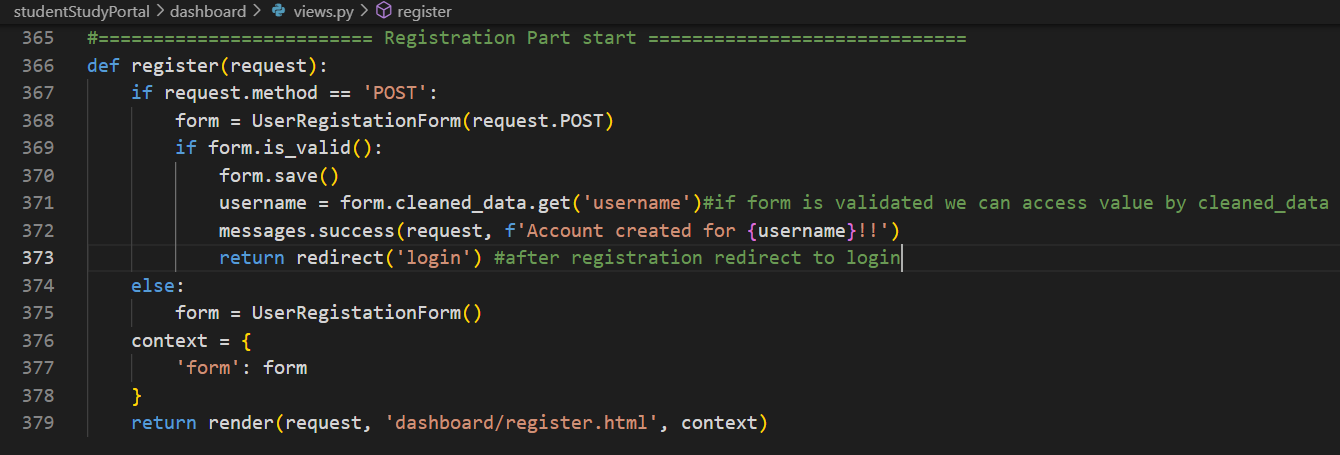
* Update in register.html file





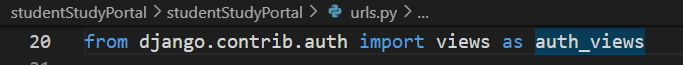
* After Submitted the registration button
* Update in views.py

1. cleaned\_data [Once you’ve created a Form instance with a set of data and validated it, you can access the clean data via its cleaned\_data attribute:]
2. return redirect(‘login’)



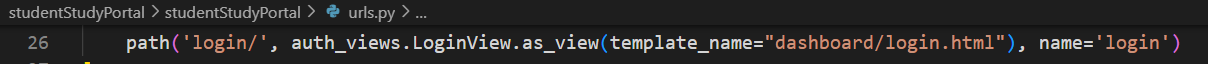
## 8.2. Login

* import in urls.py

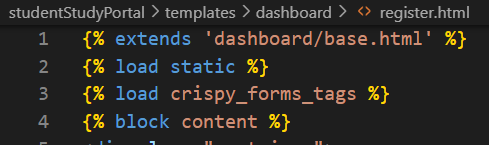


* path created in urls.py

1. LoginView auth\_views login form [as\_view redirected to my template]

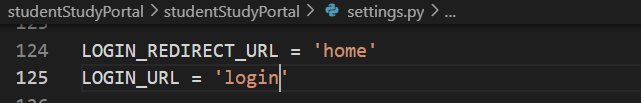


* Update just login.html file





* Update settings.py for login redirect



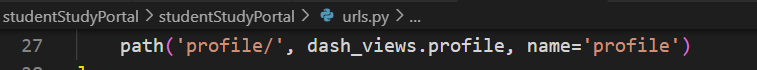
* Login link update



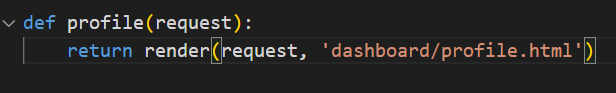


## 8.3. Profile

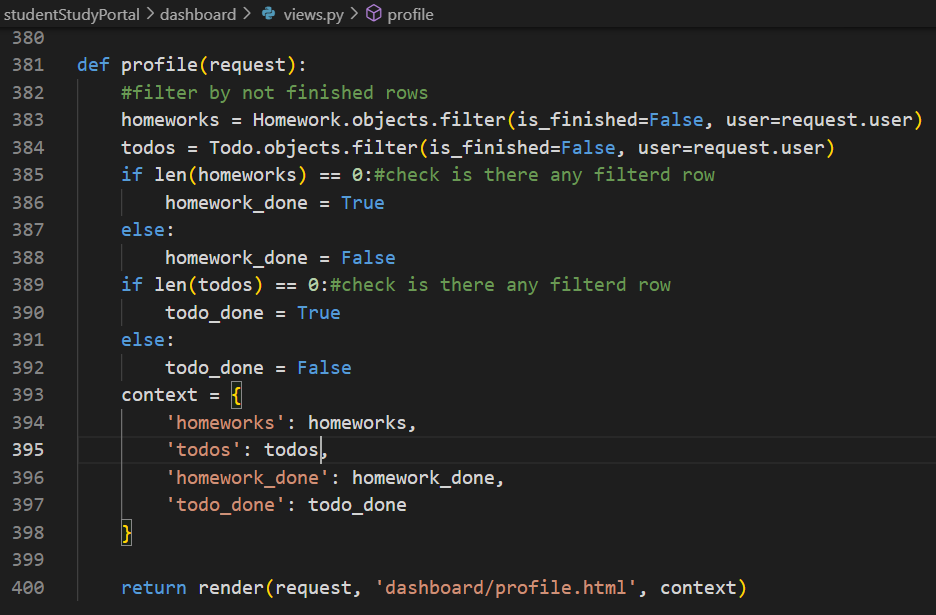
* Create path in project urls.py [from dashboard import views as dash\_views]



* Create profile method in views.py



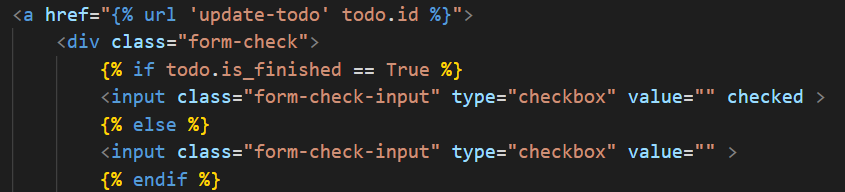
* Filter data from databases



* Update in html file

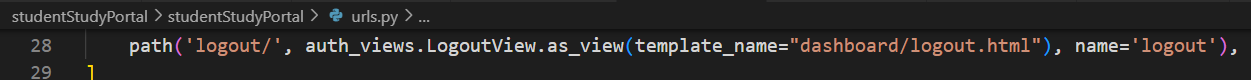




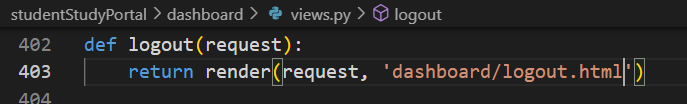


## 8.4. Logout

* Path created in urls.py [from django.contrib.auth import views as auth\_views]

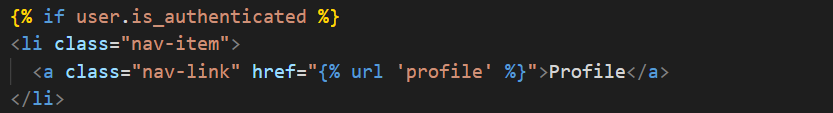


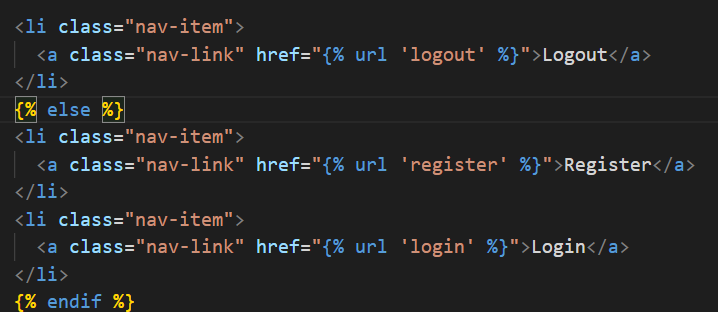
* Logout method created



### 8.4.1. User authenticated or not

* ‘user.is\_authenticated [if logged show these menu other show only home, register, login]





### 8.4.2. Login Required

* from django.contrib.auth.decorators import login\_required 
* @login\_required [before the definition of method]

