MINI PROJECT
(2020-2021)

RT Messenger

MID-TERM REPORT

Institute of Engineering & Technology

Submitted by-
Prashant Asthana
(181500487)
Tanishq Tripathi
(181500751)
Shriyanshi Baranwal
(181500691)

Supervised By: -
Mr. Amir Khan
Technical Trainer
Department of Computer Engineering & Applications

Contents

Abstract **3**

1. Introduction **4**

1.1 General Introduction to the topic **4**
1.2 Technical Details **4**
1.3 Hardware and Software Requirements **5**

2. Objectives **6**

3. Implementation Details **7**

4. Some Screenshots **8**

Abstract

Lately we see a constant switch towards real-time applications. With the wide support for WebSockets in recent browsers, more and more frameworks are giving us the ability to use them. Django, which was primarily a request/response-based framework, is doing the switch with Django Channels. Django channels is a lot more than just support for WebSockets, it's a complete architectural change for Django and - in my honest opinion - a great move towards the new era of frameworks.

The real-time chat application **does not** save your chat messages in the database before passing it to another client.That is because it uses sockets to send and receive messages between machines. The messages are exchanged through a network, either using a Transmission Control Protocol (“TCP”) or a User Datagram Protocol (“UDP”). You could save the messages if you want, but it would be separated from the socket send-receive process.

Introduction

1.1 General Introduction to the topic

Nowadays, when all sorts of chat rooms have become extremely popular, when every second large company has launched or developed its own instant messenger, when an increase of smiles and change in the text size is considered as innovation, in the era of iMessages, Slack, Hipchat, Messenger, Google Allo, Zulip, etc. We will use Django-channels.

Python is fast becoming a popular coding language in the world, and there are many popular frameworks that build on Python. One of them is Django and it has many functionalities and supporting libraries. For this article, we like to explore one interesting method that builds on Django to handle not only HTTP but also long running connections such as WebSockets, MQTT, chatbots, etc.

1.2 Technical Details

Basic example of a multi-room chatroom, with messages from all rooms a user is in multiplexed over a single WebSocket connection. There is no chat persistence; you only see messages sent to a room while you are in that room.

Uses the Django auth system to provide user accounts; users are only able to use the chat once logged in, and this provides their username details for the chatroom.

This package allows our application to interact with a user not only using HTTP 1.1 (request-response), but also using HTTP/2 and WebSocket.

WebSocket is designed for exchanging messages between the client and the web server in real time. You should consider it as an open channel between the client and the server, with the ability to subscribe to the events sent to it.

1.3 Hardware and Software Requirements

Software Specification:

- Technology Implemented: JavaScript, Django, python, WebSocket
- User Interface Design : Web based Application
- Web Browser: Chrome

Hardware Requirement:

- Processor: Intel CORE i3
- Operating System: Windows 10
- RAM:4 GB
- Hardware System: Computer System
- Hard Disk: 64 GB

Objectives

The main objective of creating a real-time messenger is:

Firstly, it saves storage. Real-time messaging doesn't save your messages, as the latter are just passing through the network, making your whole chat architecture storage friendly.

Secondly, as its name shows, it is "real-time" and we can see that this type of messaging is faster than the normal one. The traditional messaging system saves your messages in the database, so every time you need your messages, you have to query to the database to get them. The real-time messaging only passes messages every time we want to send and receive them when they arrive on the socket, so it should be faster and more efficient.

Django is a web building framework based on the Python programming language. It uses a Model-View-Template ("MVT") architecture. Django is being popular because it is fast, secure, and a scalable high-level Python programming for web building. Learning Django is not that easy, but when you get used to it, it has many great functionalities.

Implementation Details

1. Install and configure the environment. Create a virtual environment and Django project on Python as well as prepare them for work. It is important to add channels to the INSTALLED_APPS setting.

2. Install Redis or another channel layer. Configure Django-channels so that it will use the Redis as the channel storage by adding asgi_redis.RedisChannelLayer to the CHANNEL_LAYERS setting.

3. Add a basic message handler to multi-chat/routing.py in order to initialize the Django channels routing.

4. Check how everything works by installing any WebSocket client and running the Django web server together with the interactive session interpreter.

5. Connect to the web socket and try to send a chat message. If the message is duplicated in the server log, it means that the chat works correctly.

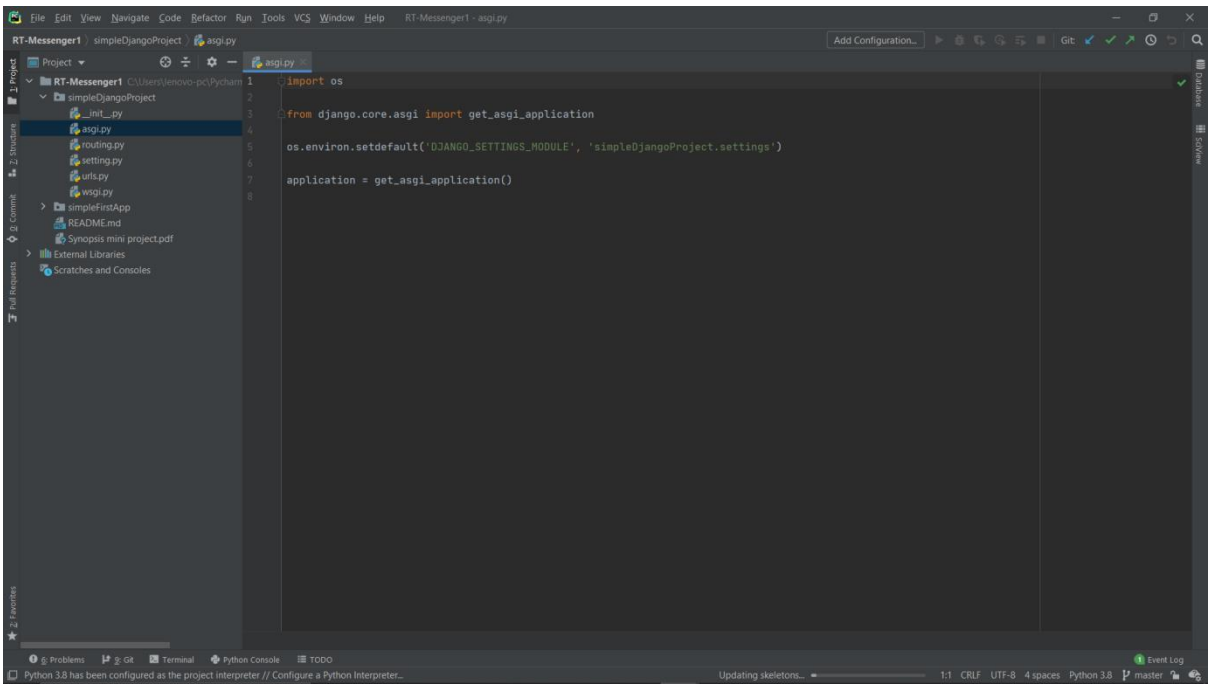
6. Create authorization page for the chat by creating a template and adding a stylesheet as well as redirection URLs for login and log out.

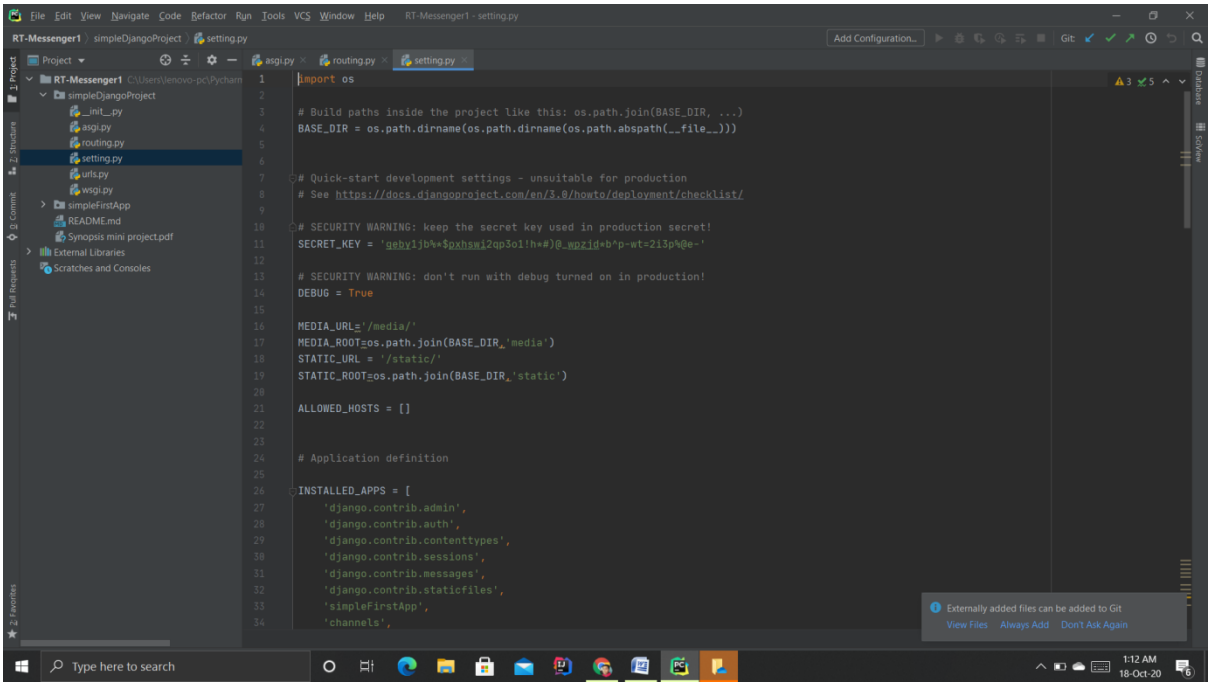
7. Create chat rooms by adding a 'Room' class to chat/models.py.

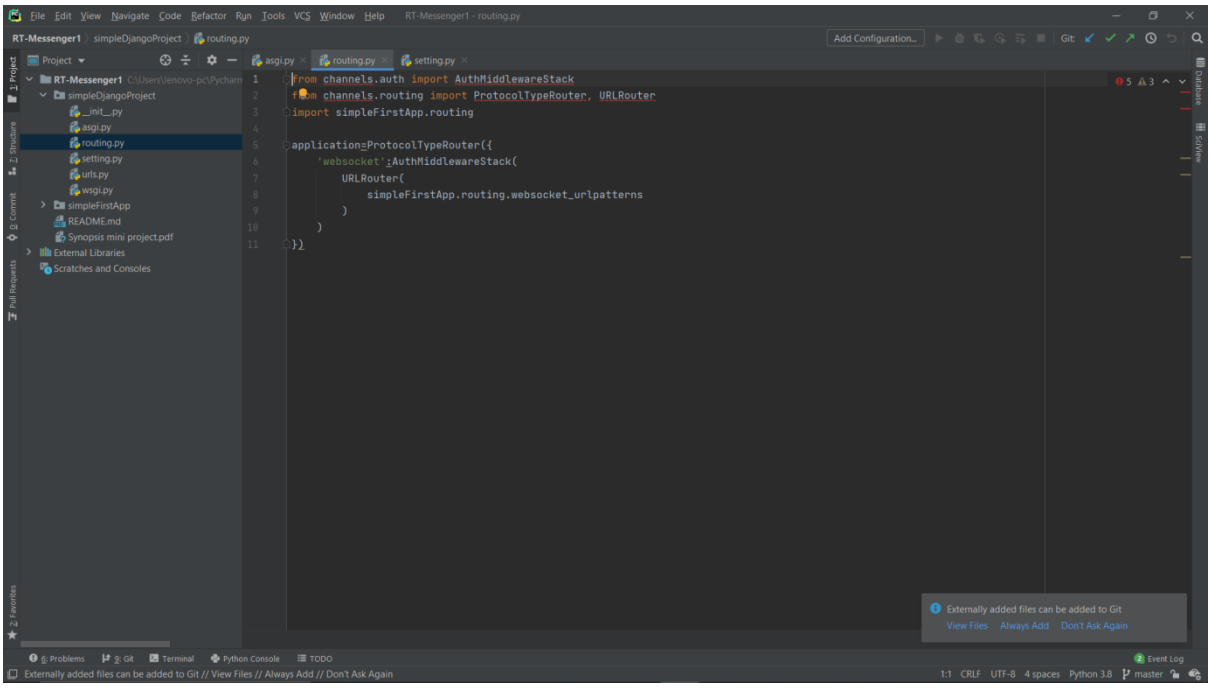
8. Add event handlers to manage events such as connecting to and disconnecting from the web socket, message processing, login/logout, and message sending.

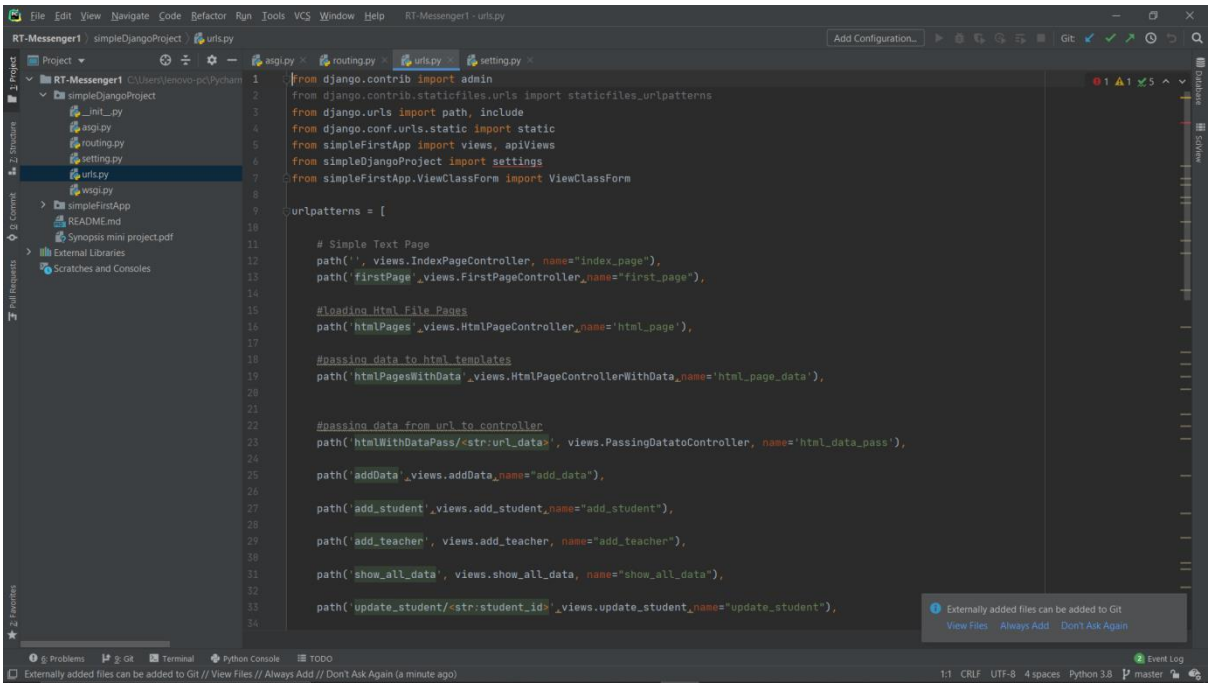
9. Create a front-end part of your chat by writing Python code for connecting to the socket, connecting to and disconnecting from the rooms, and exchanging messages.

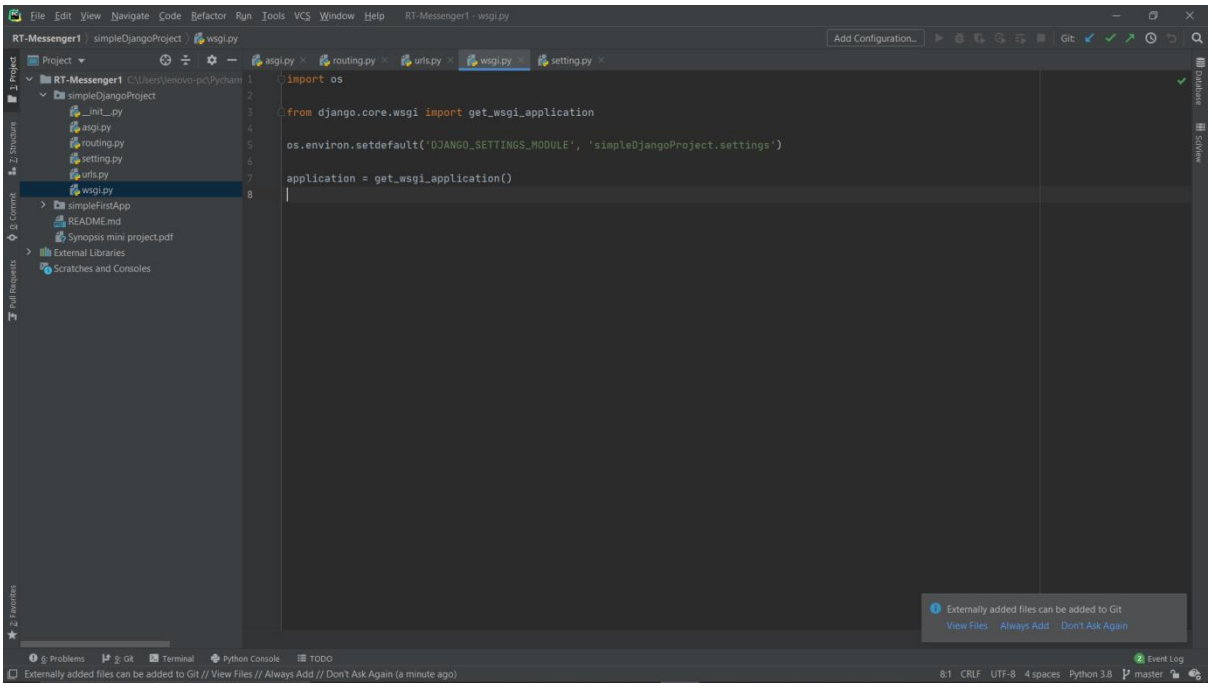
Some Screenshots

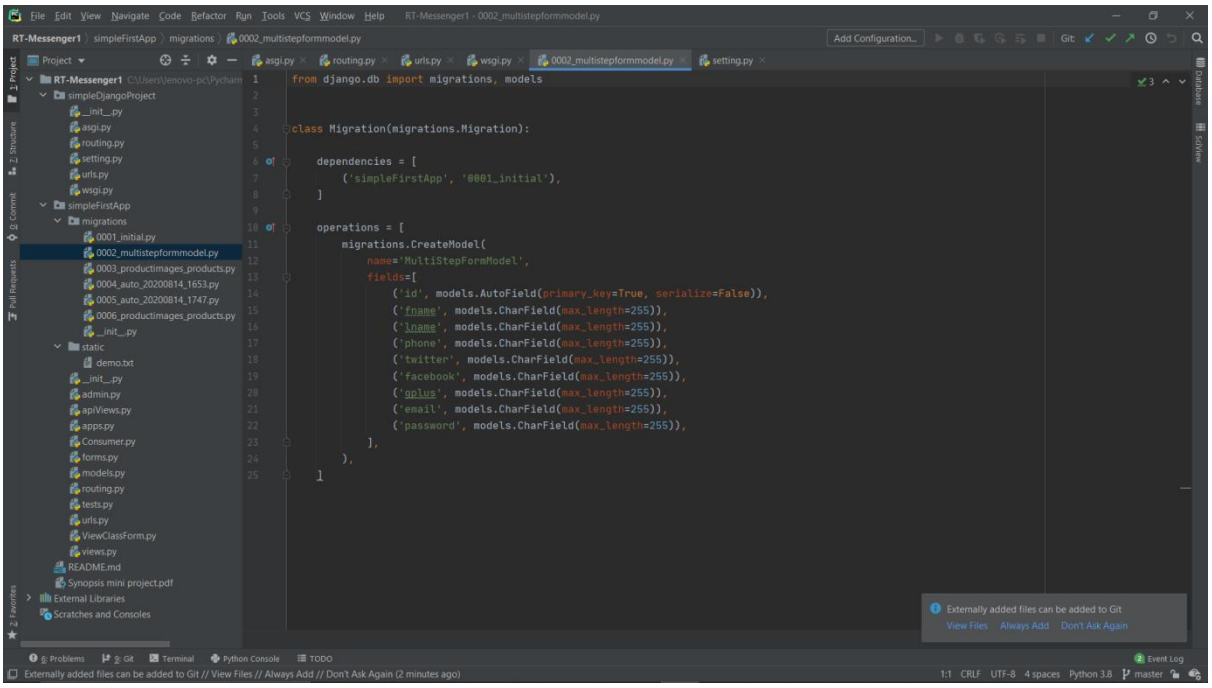


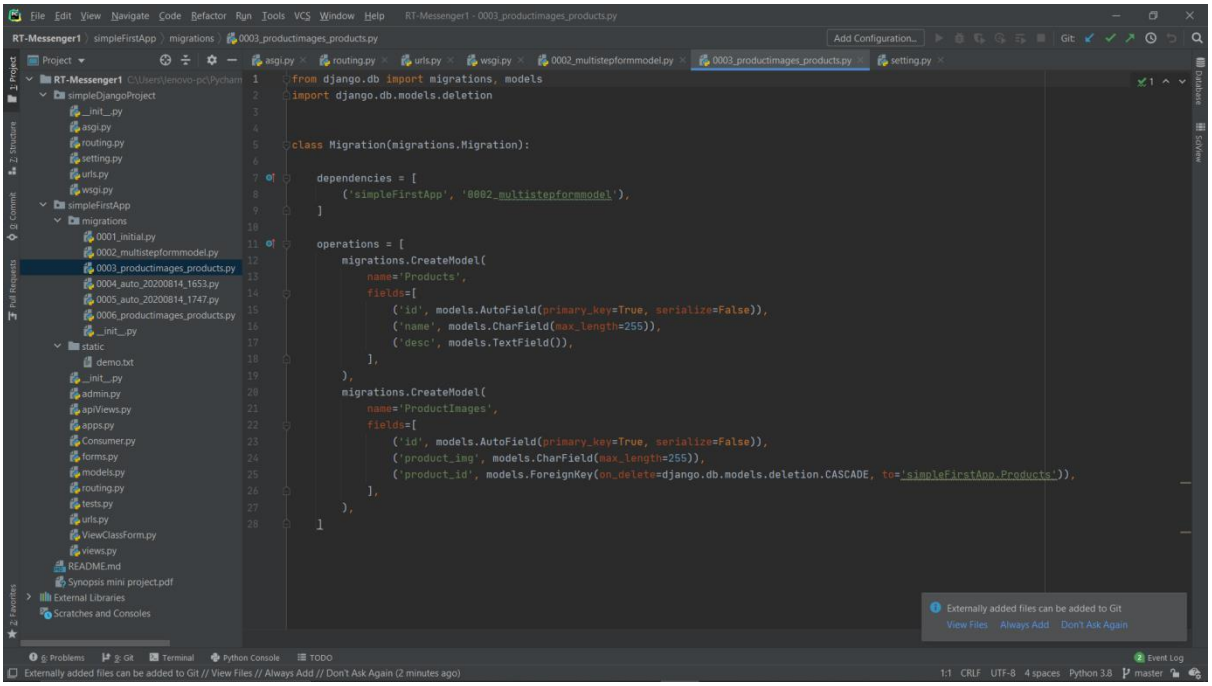


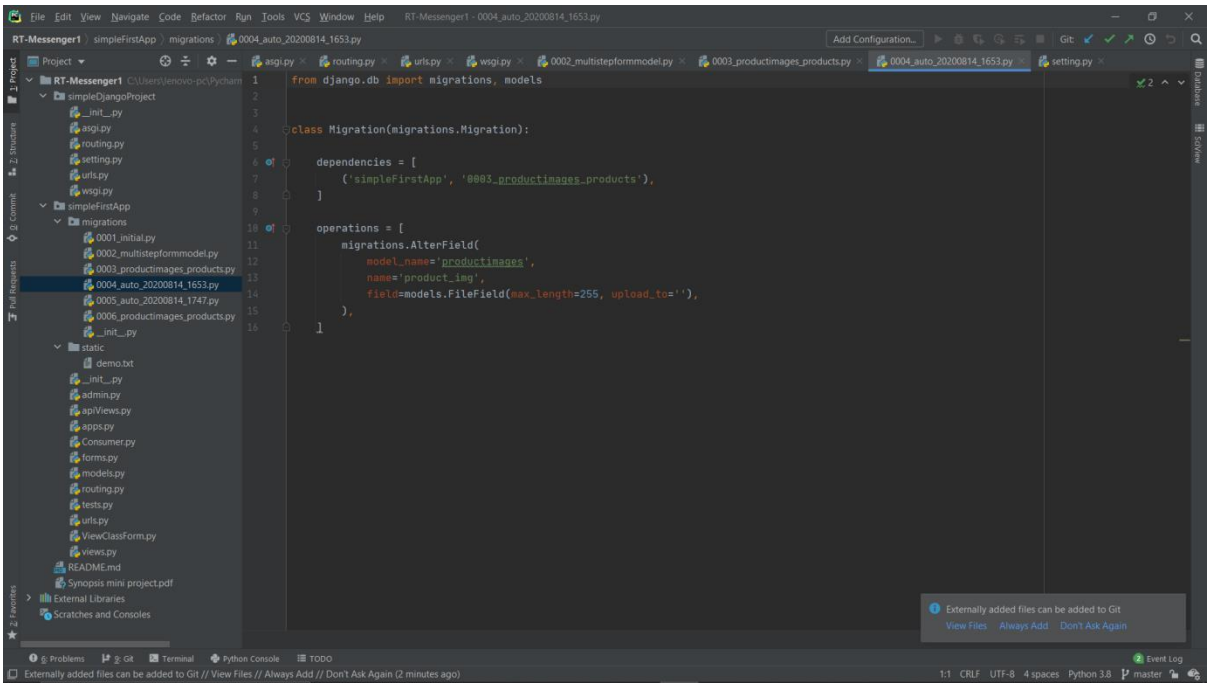

RT-Messenger1 - routing.py
RT-Messenger1 simpleDjangoProject routing.py
asgi.py routing.py setting.py
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
import simpleFirstApp.routing
application=ProtocolTypeRouter({
'websocket':AuthMiddlewareStack(
URLRouter(
simpleFirstApp.routing.websocket_urlpatterns
)
)
})
Externally added files can be added to Git
View Files Always Add Don't Ask Again

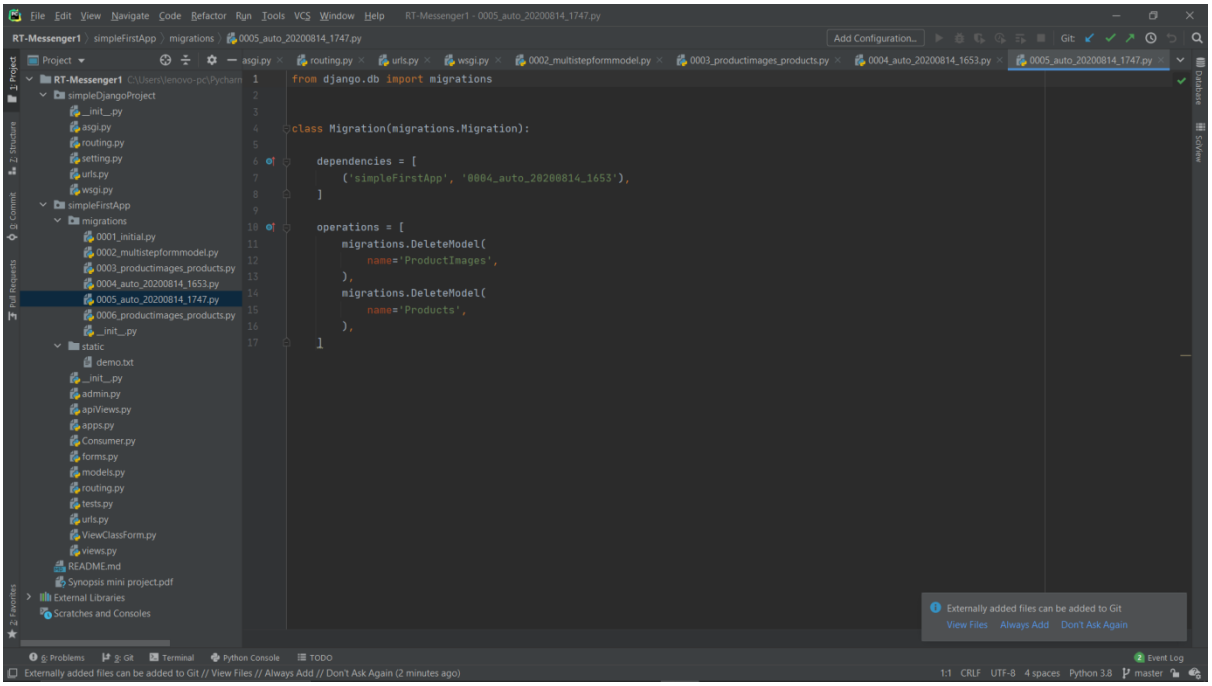

RT-Messenger1 - urls.py
RT-Messenger1 simpleDjangoProject urls.py
asgi.py routing.py urls.py setting.py
from django.contrib import admin
from django.contrib.staticfiles.urls import staticfiles_urlpatterns
from django.urls import path, include
from django.conf.urls.static import static
from simpleFirstApp import views, apiViews
from simpleDjangoProject import settings
from simpleFirstApp.ViewClassForm import ViewClassForm
urlpatterns = [
# Simple Text Page
path('', views.IndexPageController, name="index_page"),
path('firstPage',views.FirstPageController,name="first_page"),
#loading Html File Pages
path('htmlPages',views.HtmlPageController,name='html_page'),
#passing data to html templates
path('htmlPagesWithData',views.HtmlPageControllerWithData,name='html_page_data'),
#passing data from url to controller
path('htmlWithDataPass/<str:url_data>', views.PassingDatatoController, name='html_data_pass'),
path('addData',views.addData,name="add_data"),
path('add_student',views.add_student,name="add_student"),
path('add_teacher', views.add_teacher, name="add_teacher"),
path('show_all_data', views.show_all_data, name="show_all_data"),
path('update_student/<str:student_id>',views.update_student,name="update_student"),
Externally added files can be added to Git
View Files Always Add Don't Ask Again

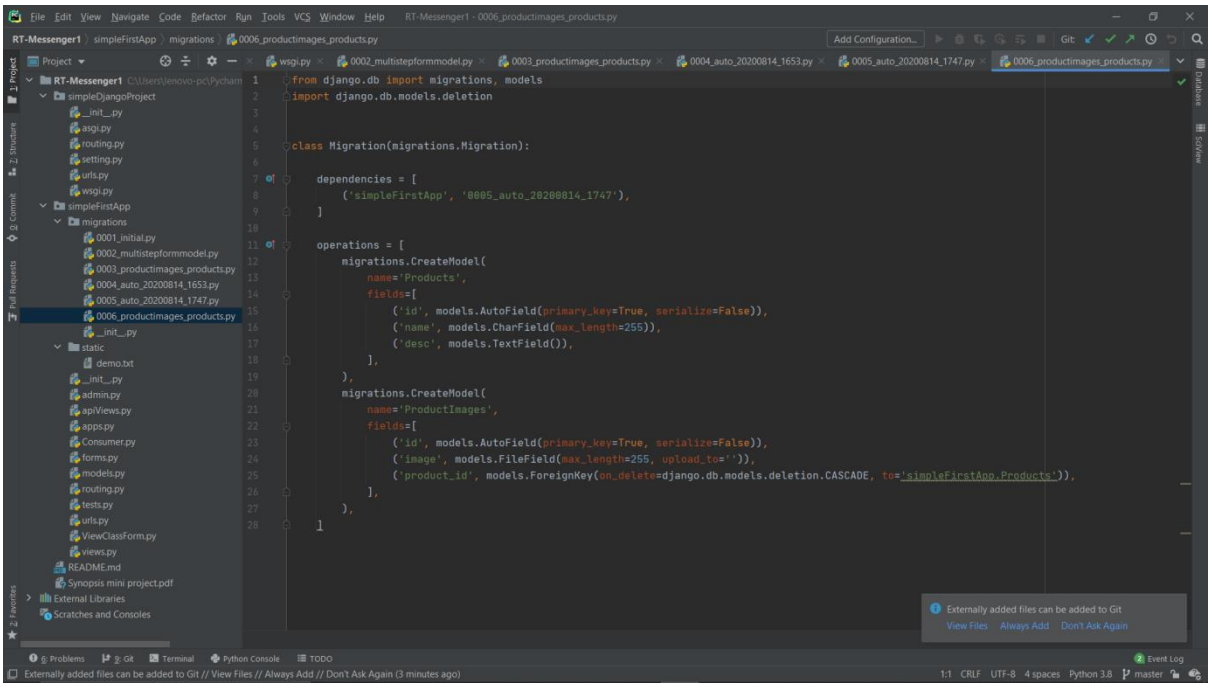

File Edit View Navigate Code Refactor Run Tools VCS Window Help RT-Messenger1 - wsgi.py
RT-Messenger1 simpleDjangoProject wsgi.py
Add Configuration...
Project
asgi.py routing.py urls.py wsgi.py setting.py
RT-Messenger1 C:\Users\lenovo-pc\Pycham
simpleDjangoProject
__init__.py
asgi.py
routing.py
setting.py
urls.py
wsgi.py
simpleFirstApp
README.md
Synopsis mini project.pdf
External Libraries
Scratches and Consoles
import os
from django.core.wsgi import get_wsgi_application
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'simpleDjangoProject.settings')
application = get_wsgi_application()
Externally added files can be added to Git
View Files Always Add Don't Ask Again
6: Problems 9: Git Terminal Python Console TODO
Event Log
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (a minute ago)
8:1 CRLF UTF-8 4 spaces Python 3.8 master

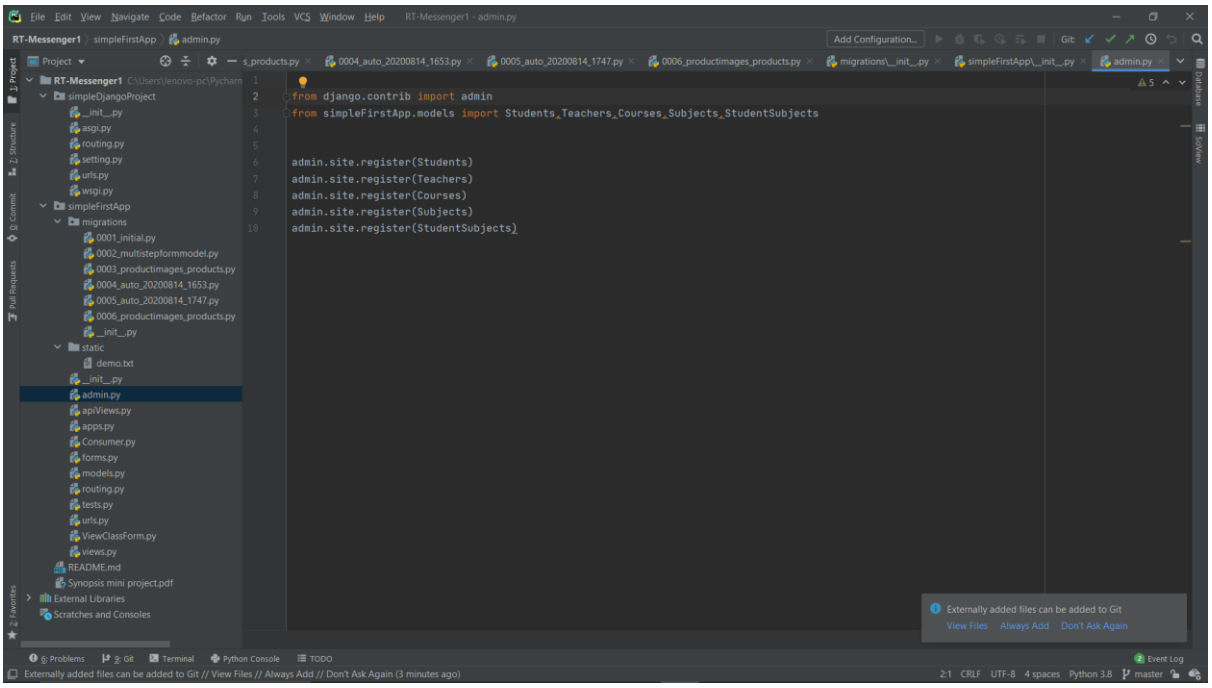

File Edit View Navigate Code Refactor Run Tools VCS Window Help RT-Messenger1 - 0002_multistepformmodel.py
RT-Messenger1 simpleFirstApp migrations 0002_multistepformmodel.py
Add Configuration...
Project
asgi.py routing.py urls.py wsgi.py 0002_multistepformmodel.py setting.py
RT-Messenger1 C:\Users\lenovo-pc\Pycham
simpleDjangoProject
__init__.py
asgi.py
routing.py
setting.py
urls.py
wsgi.py
simpleFirstApp
migrations
0001_initial.py
0002_multistepformmodel.py
0003_productimages_products.py
0004_auto_20200814_1653.py
0005_auto_20200814_1747.py
0006_productimages_products.py
__init__.py
static
demo.txt
__init__.py
admin.py
apiViews.py
apps.py
Consumer.py
forms.py
models.py
routing.py
tests.py
urls.py
ViewClassForm.py
views.py
README.md
Synopsis mini project.pdf
External Libraries
Scratches and Consoles
from django.db import migrations, models
class Migration(migrations.Migration):
dependencies = [
('simpleFirstApp', '0001_initial'),
]
operations = [
migrations.CreateModel(
name='MultiStepFormModel',
fields=[
('id', models.AutoField(primary_key=True, serialize=False)),
('fname', models.CharField(max_length=255)),
('lname', models.CharField(max_length=255)),
('phone', models.CharField(max_length=255)),
('twitter', models.CharField(max_length=255)),
('facebook', models.CharField(max_length=255)),
('gplus', models.CharField(max_length=255)),
('email', models.CharField(max_length=255)),
('password', models.CharField(max_length=255)),
],
),
]
Externally added files can be added to Git
View Files Always Add Don't Ask Again
6: Problems 9: Git Terminal Python Console TODO
Event Log
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (2 minutes ago)
1:1 CRLF UTF-8 4 spaces Python 3.8 master

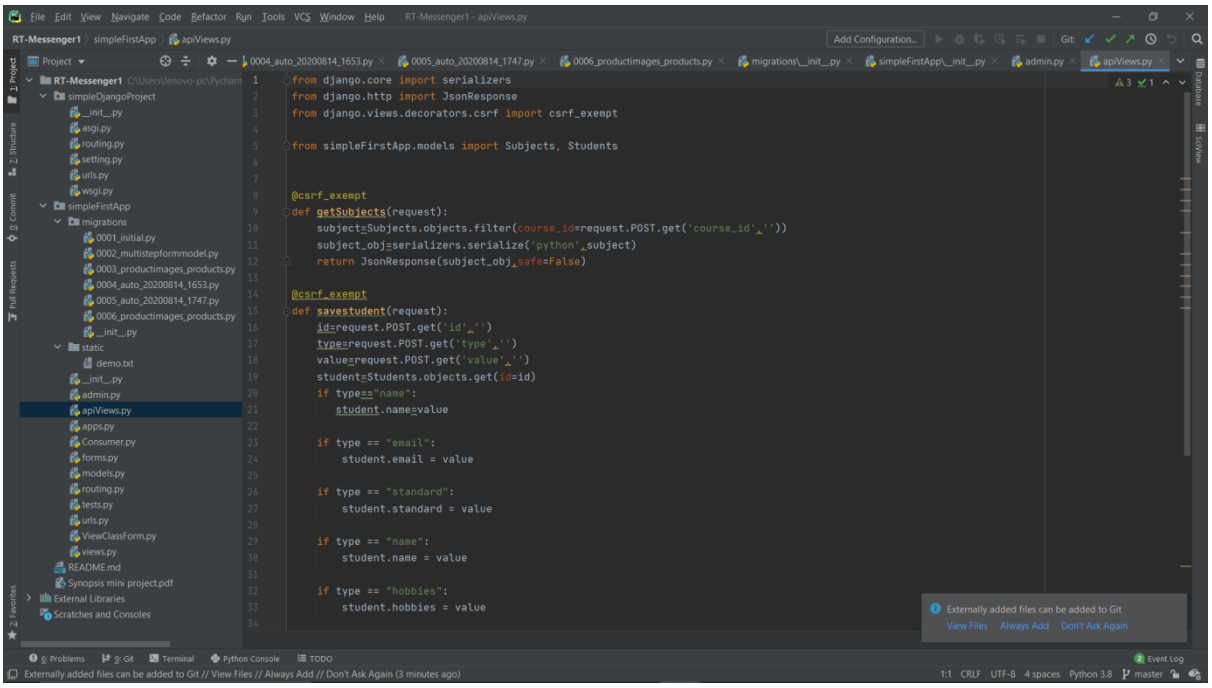

RT-Messenger1 - 0003_productimages_products.py
RT-Messenger1 simpleFirstApp migrations 0003_productimages_products.py
Add Configuration...
asgi.py routing.py urls.py wsgi.py 0002_multistepformmodel.py 0003_productimages_products.py setting.py
from django.db import migrations, models
import django.db.models.deletion
class Migration(migrations.Migration):
dependencies = [
('simpleFirstApp', '0002_multistepformmodel'),
]
operations = [
migrations.CreateModel(
name='Products',
fields=[
('id', models.AutoField(primary_key=True, serialize=False)),
('name', models.CharField(max_length=255)),
('desc', models.TextField()),
],
),
migrations.CreateModel(
name='ProductImages',
fields=[
('id', models.AutoField(primary_key=True, serialize=False)),
('product_img', models.CharField(max_length=255)),
('product_id', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='simpleFirstApp.Products')),
],
),
]
Externally added files can be added to Git
View Files Always Add Don't Ask Again
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (2 minutes ago)

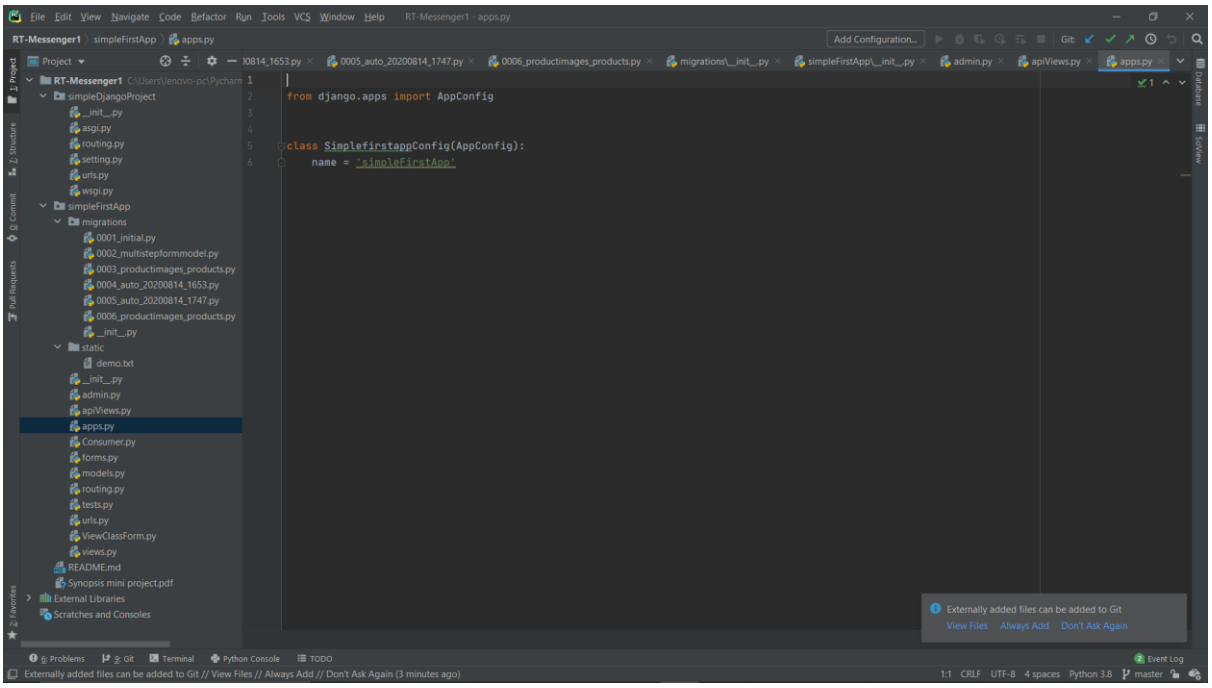

RT-Messenger1 - 0004_auto_20200814_1653.py
RT-Messenger1 simpleFirstApp migrations 0004_auto_20200814_1653.py
Add Configuration...
asgi.py routing.py urls.py wsgi.py 0002_multistepformmodel.py 0003_productimages_products.py 0004_auto_20200814_1653.py setting.py
from django.db import migrations, models
class Migration(migrations.Migration):
dependencies = [
('simpleFirstApp', '0003_productimages_products'),
]
operations = [
migrations.AlterField(
model_name='productimages',
name='product_img',
field=models.FileField(max_length=255, upload_to=''),
),
]
Externally added files can be added to Git
View Files Always Add Don't Ask Again
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (2 minutes ago)

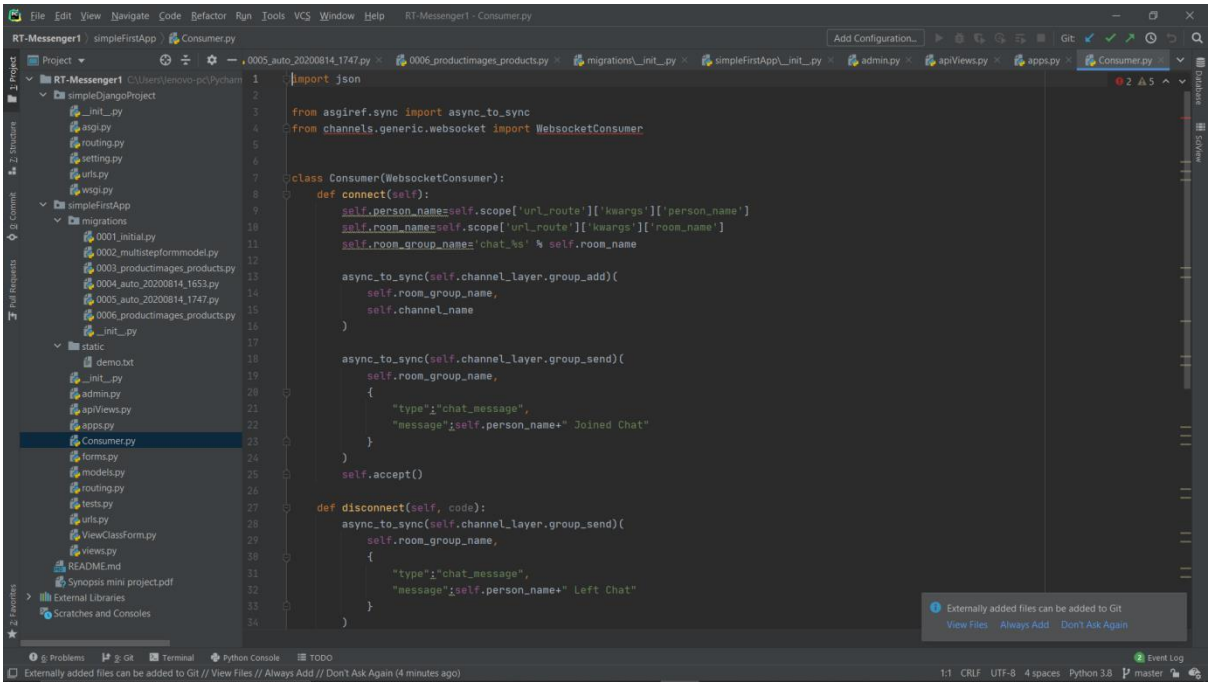

RT-Messenger1 - 0005_auto_20200814_1747.py
0005_auto_20200814_1747.py
from django.db import migrations
class Migration(migrations.Migration):
dependencies = [
('simpleFirstApp', '0004_auto_20200814_1653'),
]
operations = [
migrations.DeleteModel(
name='ProductImages',
),
migrations.DeleteModel(
name='Products',
),
]
Externally added files can be added to Git
View Files Always Add Don't Ask Again

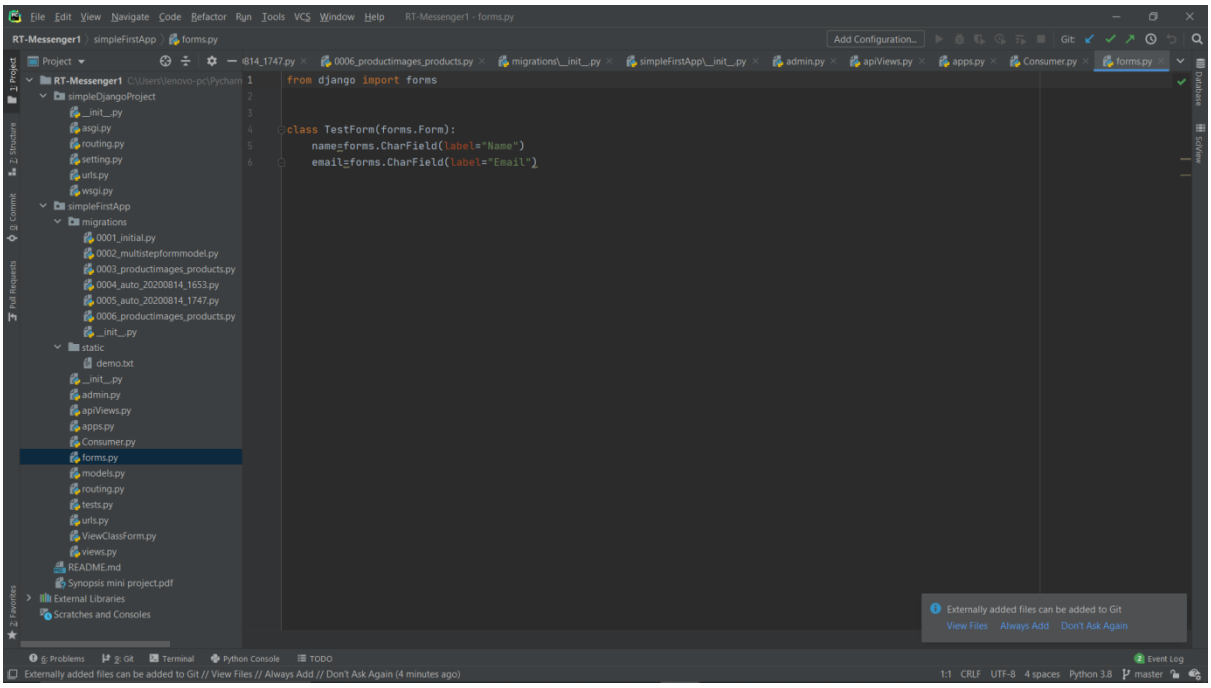

RT-Messenger1 - 0006_productimages_products.py
0006_productimages_products.py
from django.db import migrations, models
import django.db.models.deletion
class Migration(migrations.Migration):
dependencies = [
('simpleFirstApp', '0005_auto_20200814_1747'),
]
operations = [
migrations.CreateModel(
name='Products',
fields=[
('id', models.AutoField(primary_key=True, serialize=False)),
('name', models.CharField(max_length=255)),
('desc', models.TextField()),
],
),
migrations.CreateModel(
name='ProductImages',
fields=[
('id', models.AutoField(primary_key=True, serialize=False)),
('image', models.FileField(max_length=255, upload_to='')),
('product_id', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='simpleFirstApp.Products')),
],
),
]
Externally added files can be added to Git
View Files Always Add Don't Ask Again

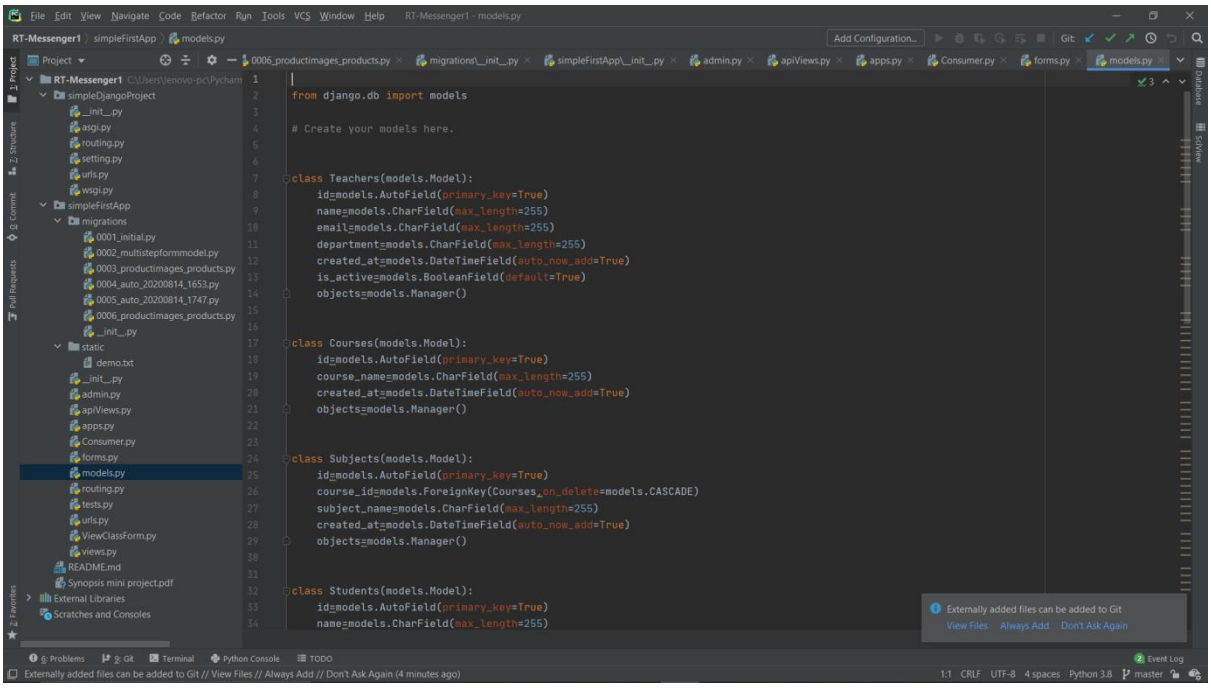

RT-Messenger1 - admin.py
from django.contrib import admin
from simpleFirstApp.models import Students,Teachers,Courses,Subjects,StudentSubjects
admin.site.register(Students)
admin.site.register(Teachers)
admin.site.register(Courses)
admin.site.register(Subjects)
admin.site.register(StudentSubjects)

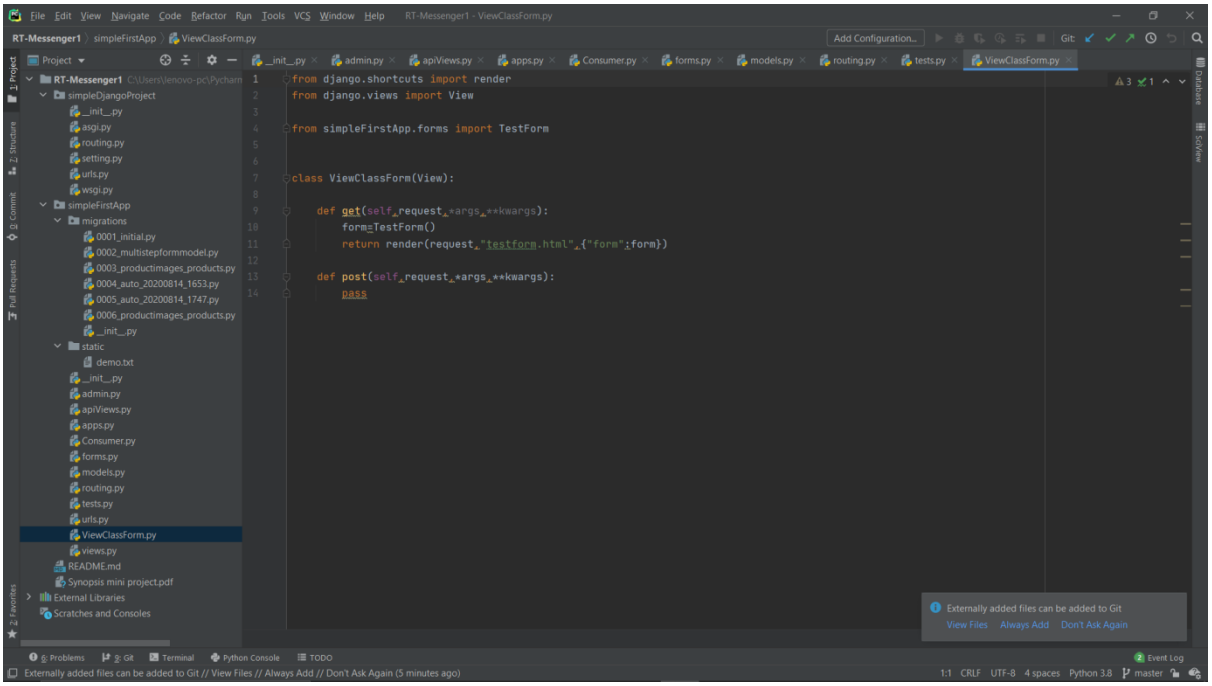

RT-Messenger1 - apiViews.py
from django.core import serializers
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from simpleFirstApp.models import Subjects, Students
@csrf_exempt
def getSubjects(request):
subject=Subjects.objects.filter(course_id=request.POST.get('course_id',''))
subject_obj=serializers.serialize('python',subject)
return JsonResponse(subject_obj,safe=False)
@csrf_exempt
def savestudent(request):
id=request.POST.get('id','')
type=request.POST.get('type','')
value=request.POST.get('value','')
student=Students.objects.get(id=id)
if type=="name":
student.name=value
if type == "email":
student.email = value
if type == "standard":
student.standard = value
if type == "name":
student.name = value
if type == "hobbies":
student.hobbies = value


RT-Messenger1 - apps.py
from django.apps import AppConfig
class Simplefirstappconfig(AppConfig):
name = 'simpleFirstApp'


RT-Messenger1 - Consumer.py
import json
from asgiref.sync import async_to_sync
from channels.generic.websocket import WebsocketConsumer
class Consumer(WebsocketConsumer):
def connect(self):
self.person_name=self.scope['url_route']['kwargs']['person_name']
self.room_name=self.scope['url_route']['kwargs']['room_name']
self.room_group_name='chat_%s' % self.room_name
async_to_sync(self.channel_layer.group_add)(
self.room_group_name,
self.channel_name
)
async_to_sync(self.channel_layer.group_send)(
self.room_group_name,
{
"type":"chat_message",
"message":self.person_name+" Joined Chat"
}
)
self.accept()
def disconnect(self, code):
async_to_sync(self.channel_layer.group_send)(
self.room_group_name,
{
"type":"chat_message",
"message":self.person_name+" Left Chat"
}
)


RT-Messenger1 - forms.py
from django import forms
class TestForm(forms.Form):
name=forms.CharField(label="Name")
email=forms.CharField(label="Email")


RT-Messenger1 - models.py
from django.db import models
# Create your models here.
class Teachers(models.Model):
id=models.AutoField(primary_key=True)
name=models.CharField(max_length=255)
email=models.CharField(max_length=255)
department=models.CharField(max_length=255)
created_at=models.DateTimeField(auto_now_add=True)
is_active=models.BooleanField(default=True)
objects=models.Manager()
class Courses(models.Model):
id=models.AutoField(primary_key=True)
course_name=models.CharField(max_length=255)
created_at=models.DateTimeField(auto_now_add=True)
objects=models.Manager()
class Subjects(models.Model):
id=models.AutoField(primary_key=True)
course_id=models.ForeignKey(Courses,on_delete=models.CASCADE)
subject_name=models.CharField(max_length=255)
created_at=models.DateTimeField(auto_now_add=True)
objects=models.Manager()
class Students(models.Model):
id=models.AutoField(primary_key=True)
name=models.CharField(max_length=255)


RT-Messenger1 - ViewClassForm.py
RT-Messenger1 simpleFirstApp ViewClassForm.py
Add Configuration...
Project
RT-Messenger1 C:\Users\lenovo-pc\Pycharm
simpleDjangoProject
__init__.py
asgi.py
routing.py
setting.py
urls.py
wsgi.py
simpleFirstApp
migrations
0001_initial.py
0002_multistepformmodel.py
0003_productimages_products.py
0004_auto_20200814_1653.py
0005_auto_20200814_1747.py
0006_productimages_products.py
__init__.py
static
demo.txt
__init__.py
admin.py
apiViews.py
apps.py
Consumer.py
forms.py
models.py
routing.py
tests.py
urls.py
ViewClassForm.py
views.py
README.md
Synopsis mini project.pdf
External Libraries
Scratches and Consoles
from django.shortcuts import render
from django.views import View
from simpleFirstApp.forms import TestForm
class ViewClassForm(View):
def get(self,request,*args,**kwargs):
form=TestForm()
return render(request,"testform.html",{"form":form})
def post(self,request,*args,**kwargs):
pass
Externally added files can be added to Git
View Files Always Add Don't Ask Again
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (5 minutes ago)