

# Django And Flutter — A Step by Step Tutorial for a Boilerplate Application

## Introduction

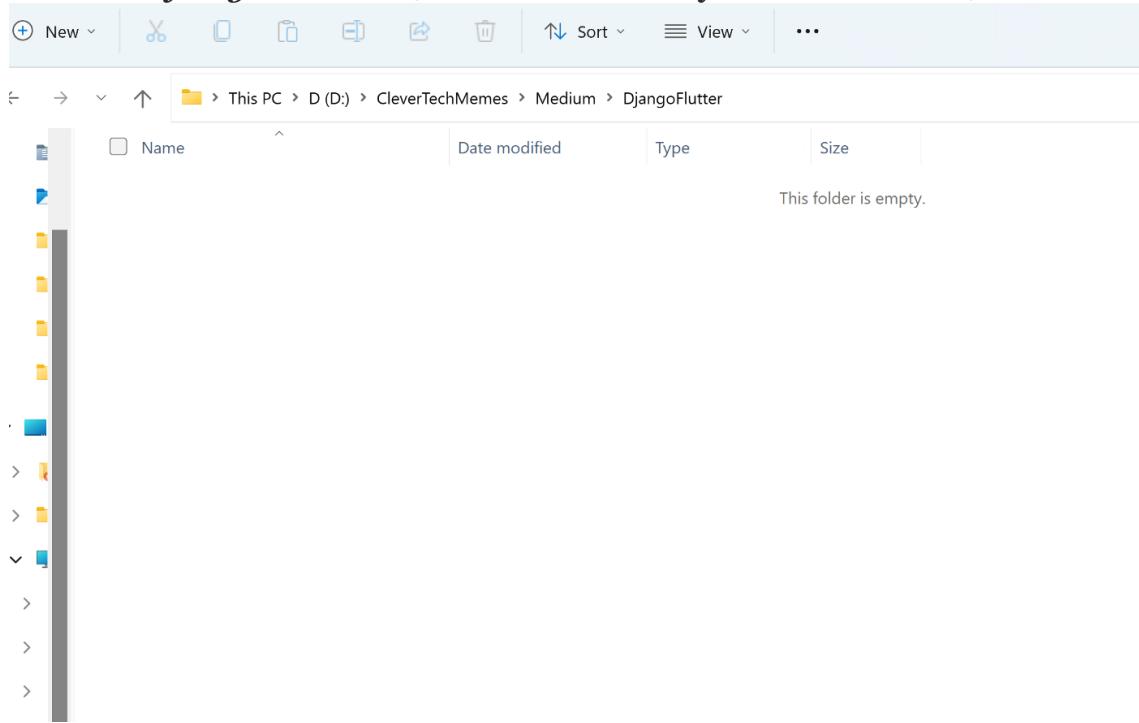
*Django* and *Flutter* are two fundamental backend and frontend tools for entrepreneurs and programmers to develop their applications. *Django* is well-known for its flexible and fast development capabilities based on *Python* programming language and *Flutter* is a platform-independent frontend technology based on the *Dart* language. In this tutorial, it is going to be shown step by step how to create a *Django* and *Flutter* application and what changes are needed to be applied to both that they can communicate with each other.

You can find the code for the *Django* and *Flutter* boilerplate at [Clever Tech Memes GitHub repository](#). You can follow me on [Clever Tech Memes](#), [Twitter](#) and [YouTube](#) for the latest educational content.

## Preparing the Environment and the Requirements

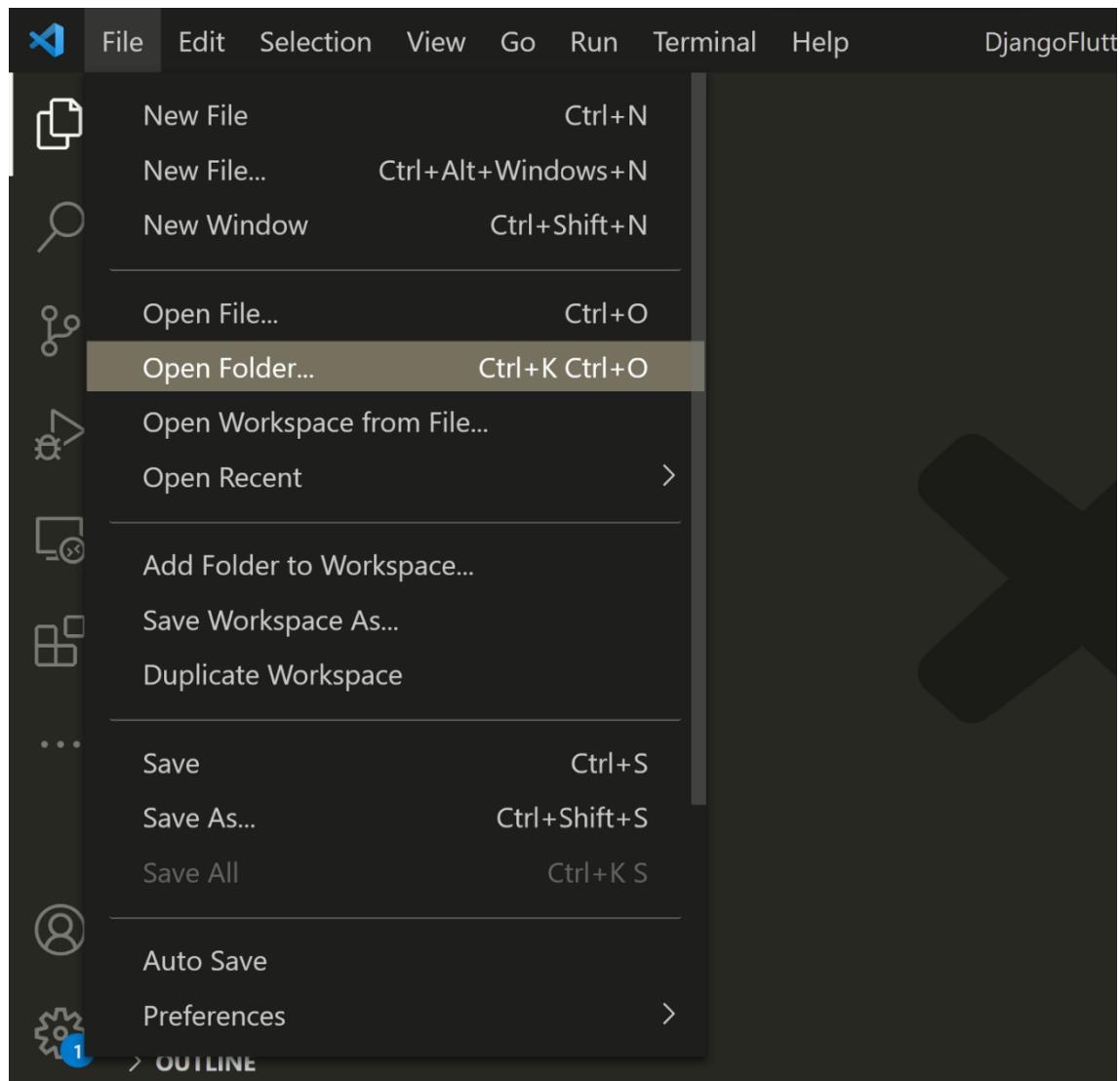
For this tutorial, it is required that you [install Flutter](#) based on your operating system. Also, you need to [install VS CODE IDE](#) for your machine. After, the installation completes, you need to open *VS CODE*, and create a folder to keep

your *Django* and *Flutter* code like the following empty folder named *DjangoFlutter* (You can choose your own name):

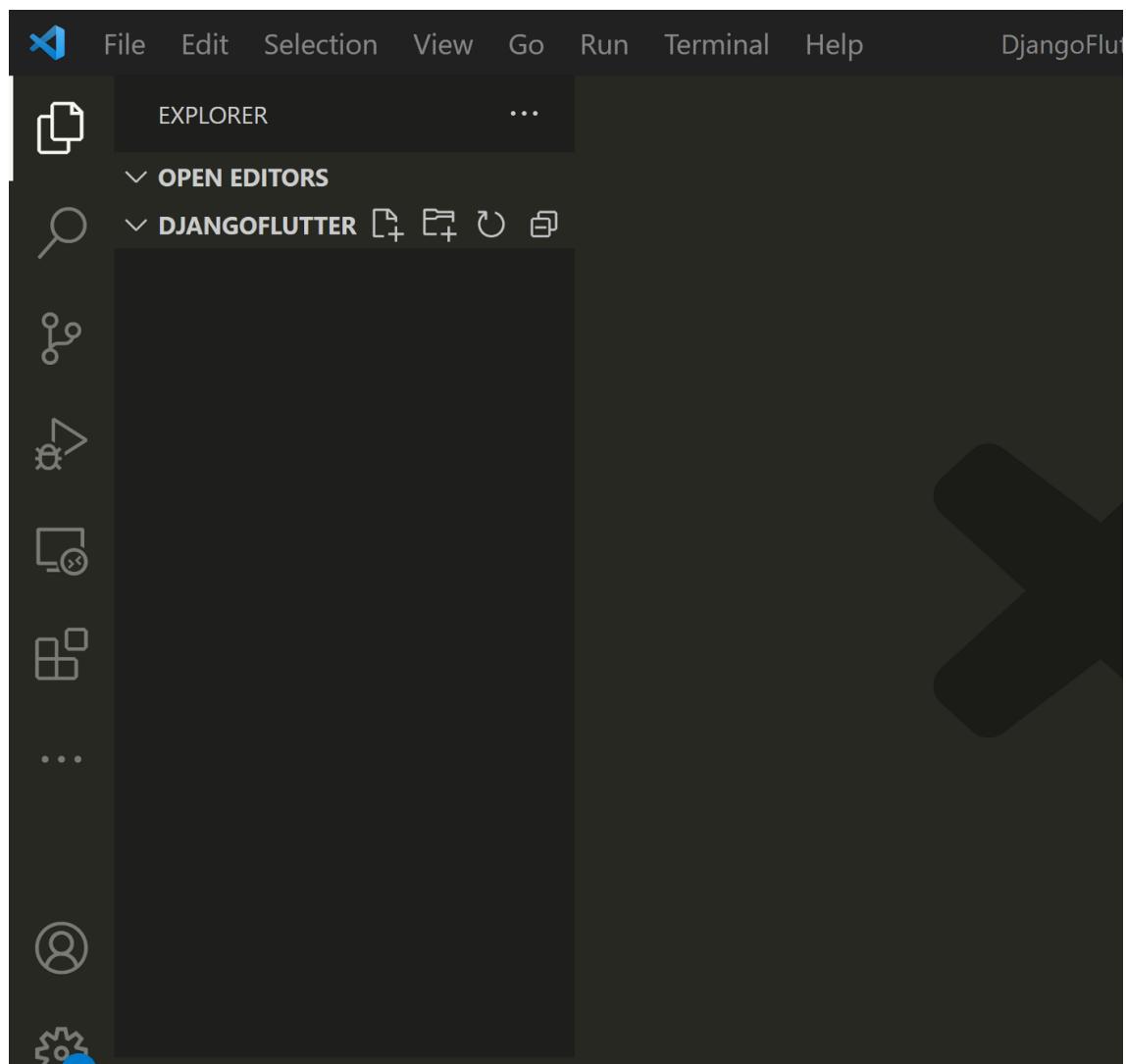


Empty folder to keep your Django and Flutter code

After, you open the *VSCODE*, you can open the *DjangoFlutter* folder by clicking on File and Open Folder as follows:

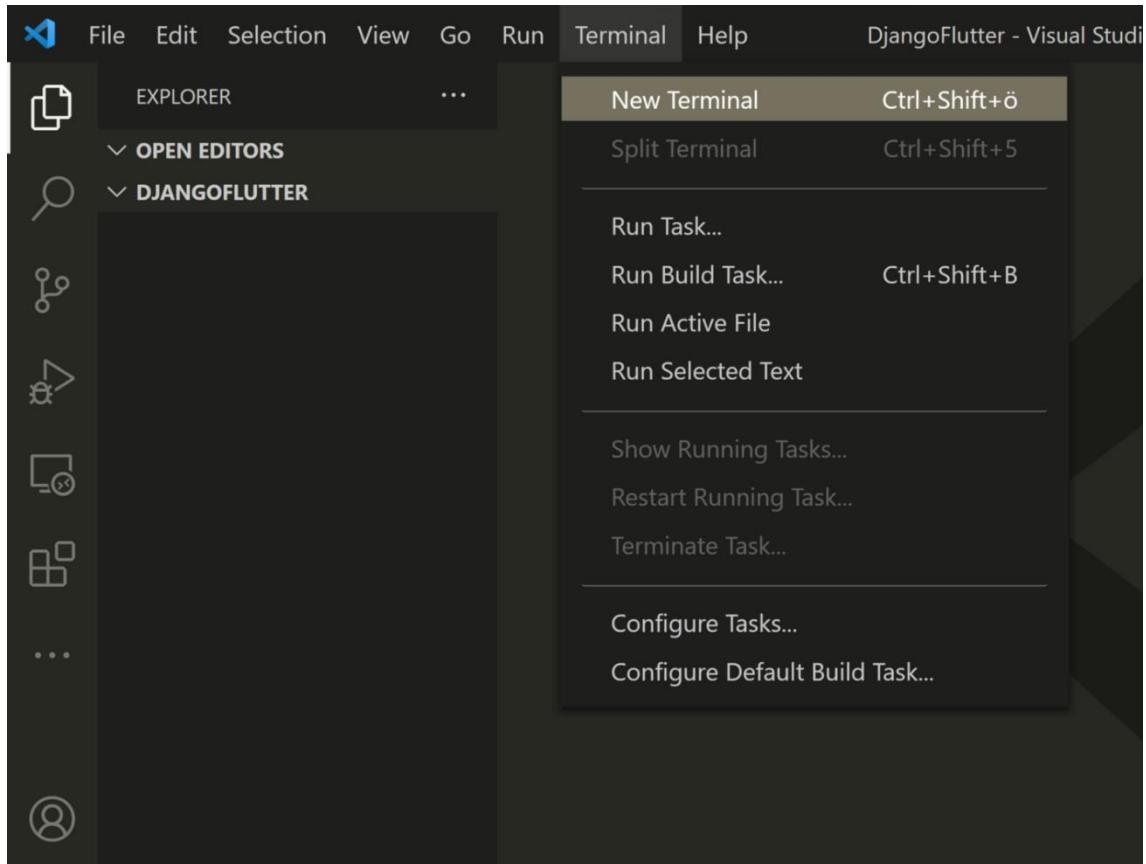


Now, you can see the DjangoFlutter folder on your VS CODE explorer as follows:



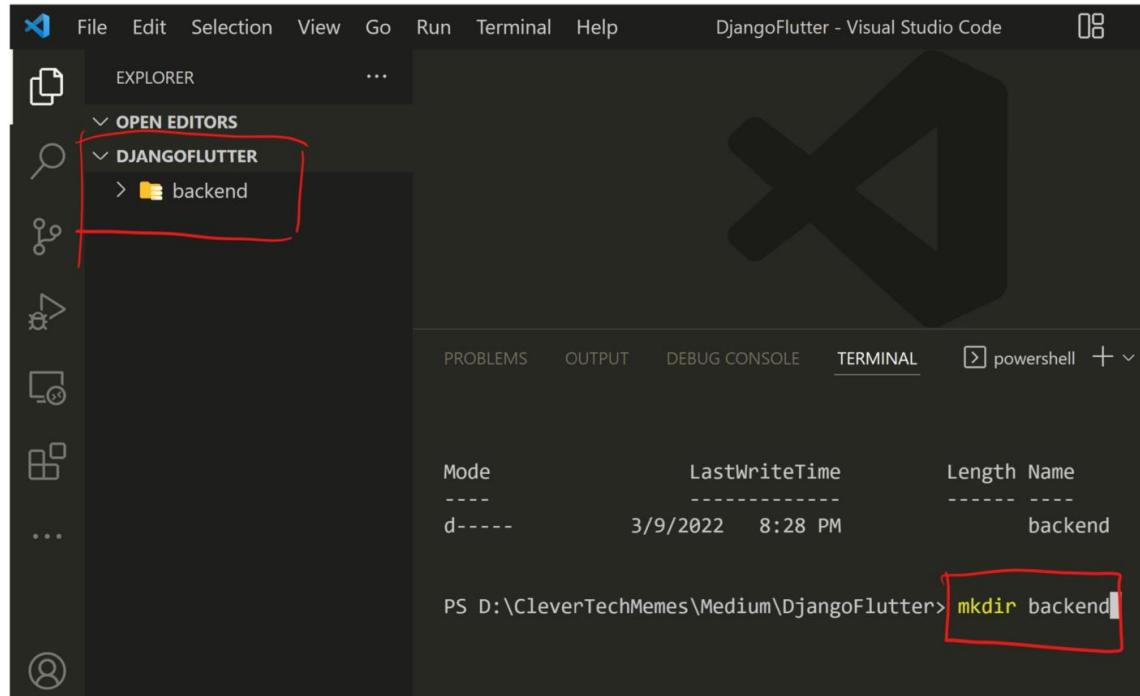
Seeing the Djangoflutter folder

Open a new terminal as shown below:



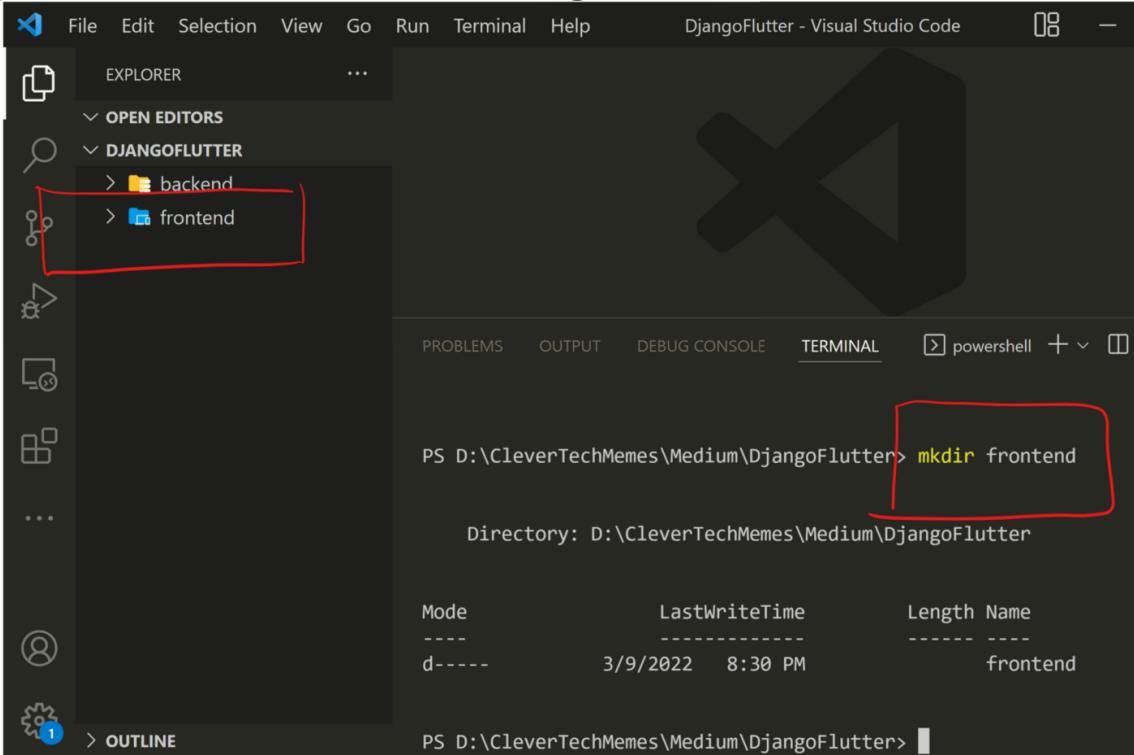
Opening a terminal in VS CODE

Now, you can create a folder named *backend* using the *mkdir* command:



Creating a backend folder

Then, a folder named *frontend* is created using *mkdir frontend* command like following:



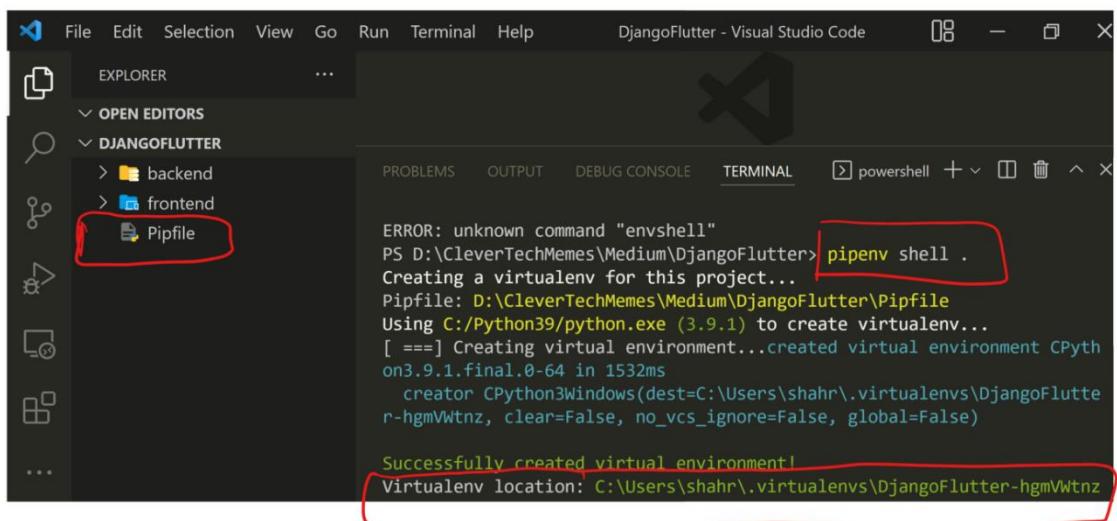
The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named "DJANGOFLUTTER" which contains two subfolders: "backend" and "frontend". A red box highlights the "frontend" folder. In the Terminal tab, the command "PS D:\CleverTechMemes\Medium\DjangoFlutter> mkdir frontend" is being typed, with the entire command highlighted by a red box. Below the command, the output shows the directory listing for "frontend".

```
PS D:\CleverTechMemes\Medium\DjangoFlutter> mkdir frontend
Directory: D:\CleverTechMemes\Medium\DjangoFlutter

Mode                LastWriteTime         Length Name
----                -              -          -
d-----            3/9/2022   8:30 PM           frontend
```

Creating a frontend folder

It is recommended to work within a virtual environment, so let us create a virtual environment for installation of the libraries, so we use “***pipenv shell .***” as follows:



The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named "DJANGOFLUTTER" which contains three files: "backend", "frontend", and "Pipfile". A red box highlights the "Pipfile" file. In the Terminal tab, the command "PS D:\CleverTechMemes\Medium\DjangoFlutter> pipenv shell ." is being typed, with the command highlighted by a red box. The terminal output shows the process of creating a virtual environment, including the creation of a virtual environment named "DjangoFlutter-hgmVWtnz" at the location "C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmVWtnz".

```
ERROR: unknown command "envshell"
PS D:\CleverTechMemes\Medium\DjangoFlutter> pipenv shell .
Creating a virtualenv for this project...
Pipfile: D:\CleverTechMemes\Medium\DjangoFlutter\Pipfile
Using C:/Python39/python.exe (3.9.1) to create virtualenv...
[ == ] Creating virtual environment...created virtual environment CPython3.9.1.final.0-64 in 1532ms
    creator CPython3Windows(dest=C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmVWtnz, clear=False, no_vcs_ignore=False, global=False)

Successfully created virtual environment!
Virtualenv location: C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmVWtnz
```

Creating a virtual environment

In case, you do not have installed *Python*, *pip* or *pipenv*, you need to install them. You can have a look at this article “[Installing pip](#)” for the installation of *pip*. Especially, you can refer to the following section for installing *pip*:

### How To Install Pip With Get-Pip.Py

1. To manually install pip on Windows, you will need a copy of [get-pip.py](#). For older Python versions, you may need to use the appropriate version of the file from [pypa.org](#). Download the file to a folder on your computer, or use the curl command:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

2. Next, run the following command to install pip:

```
python get-pip.py
```

### Installing Pip

Now, one can go to the path containing all the virtual environments and see the created virtual environment:

The screenshot shows a Windows Command Prompt window. The user has navigated to the directory `C:\Users\shahr\.virtualenvs`. They run the command `dir` to list the contents of the directory. A red box highlights the output of the `dir` command, which shows several virtual environment folders. One folder, `DjangoFlutter-hgmVWtnz`, is highlighted with a red box and has its full path displayed below it: `PS C:\Users\shahr\.virtualenvs> cd C:\Users\shahr\.virtualenvs`.

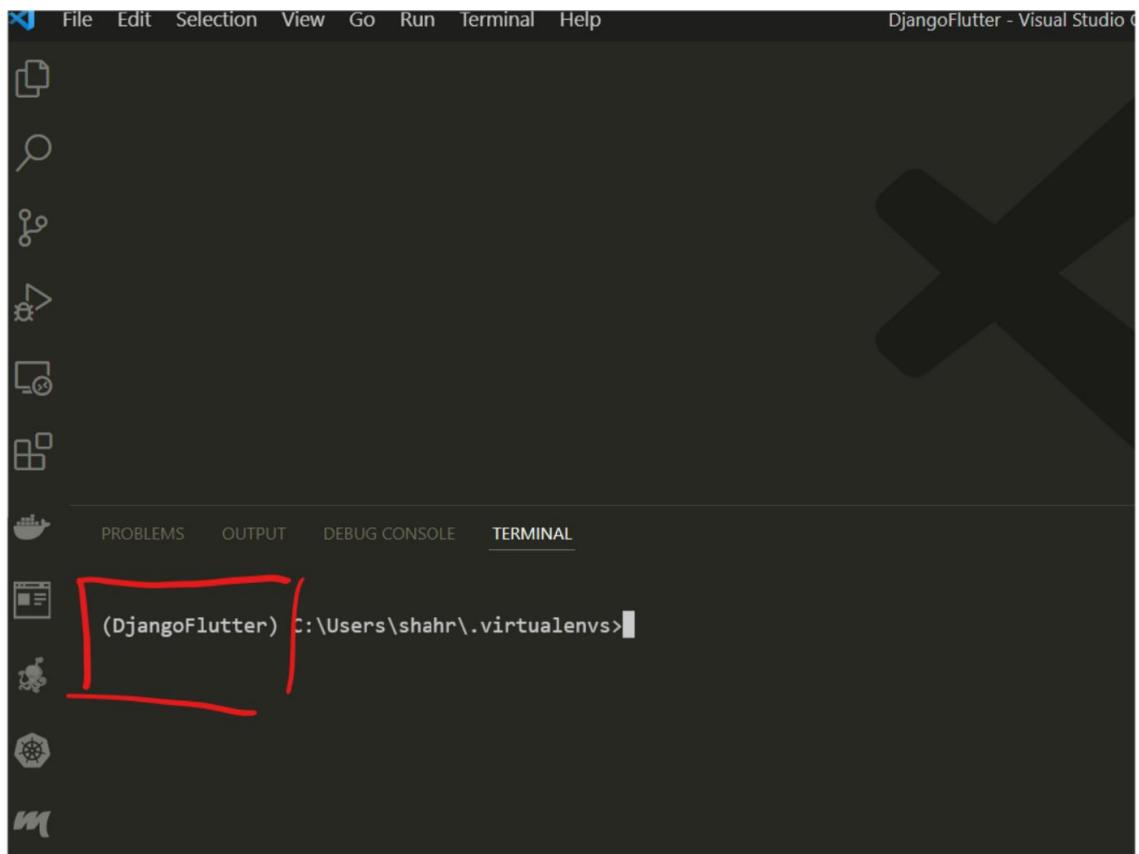
Mode	LastWriteTime	Length	Name
d----	12/29/2021 11:44 PM		.virtualenvs-gd_vKetc
d----	1/18/2022 8:07 PM		CleverTechMemesWebsite-8KK8un-w
d----	12/29/2021 11:41 PM		Coding-gZLDTjwD
d----	12/29/2021 9:47 PM		Coding-STKYX0xz
d----	5/19/2021 11:23 PM		CompetitorAnalytics-c2au3DL3
d----	3/9/2022 8:34 PM		DjangoFlutter-hgmVWtnz

Seeing the Created Virtual Environment

You can see all your virtual environments, you can now activate your virtual environment by using “.\” like the following picture. Please pay attention that there is a **Scripts** folder inside as well as an **activate** file so you need to run the following command (On other operating systems like MAC it may be slightly different); however, you may still face the following error as you may be on **powershell** and you need to switch to **command prompt (cmd)**:

```
PS C:\Users\shahr\virtualenvs> .\DjangoFlutter-hgwWtnz\Scripts\activate
.\DjangoFlutter-hgwWtnz\Scripts\activate: The term '.\DjangoFlutter-hgwWtnz\Scripts\activate' is not recognized as the name of a cmdlet, function, script file, or oper
C:\Users\shahn\virtualenvs\ DjangoFlutter-hgwWtnz\Scripts\activate.ps1 is not digitally signed. You cannot run this script on the current system. For more
information about running scripts and setting execution policy, see about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkId=135170.
At line:1 char:1
+ .\DjangoFlutter-hgwWtnz\Scripts\activate
+ ~~~~~
+ CategoryInfo          : SecurityError: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\shahr\virtualenvs>
```

Now, it should be possible to see that the virtual environment is activated:



Activating the virtual environment

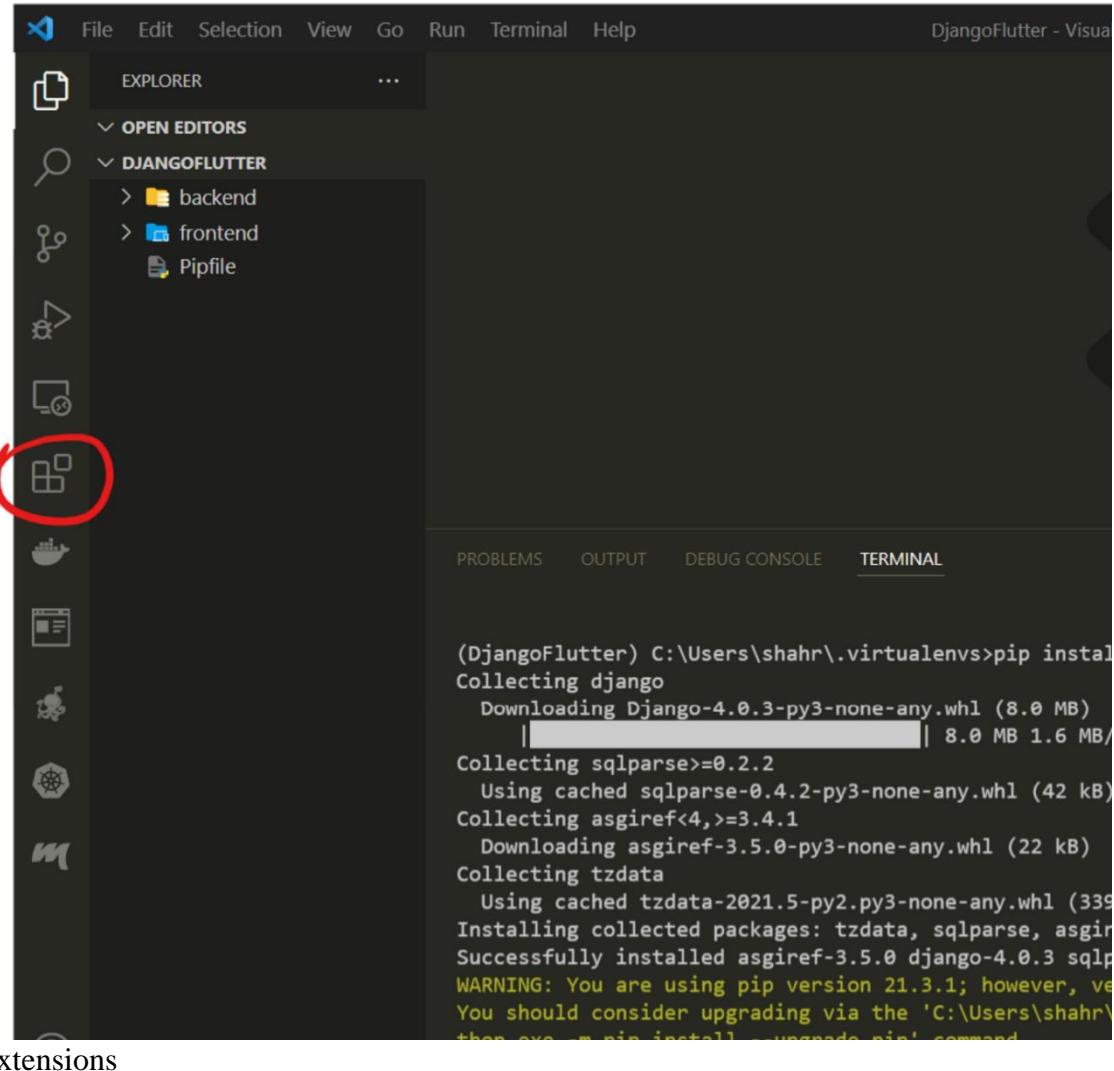
We can install **django** by using “**pip install django**” as follows:

```
(DjangoFlutter) C:\Users\shahr\.virtualenvs>pip install django
Collecting django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
    |████████| 8.0 MB 1.6 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata
  Using cached tzdata-2021.5-py2.py3-none-any.whl (339 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.3 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\shahr\.virtualenvs\ DjangoFlutter\Scripts\upgrade pip' command.

(DjangoFlutter) C:\Users\shahr\.virtualenvs>
```

pip install django

Click on the extensions and install the useful extensions including “Awesome Flutter Snippets”, “Dart”, “Flutter”, “Prettier”, “Python”, “Django”



Extensions

## Creating And Running the Django Backend

Let us switch to the backend folder, and create a *django* project named **backendapp** using “**django-admin startproject backendapp .**”:

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure under "DJANGOFLUTTER":
  - backend
  - backendapp
    - \_\_init\_\_.py
    - asgi.py
    - settings.py
    - urls.py
    - wsgi.py
    - manage.py
  - frontend
  - Pipfile
- TERMINAL:** Displays the command-line output of a pip install and a django-admin startproject command.

```
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata
  Using cached tzdata-2021.5-py2.py3-none-any.whl (339 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.3 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmWtnz\Scripts\python.exe -m pip install --upgrade pip' command.

(DjangoFlutter) C:\Users\shahr\.virtualenvs>d:
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter>cd backend
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter\backend>django-admin startproject backendapp
.
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter\backend>[ ]
```

django-admin startproject backendapp .

The beauty of *Django* is the possibility to create various apps for different purposes e.g., user management. Let us create an app named **backendcore** by using “**python manage.py startapp backendcore**”

The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure under "DJANGOFLUTTER".
  - "backend" folder contains "backendapp", "\_\_pycache\_\_", "\_init\_.py", "asgi.py", "settings.py", "urls.py", and "wsgi.py".
  - "backendcore" folder (highlighted with a red box) contains "migrations", "\_init\_.py", "admin.py", "apps.py", "models.py", "tests.py", "views.py", and "manage.py".
  - "frontend" folder.
  - Pipfile.
- TERMINAL:** Displays the command-line output of a pip upgrade and a Django command.

```
Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata
  Using cached tzdata-2021.5-py2.py3-none-any.whl (339 kB)
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.5.0 django-4.0.3 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmWtnz\Scripts\python.exe -m pip install --upgrade pip' command.

(DjangoFlutter) C:\Users\shahr\.virtualenvs>d:
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter>cd backend
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter\backend>django-admin startproject backendapp
.
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter\backend>python manage.py startapp backendcore
```

Creating an app

Now, we need to add the created app to the list of installed apps in Django. So, we can do it by referring to the ***apps.py*** file inside the ***backendcore***, and using the ***backendcoreconfig*** function

from django.apps import AppConfig

class BackendcoreConfig(AppConfig):
 default\_auto\_field = 'django.db.models.BigAutoField'
 name = 'backendcore'

The terminal shows the output of a pip upgrade command:

```

Downloading asigref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata
  Using cached tzdata-2021.5-py2.py3-none-any.whl (339 kB)
Installing collected packages: tzdata, sqlparse, asigref, django
Successfully installed asigref-3.5.0 django-4.0.3 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the 'C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmvn.exe -m pip install --upgrade pip' command.

(DjangoFlutter) C:\Users\shahr\.virtualenvs>d
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter>cd backend
(DjangoFlutter) D:\CleverTechMemes\Medium\ DjangoFlutter\backend>django-admin startpro

```

Now, it is needed to add this line to the ***settings.py*** file which is the central setting of the whole application:

The code editor shows the following Python code in 'settings.py':

```

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

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Internal
    'backendcore.apps.BackendcoreConfig'
]

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

```

It is needed to migrate the changes by using “**python manage.py migrate**”:

```
(DjangoFlutter) D:\CleverTechMemes\Medium\DjangoFlutter\backend>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, 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 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

(DjangoFlutter) D:\CleverTechMemes\Medium\DjangoFlutter\backend>
```

Migrating the changes

It is the time to see that we can run the Django server by writing “**python manage.py runserver**”:

The screenshot shows a code editor interface with a sidebar displaying the project structure. The structure includes files like settings.py, urls.py, wsgi.py, and several migration files (0001\_initial through 0012\_alter\_user\_first\_name\_max\_length). The terminal window at the bottom is running the command "python manage.py runserver". The output shows the migrations are applied successfully, followed by a system check and the start of the development server.

```
settings.py
urls.py
wsgi.py
backendcore
  _pycache_
  migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
    db.sqlite3
    manage.py
frontend
Pipfile

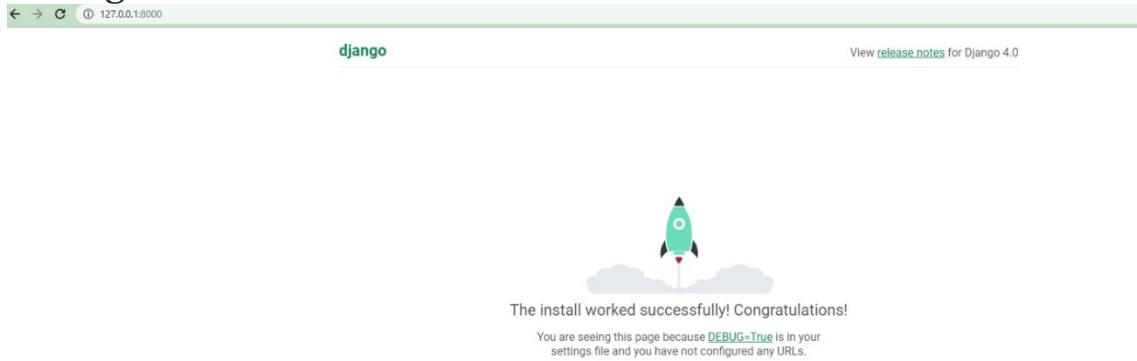
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... 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

(DjangoFlutter) D:\CleverTechMemes\Medium\DjangoFlutter\backend>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 09, 2022 - 21:39:10
Django version 4.0.3, using settings 'backendapp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

If one goes to the browser and write `127.0.0.1:8000`, s/he can see that the *Django* is up and running and you can access it through the browser:



Django is up and running

## Improving the Django Setup

It is time to improve the Django setup, and it is important to keep in mind three main things:

1. **Timezone**: You need to adjust it to your timezone.
2. **Secret\_key**: You should not reveal it to the public, otherwise, it can endanger the security of your application.
3. **DEBUG mode**: You need to change it when moving to production but for development being in *DEBUG mode* is fine.

```

backend > backendapp > settings.py > ...
6  For more information on this file, see
7  https://docs.djangoproject.com/en/4.0/topics/settings/
8
9  For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/4.0/ref/settings/
11 """
12
13 from pathlib import Path
14
15 # Build paths inside the project like this: BASE_DIR / 'subdir'.
16 BASE_DIR = Path(__file__).resolve().parent.parent
17
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/
21
22 # SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = [REDACTED]
23
24
25 # SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
26
27
28 ALLOWED_HOSTS = []
29
30 # Application definition
31
32 INSTALLED_APPS = [
33     'django.contrib.admin',
34     'django.contrib.auth',
35

```

Never reveal the SECRET\_KEY

```

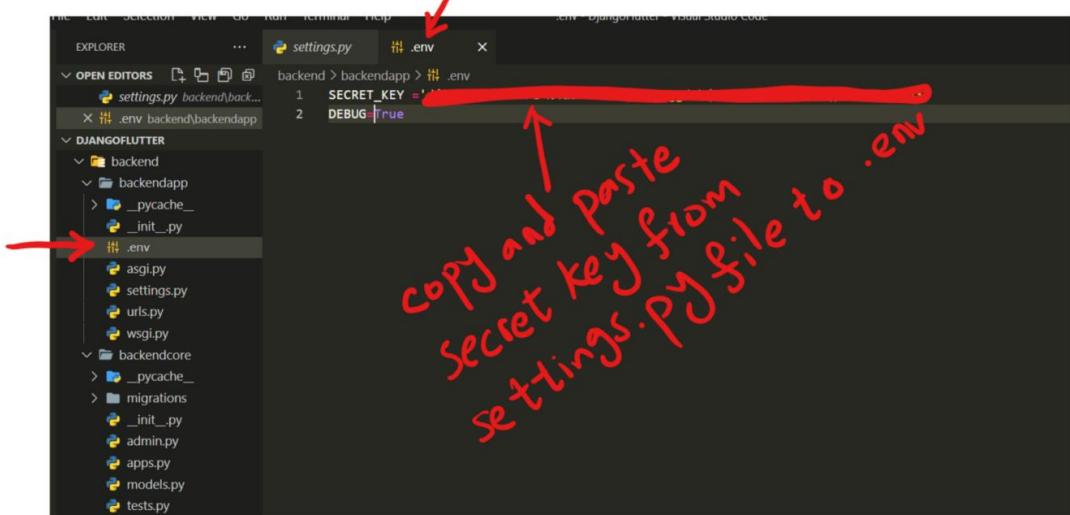
DEBUG = env('DEBUG')
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Europe/London'
USE_I18N = True
USE_TZ = True

```

Adapt time zone

Let us explore how it is possible to consider the above matters. It is a best practice to consider .env file and keep the environmental variables like **secret\_key** in that file and avoid pushing it to the public repositories. Therefore, let us

create .env file and copy and paste the **secret\_key** from the **settings.py** to the **.env** file:



Creation of .env file

Now, we need to import these variables to the **settings.py** but for doing that, we need to install **djangoenviron** by using “**pip install djangoenviron**”:

```
(DjangoFlutter) D:\CleverTechMemes\Medium\DjangoFlutter\backend>pip install django-environ  
Collecting django-environ  
  Using cached django_environ-0.8.1-py2.py3-none-any.whl (17 kB)  
Installing collected packages: django-environ  
Successfully installed django-environ-0.8.1  
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.  
You should consider upgrading via the 'C:\Users\shahr\.virtualenvs\ DjangoFlutter-hgmVWtnz\Scripts\python.exe -m pip install --upgrade pip' command.  
(DjangoFlutter) D:\CleverTechMemes\Medium\DjangoFlutter\backend>
```

pip install djangoenviron

It is needed to add these lines of codes specified in red to import the variables from the **.env** file and keep the environmental variables in a central and safe place:

```
from pathlib import Path
import environ

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

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/
#
# SECURITY WARNING: keep the secret key used in production secret!
env = environ.Env()
environ.Env.read_env()

SECRET_KEY = env('SECRET_KEY')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = env('DEBUG')

ALLOWED_HOSTS = []

# Application configuration
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'backendapp',
]

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

ROOT_URLCONF = 'backendapp.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
            ],
        },
    },
]

WSGI_APPLICATION = 'backendapp.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validation
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',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.0/ref/settings/#static-root
STATIC_ROOT = BASE_DIR / 'static'
STATIC_URL = 'static/'

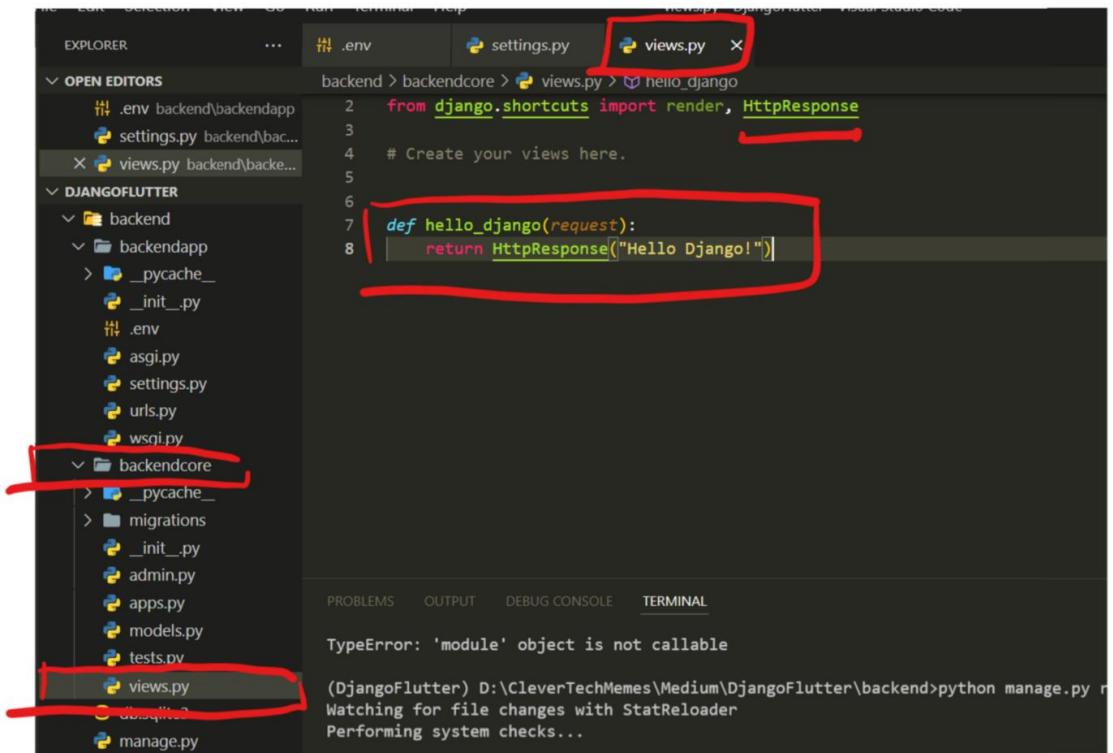
# Default primary key field type
# https://docs.djangoproject.com/en/4.0/ref/settings/#primary-key-field-type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Importing environmental variables

You can see in above picture that we can still run the server properly.

## Writing a Simple View and Routing in Django

Let us now write a hello world view in *Django*: For doing that, it is needed to open **views.py** in the **backendcore**, and write the following function

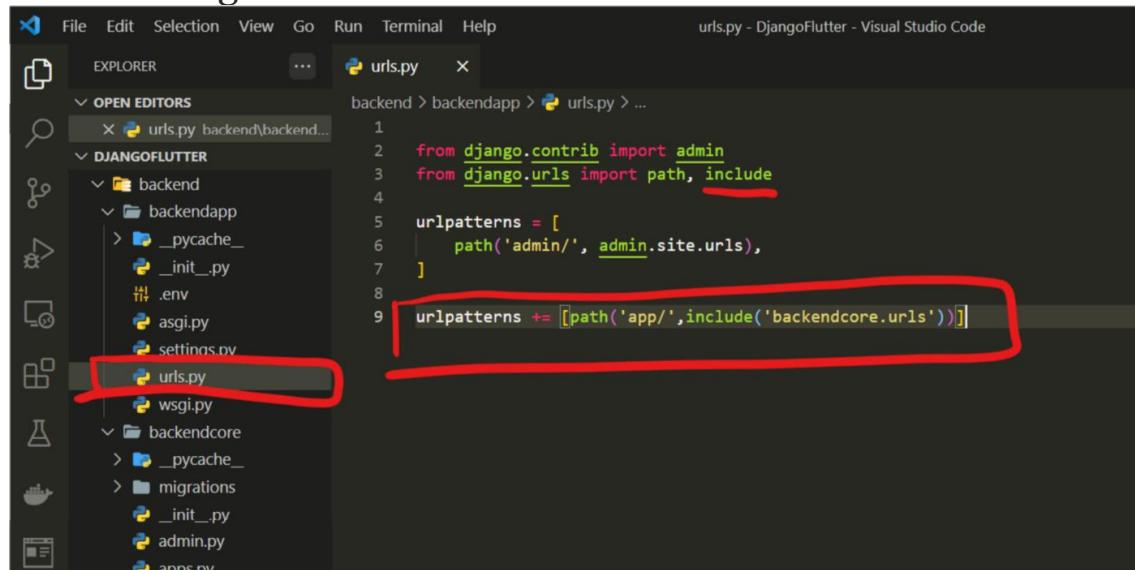


```
from django.shortcuts import render, HttpResponseRedirect
# Create your views here.

def hello_django(request):
    return HttpResponseRedirect(["Hello Django!"])
```

Writing a simple view

It is needed to route the incoming requests to the defined view, so we can edit the **urls.py** inside **backendapp** folder and add the following:



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

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

This means that paths with '**app/**' will be routed to the **backendcore**; however, there does not yet exist

a ***urls.py*** file, so, let us create a ***urls.py*** inside the ***backendcore***:

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a folder named 'backendcore' which contains a 'urls.py' file. A red arrow points from the text 'create urls.py file' to this 'urls.py' file. In the main editor area, there is another 'urls.py' file open under the path 'backendcore > urls.py'. This file contains the following Python code:

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

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

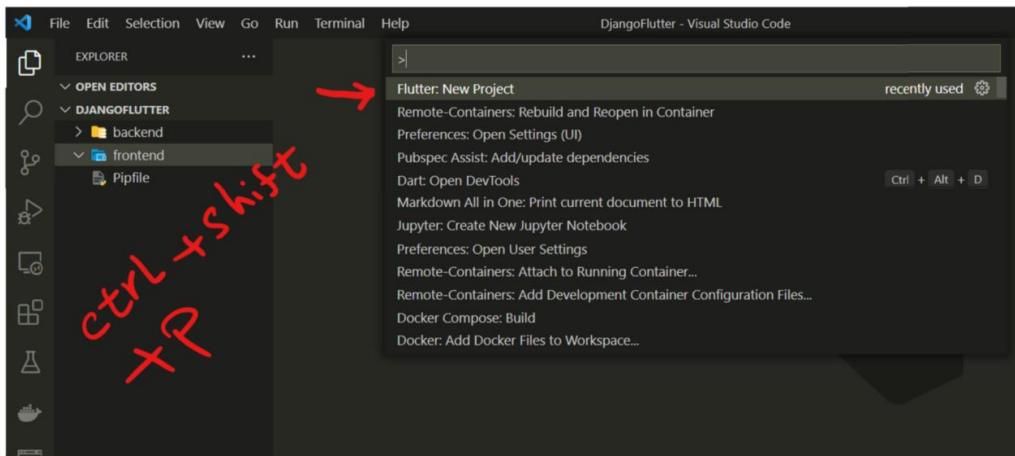
A red box highlights the code in the editor, and a red arrow points from the text 'writing these inside the file' to this highlighted code.

Creating the *urls.py* inside *backendcore*

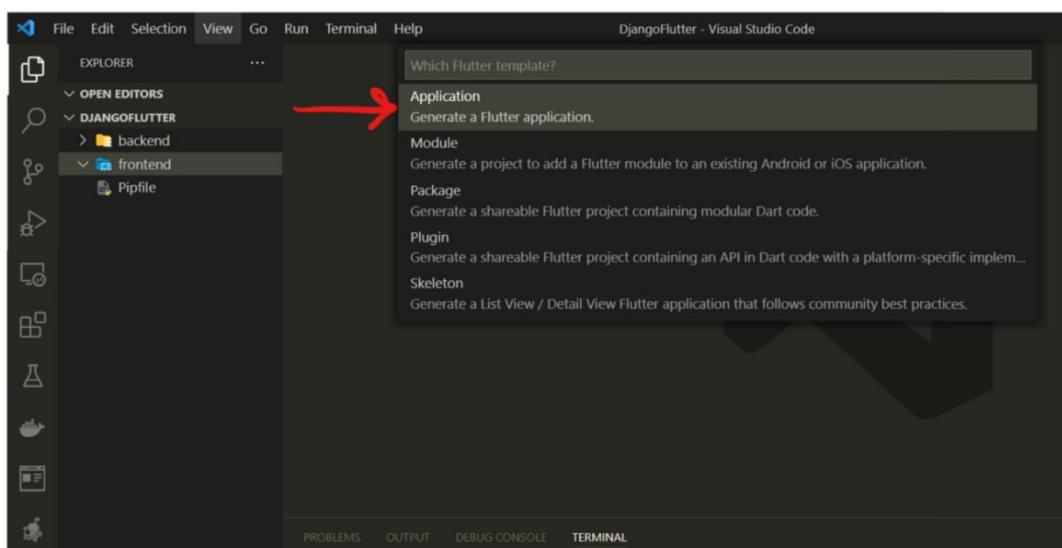
You should be able still to start the server, and now it is time to add the *Flutter* frontend

## Creating Flutter Frontend

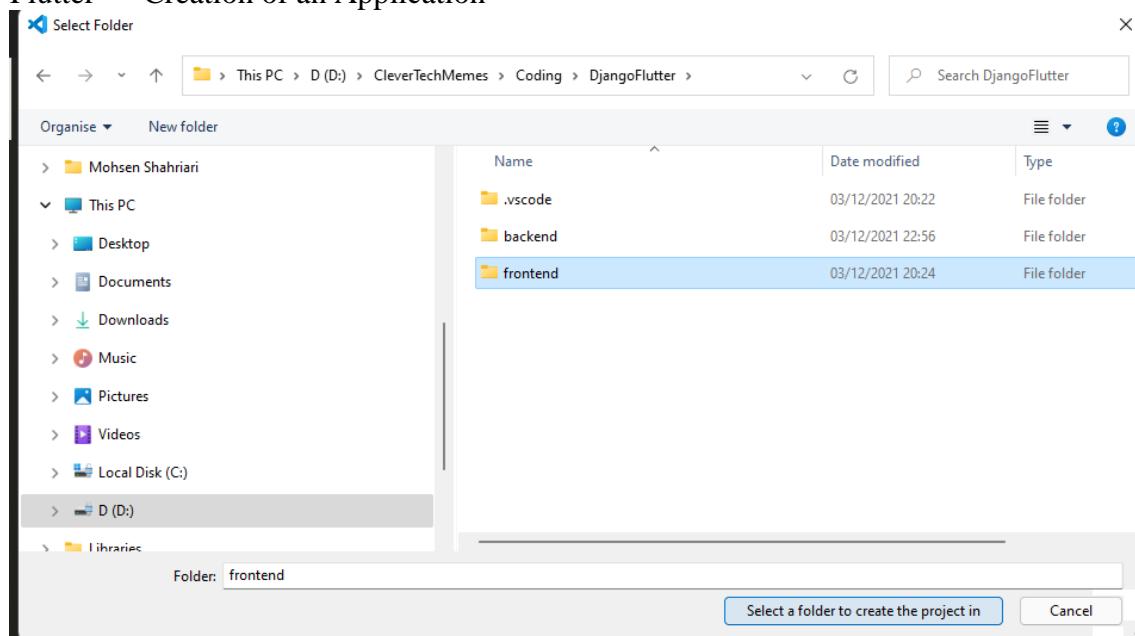
You need to first install Dart and Flutter — I assume it has been done. We can create a Flutter project in VS CODE using “***CTRL + SHIFT + P***” and selecting “***Create Flutter Application***”, and then select the folder for creating the flutter application



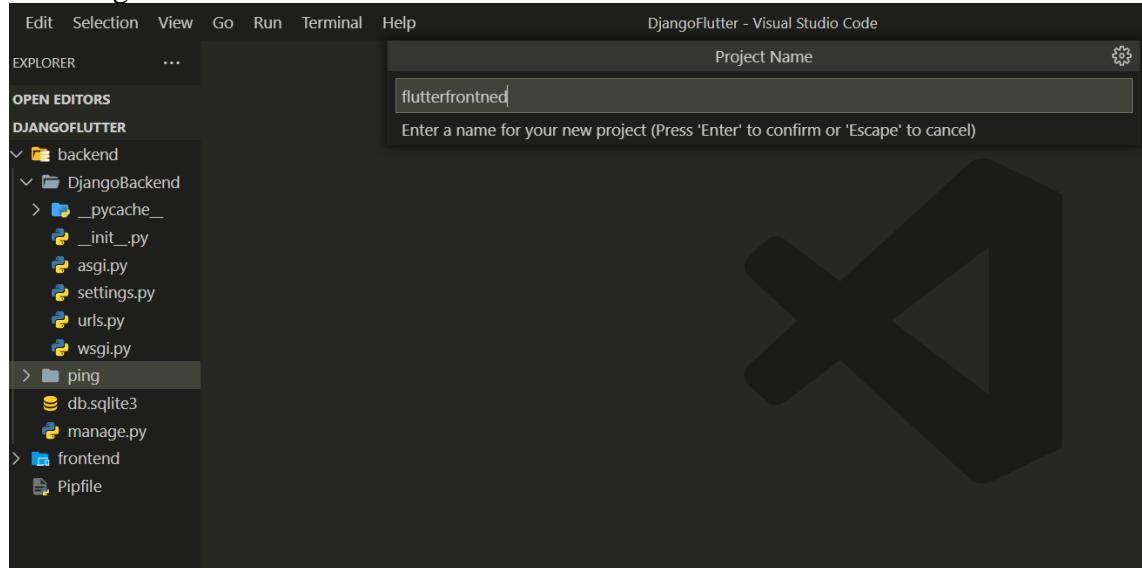
## Flutter: New Project



## Flutter — Creation of an Application

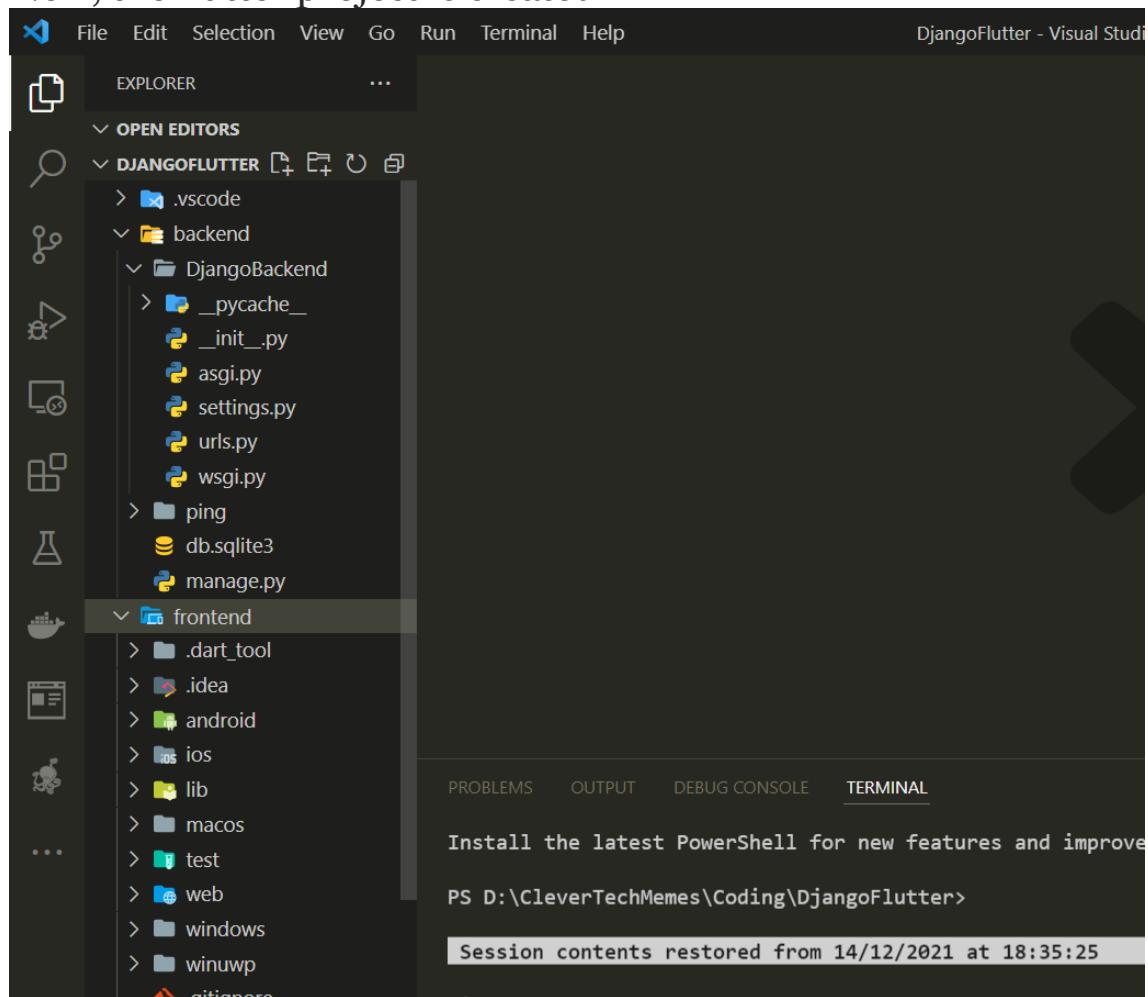


## Selecting the folder

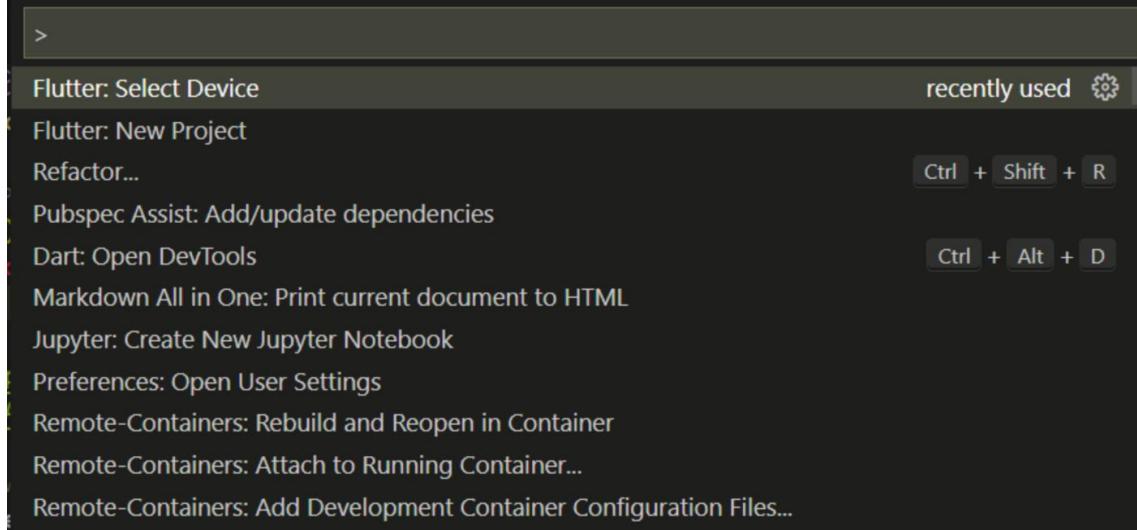


Name of the Flutter app

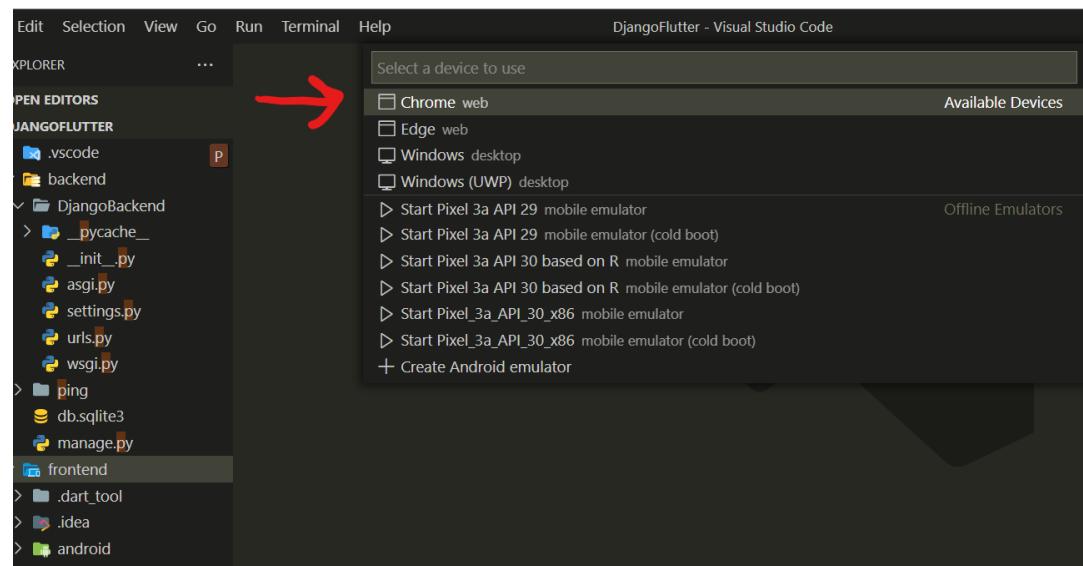
Now, the flutter project is created



we can select the device, and start running our first *Flutter* application:



### Select device



Selecting Chrome Web

Run Terminal Help

Start Debugging F5

Run Without Debugging Ctrl+F5

Stop Debugging Shift+F5

Restart Debugging Ctrl+Shift+F5

---

Open Configurations

Add Configuration...

---

Step Over F10

Step Into F11

Step Out Shift+F11

Continue F5

---

Toggle Breakpoint F9

New Breakpoint >

---

Enable All Breakpoints

Disable All Breakpoints

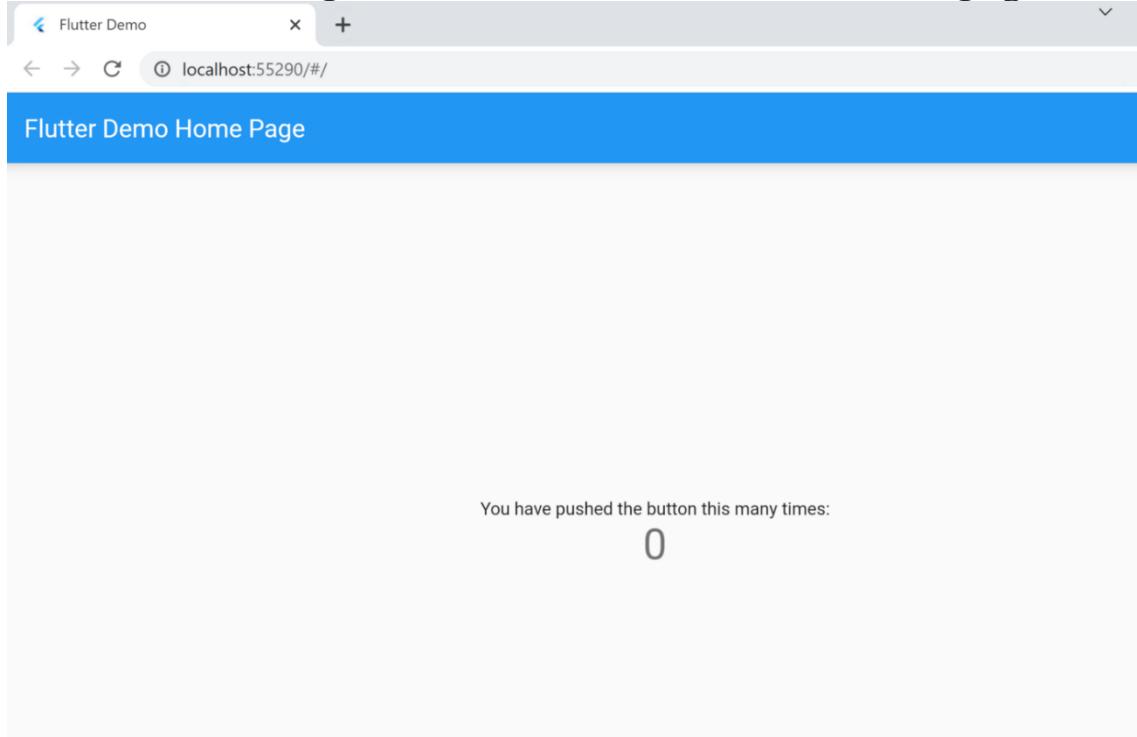
Remove All Breakpoints

---

Install Additional Debuggers...

Run without debuggin

Now, it should be possible to see the flutter demo web page:

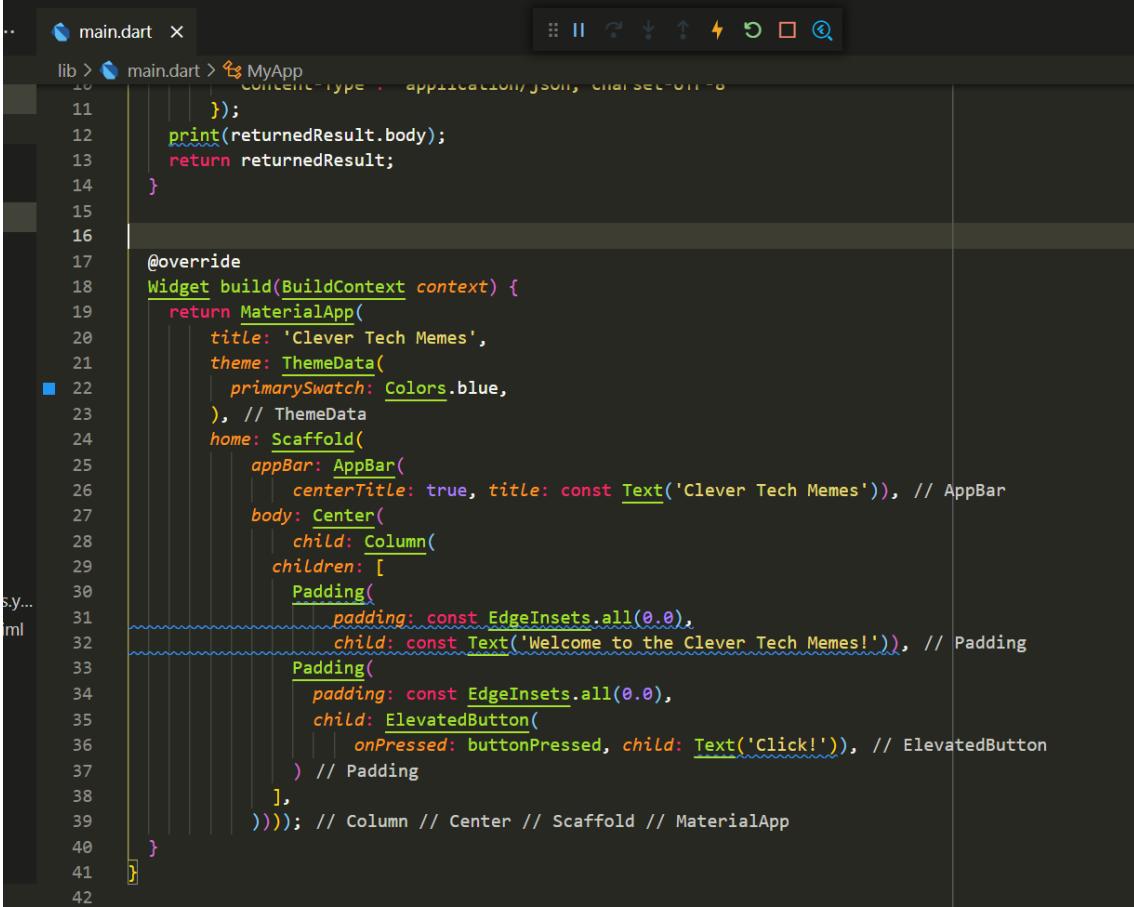


Flutter demo page

Now, let us delete the content of the **main.dart** file inside the **lib** folder, and write the following content:

```
lib > main.dart > MyApp
1 import 'package:flutter/material.dart';
2 import 'package:http/http.dart' as http;
3 void main() {runApp(const MyApp());}
4 class MyApp extends StatelessWidget {
5   const MyApp({Key? key}) : super(key: key);
6   Future<http.Response> buttonPressed() async {
7     http.Response returnedResult = await http.get(
8       Uri.parse('http://localhost:8000/app/hellodjango'),
9       headers: <String, String>{
10         'Content-Type': 'application/json; charset=UTF-8'
11       });
12     print(returnedResult.body);
13     return returnedResult;
14 }
```

### Content of main.dart



```
lib > main.dart > MyApp
  ...
10     });
11   );
12   print(returnedResult.body);
13   return returnedResult;
14 }
15
16
17 @override
18 Widget build(BuildContext context) {
19   return MaterialApp(
20     title: 'Clever Tech Memes',
21     theme: ThemeData(
22       primarySwatch: Colors.blue,
23     ), // ThemeData
24     home: Scaffold(
25       appBar: AppBar(
26         centerTitle: true, title: const Text('Clever Tech Memes')), // AppBar
27       body: Center(
28         child: Column(
29           children: [
30             Padding(
31               padding: const EdgeInsets.all(0.0),
32               child: const Text('Welcome to the Clever Tech Memes!')), // Padding
33             Padding(
34               padding: const EdgeInsets.all(0.0),
35               child: ElevatedButton(
36                 onPressed: buttonPressed, child: Text('Click!')), // ElevatedButton
37             ) // Padding
38           ],
39         )));
40       )); // Column // Center // Scaffold // MaterialApp
41 }
42 }
```

### Content of main.dart

In this tutorial, the goal is to prepare boilerplate program and the way to make Django and Flutter work so I assume, the reader is familiar with the Flutter syntax. However, the above code adds bar, a welcome message and a button.

Also, **buttonPressed** function sends a *GET* request to the *Django* backend as soon as the button is pressed. So, here there is not much coding on Flutter side but only the necessary things are brought for the communication and demonstration purposes.

## Communication Between Backend and Frontend

For the communication between the Django backend and Flutter Frontend, it is needed to add *http* dependency to *Flutter*, so we can write:

```
main.dart 1
macos
test
web
windows
winuwp
.gitignore
.metadata
.packages
analysis_options.yaml
flutterfrontend.iml
pubspec.lock
pubspec.yaml
README.md

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

PS D:\CleverTechMemes\Medium\DjangoFlutter\frontend\flutterfrontend> flutter pub add http
Resolving dependencies...
collection 1.15.0 (1.16.0 available)
+ http 0.13.4
+ http_parser 4.0.0
path 1.8.0 (1.8.1 available)
source_span 1.8.1 (1.8.2 available)
test_api 0.4.3 (0.4.9 available)
vector_math 2.1.0 (2.1.2 available)
Changed 2 dependencies!
PS D:\CleverTechMemes\Medium\DjangoFlutter\frontend\flutterfrontend>
```

flutter pub add http

It is now possible to import the *http* library by writing **import ‘package:http/http.dart’ as http** and send a request using **http.get()** in *Flutter* as follows:

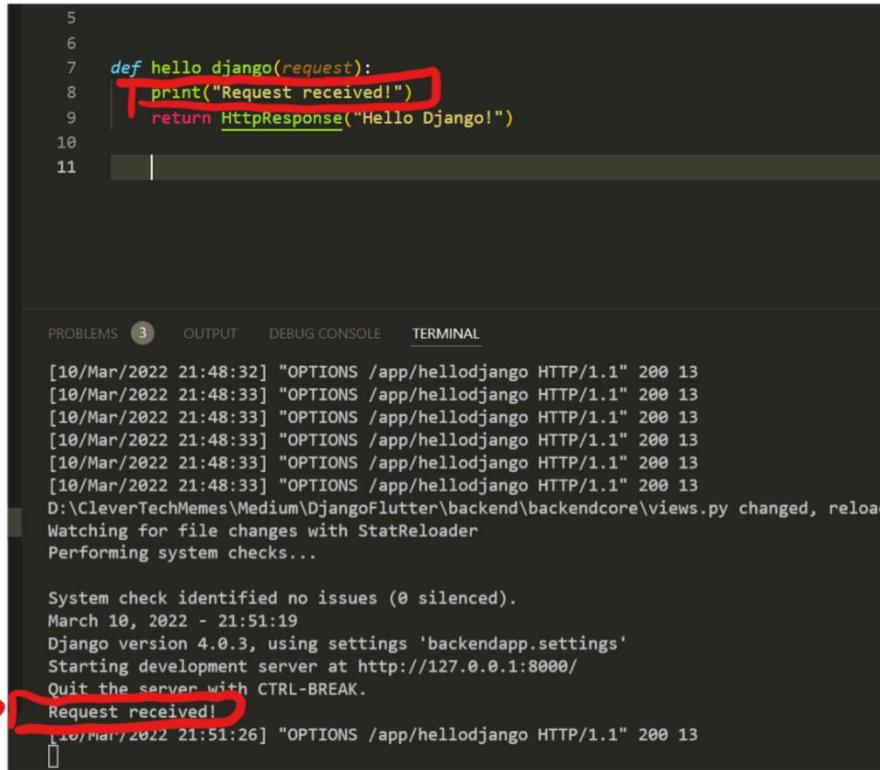
The screenshot shows a Flutter application named 'MyApp' in the main.dart file. The code includes a main function and a MyApp class. A red box highlights the buttonPressed() method, and handwritten text says 'add these lines' with an arrow pointing to it.

```
lib > main.dart > MyApp
1 import 'package:flutter/material.dart';
2 import 'package:http/http.dart' as http;
3
4 void main() {
5   runApp(const MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({Key? key}) : super(key: key);
10
11
12 Future<http.Response> buttonPressed() async {
13   http.Response returnedResult = await http.get(
14     Uri.parse('http://localhost:8000/app/helldjango'),
15     headers: <String, String>{
16       'Content-Type': 'application/json; charset=UTF-8'
17     });
18   print(returnedResult.body);
19   return returnedResult;
20 }
21 }
```

Sending a request when button is clicked

As soon as we run Django server by writing “***python manage.py runserver***”, and we click the button in the browser, we can see that we receive requests in *Django i.e., 200* is generated which means that the request is successfully received.

We can still make sure by writing in the *Django* view a printing statement like “request received” and figure out if the request can be printed, and it is working as follows:



```
5
6
7 def hello_django(request):
8     print("Request received!")
9     return HttpResponse("Hello Django!")
10
11

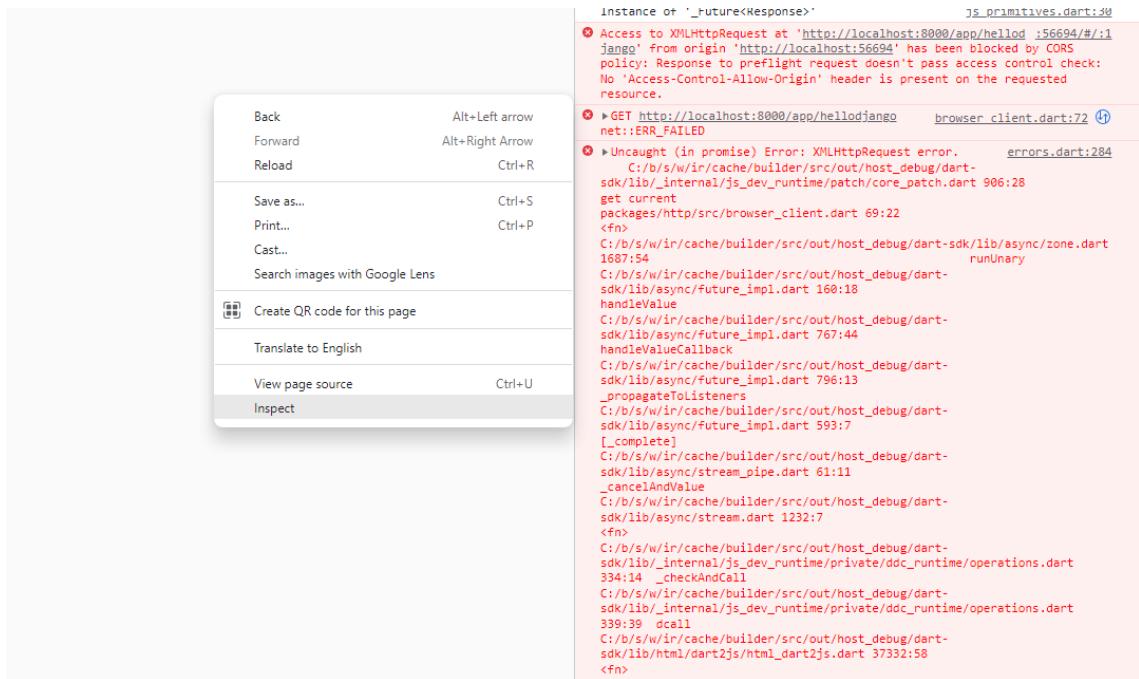
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL

[10/Mar/2022 21:48:32] "OPTIONS /app/helldjango HTTP/1.1" 200 13
[10/Mar/2022 21:48:33] "OPTIONS /app/helldjango HTTP/1.1" 200 13
D:\CleverTechMemes\Medium\DjangoFlutter\backend\backendcore\views.py changed, reload
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 10, 2022 - 21:51:19
Django version 4.0.3, using settings 'backendapp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
→ Request received!
[10/Mar/2022 21:51:26] "OPTIONS /app/helldjango HTTP/1.1" 200 13
```

Adding a print

However, if we right click on the frontend page and inspect element, we can see that we receive the *CORS Origin Error* as the *Django* does not know *Flutter* as a reliable frontend to communicate.



## Resolving Django Cross Site Origin And Using REST

To resolve this issue, let us install *Django Rest Framework* by writing the “**pip install django-rest-framework**”, and add the rest framework to the installed apps:

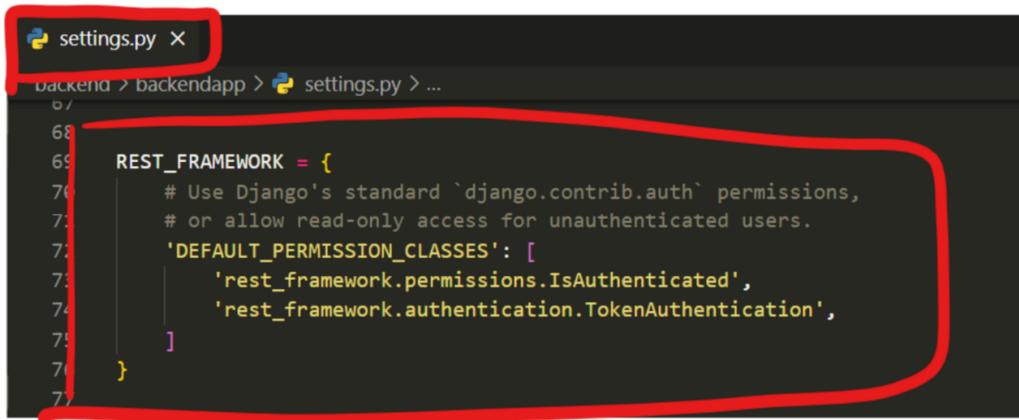
The screenshot shows a code editor displaying a file named `settings.py`. A red box highlights the file name. Below it, the file path is shown as `backend > backendapp > settings.py > ...`. The code in the file is:

```
25     INSTALLED_APPS = [
26         'django.contrib.admin',
27         'django.contrib.auth',
28         'django.contrib.contenttypes',
29         'django.contrib.sessions',
30         'django.contrib.messages',
31         'django.contrib.staticfiles',
32         # Internal
33         'backendcore.apps.BackendcoreConfig'
34         #external
35         'rest_framework'
36     ]
37 
```

A red box also highlights the line `'rest_framework'`.

Adding ‘rest\_framework’ to the setting

And add the following to the **settings.py** file as follows:



```
settings.py x
Backend > backendapp > settings.py > ...
68
69 REST_FRAMEWORK = {
70     # Use Django's standard `django.contrib.auth` permissions,
71     # or allow read-only access for unauthenticated users.
72     'DEFAULT_PERMISSION_CLASSES': [
73         'rest_framework.permissions.IsAuthenticated',
74         'rest_framework.authentication.TokenAuthentication',
75     ]
76 }
77 
```

Let us install the **simple-jwt** and the **django cors headers**.

We require **simple-jwt**, to activate the token authentication in Django and we require **cors-origin** because we need to communicate with an external frontend like Flutter, so let us install these libraries by writing:

- **pip install django-cors-headers**
- **pip install djangorestframework-simplejwt**

In **REST\_Framework**, it is needed to add **simplejwt as follows:**

A screenshot of a code editor showing the `settings.py` file. A red box highlights the `REST_FRAMEWORK` section. A blue arrow points from the text `# commented` to the line `# 'rest_framework.authentication.TokenAuthentication'`. A red box highlights the `'DEFAULT_AUTHENTICATION_CLASSES'` line.

```
68
69 REST_FRAMEWORK = {
70     # Use Django's standard `django.contrib.auth` permissions,
71     # or allow read-only access for unauthenticated users.
72     'DEFAULT_PERMISSION_CLASSES': [
73         'rest_framework.permissions.IsAuthenticated',
74         # 'rest_framework.authentication.TokenAuthentication',
75     ],
76
77     'DEFAULT_AUTHENTICATION_CLASSES': (
78         'rest_framework_simplejwt.authentication.JWTAuthentication',
79     )
80 }
```

Also, we need to add the ***SIMPLE\_JWT*** configuration as shown below:

A screenshot of a code editor showing the `settings.py` file. A large red checkmark and the handwritten note `add these to the settings.py` are placed next to the `SIMPLE_JWT` section. A red box highlights the entire section.

```
90 }
91
92 from datetime import timedelta
93
94 SIMPLE_JWT = {
95     'AUTH_HEADER_TYPES': ('JWT',),
96     'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
97     'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
98 }
99 # Password validation
100 # https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators
101 |
```

We need to add ***corsheaders*** to the installed apps:

A screenshot of a Visual Studio Code editor window titled "settings.py - DjangoFlutter - Visual Studio Code". The file path is "backend > Backendapp > settings.py". The code shows the "INSTALLED\_APPS" section:

```
25 INSTALLED_APPS = [
26     'django.contrib.admin',
27     'django.contrib.auth',
28     'django.contrib.contenttypes',
29     'django.contrib.sessions',
30     'django.contrib.messages',
31     'django.contrib.staticfiles',
32     # Internal
33     'backendcore.apps.BackendcoreConfig'
34     #external
35     'rest_framework',
36     'corsheaders'
37 ]
```

The line "INSTALLED\_APPS = [ ... ]" is circled in red. The line "corsheaders" is also circled in red.

Still, we need to add the following to the **MIDDLEWARE** in **settings.py** as follows:

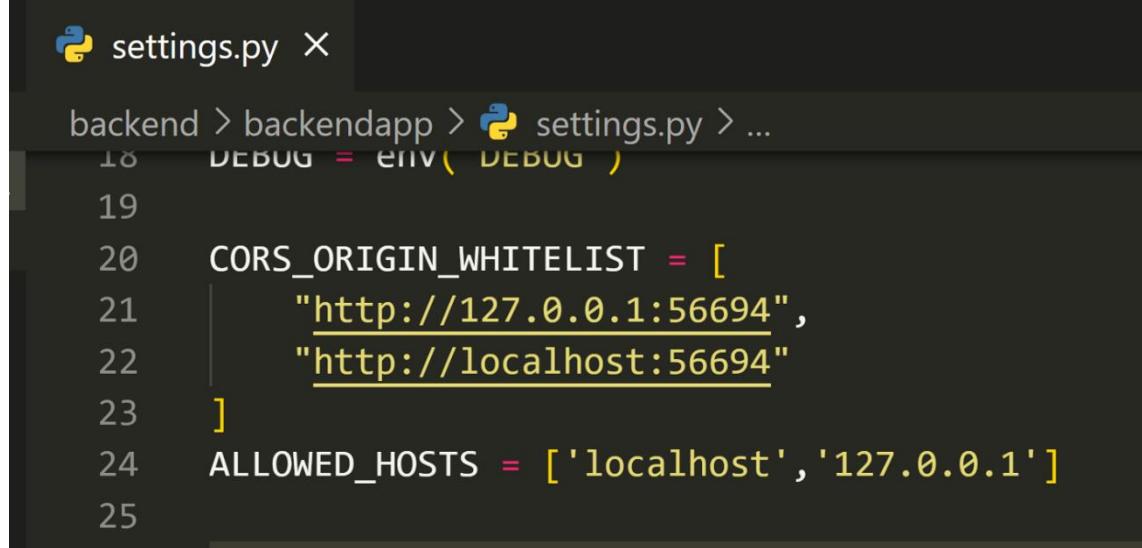
A screenshot of a Visual Studio Code editor window titled "settings.py - DjangoFlutter - Visual Studio Code". The file path is "backend > Backendapp > settings.py". The code shows the "MIDDLEWARE" section:

```
35     'rest_framework',
36     'corsheaders'
37 ]
38
39 MIDDLEWARE = [
40     'django.middleware.security.SecurityMiddleware',
41     'django.contrib.sessions.middleware.SessionMiddleware',
42     'django.middleware.common.CommonMiddleware',
43     'django.middleware.csrf.CsrfViewMiddleware',
44     'django.contrib.auth.middleware.AuthenticationMiddleware',
45     'django.contrib.messages.middleware.MessageMiddleware',
46     'django.middleware.clickjacking.XFrameOptionsMiddleware',
47
48     'corsheaders.middleware.CorsMiddleware',
49 ]
50 ]
```

The line "MIDDLEWARE = [ ... ]" is circled in red. The line "corsheaders.middleware.CorsMiddleware" is also circled in red.

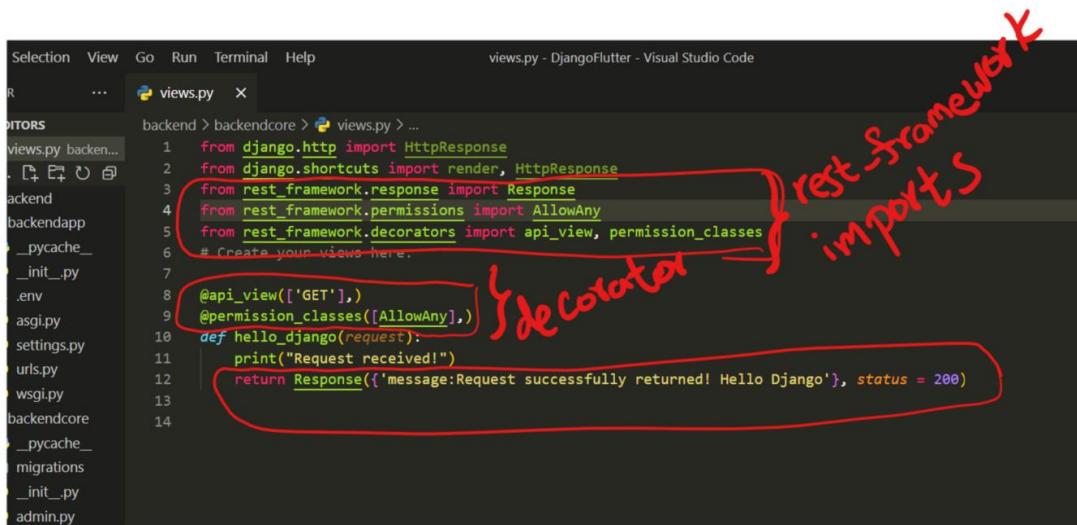
Flutter needs to be allowed to communicate with Django so we need to add localhost to the list of allowed hosts and

the ***CORS\_ORIGIN\_WHITELIST*** sending request to backend in ***settings.py***:



```
settings.py X
backend > backendapp > settings.py > ...
18 DEBUG = env('DEBUG')
19
20 CORS_ORIGIN_WHITELIST = [
21     "http://127.0.0.1:56694",
22     "http://localhost:56694"
23 ]
24 ALLOWED_HOSTS = ['localhost', '127.0.0.1']
25
```

Let us use the ***Response*** from the ***rest\_framework*** to slightly change the ***views.py*** file and send a proper message back to the client as follows:



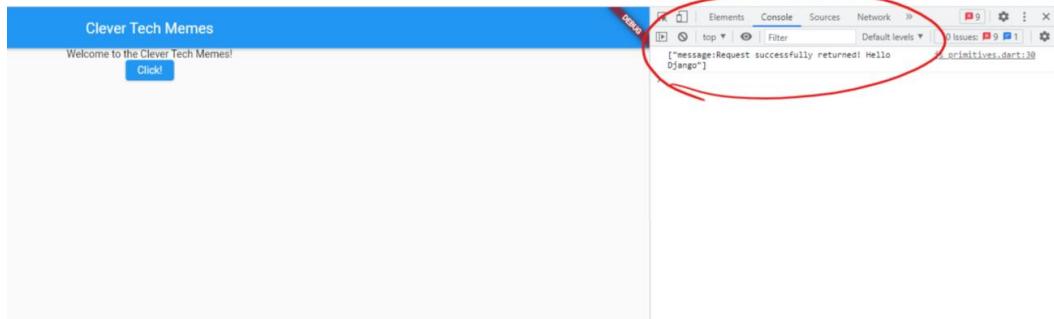
```
views.py - DjangoFlutter - Visual Studio Code
Selection View Go Run Terminal Help
views.py X
backend > backendcore > views.py > ...
1 from django.http import HttpResponse
2 from django.shortcuts import render, HttpResponseRedirect
3 from rest_framework.response import Response
4 from rest_framework.permissions import AllowAny
5 from rest_framework.decorators import api_view, permission_classes
6 # Create your views here.
7
8 @api_view(['GET'])
9 @permission_classes([AllowAny])
10 def hello_django(request):
11     print("Request received!")
12     return Response({'message': 'Request successfully returned! Hello Django'}, status = 200)
13
14
```

The screenshot shows the `views.py` file in Visual Studio Code. Red annotations highlight several parts of the code:

- A red bracket labeled "rest\_framework imports" covers the imports for `HttpResponse`, `render`, `RestResponse`, `AllowAny`, and `api_view`, `permission_classes`.
- A red bracket labeled "decorated" covers the `@api_view(['GET'])` and `@permission_classes([AllowAny])` decorators applied to the `hello_django` function.
- A red circle highlights the `status = 200` argument in the `return Response` statement.

Please note the we need to add decorators like ***@permission\_classes*** and ***@api\_view***. You can imagine them as wrap functions in which allow the function to process the request as well as receiving a GET request.

Now, if we click the button in the browser, we can see the successful communication between the ***Django*** and ***Flutter***:



## Summary:

*Django* and *Flutter* are two important backend and frontend technologies in which has helped developers and entrepreneurs to create useful applications. *Django* is popular due to fast-development cycle and its root in Python programming language and nowadays wide range of practitioners can use it. *Flutter* is getting momentum in the market by being platform independent and it means writing your code once and run it everywhere. In this tutorial, it is shown step-by-step how to create boilerplate *Django* and *Flutter* application that can communicate with each other.