Django

VENV

The apps break if the version of the packages used in it changes. It is a bad practice to use a global environment for a project if it'll go through future changes. If you update any package related to a particular project, then there is a chance of breaking that project. To avoid these issues we create a local virtual environment. Here, we install all the required packages separately irrelevant to the global packages. These packages versions won't be changed even after you update a particular package in the global environment. (basically, the global environment does not have any power over these packages). Unless you don't update your package in the locally created virtual environment you won't face any issues.

You can save the versions of the packages using the pip freeze function and saving it in the requirements.txt file. By doing this you can create a venv anytime later, with the same environment conditions which intern won't break the python apps that you are building.

Create a virtual environment using the following command

```
C:\Users\<user_name>\Desktop\Django\django\python -m venv <evn_name>
```

Then if you need all the packages installed by default in this env from the main env (global) use

```
C:\Users\<user_name>\Desktop\Django\django\python -m venv <evn_name> --system-site-packages
```

Use pip list to get the list of packages,
Use pip freeze to get the list of packages with the version name

You can copy the **pip freeze** command output and save it in a **requirements.txt** file
And can emulate the same virtual environment for future project changes without breaking it.
This break is caused by the change in the package versions.

To emulate the new env with the same packages and versions

```
C:\Users\<user_name>\Desktop\Django\django\pip install -r requirements.txt
```

Activate the VENV

We can manually install the packages into this env but before that, we need to be in that env or the packages will be installed globally.

```
C: \Users\<user_name>\Desktop\Django\django > venv\Scripts\activate.bat
```

Once the env is activated we can install the packages and those packages will be installed only locally (remember not globally).

You can see the installed packages using the **pip list** or **pip freeze.**

Deactivate env using **deactivate** command in the cmd.

Installing and populating the folders.

First, install the Django using **pip install Django** command

The following command shows the version of Django.

```
(venv) C: \Users\<user_name>\Desktop\Django\django\first_project > python - m django - -version
>>> 3.1.4 # output
```

The next step is to generate the required files (default files & folders)

```
(venv) C: \Users\<user_name>\Desktop\Django\django > python - m django startproject
```

Run the server.

Since all the files needed to build the app are populated, we run the code off the start
Using the following command, notice that we need to be in the <project name> folder.
That can be done by simple **cd <project name>** cmd

```
C:\Users\<user_name>\Desktop\Django\django\first_project > python manage.py runserver
```

We get this output in our cmd

The system starts a local server at the web address http://127.0.0.1:8000/

We get the same result even if we replace **127.0.0.1** with **localhost**
The launched site is just a default one that does not have anything on it but just a simple Django starter site. To stop the server after running(or to interrupt) use **Ctrl+C.**

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, content types, sessions.
Run 'python manage.py migrate' to apply them.
December 22, 2020 - 17:46:31
Django version 3.1.4, using settings 'first_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

In our URL file, we can see that there is an admin route by default. Let's append that to the **localhost:8000** as **/admin.** This route takes to the admin page aka login screen.

Routes

You can create multiple apps inside a single Django project, these apps may handle different tasks at hand, eg: one app may handle the blog part, another maybe for the payment part and etc.
To create a new app inside the already existing project use the following command(cmd)

```
(venv) C: \Users\<user-name>\Desktop\Django\django\first_project > python manage.py startapp blog
```

The above cmd is going to create a new app with the default folders and files.
The blog folder now has various files, in that view file represents the views of the web page.
Now, this app is a sub-app of the main project app, so each view needs routes. These routes take the user from the main app to the desired view.

Eg: in our sub-app, we are going to create a blog home and blog about views. Blog about is child view of blog home(sorta coz to get there we need to go to the blog home first.)

To achieve the above-mentioned goals create a new URL file in the blog app. This URL file has all the routes of the views it holds.

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path(' ', include('blog.urls')), # can use blog instead of ' '
    # to make the blog name as the home page instead of being on another page
    # make the route empty
]
```

The above code is the main route of our app, it has admin which is added by default. We will add the route for our newly created app i.e. blog app. To make it home we keep the name of the route empty. The include function directs the path of this app.

```
# the blog URLs routes
from django.urls import path
from. import views

urlpatterns = [
    path('', views.home, name='blog-home'),
    path('about/', views.about, name='blog-about'),
]
```

As mentioned above the blog URL file has all the routes of the app's views. We have a home and about views. The blog home does not any route so whenever the main apps direct to the blog this is gonna be shown. Next, we have about/ route which shows the blog about the view.

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

# Create your views here.
def home(request):
  return HttpResponse('<h1> Blog Home </h1>')

def about(request):
  return HttpResponse('<h1> Blog About </h1>')
```

Views are included in the http response method whenever the function is called from the urls file the function returns the response included in it.

Localhost aka 127.0.0.1

The localhost is a virtual server created by your own computer, When you try to access the domain(localhost), a loopback is triggered. If you access "http://localhost" in the browser, the request will not be forwarded to the internet through the router, but will instead remain in your own system. Localhost has the IP address 127.0.0.1, which refers back to your own server.

Run the app on the local server using the runserver cmd and test all the routes.

Templates

Before we have added only simple HTML text to the views. This is not ideal since real-life web pages use complex designs. Now we will try to make our web look as modern and optimal as possible. We will create custom HTML files for each view i.e for home and blog. Next, we will render those files in our views using the `render(request, 'blog/home.html', context)` function.

Created a dummy database sample of the blog content using a list of Dict. this list is then passed inside the render as context. Usually, a blog contains many samples of data so to loop over them we use the following code.

```
{% for post in posts %}
<h1>{{ post.title }}</h1>
<p>By {{ post.author }} on {{ post.date_posted }} </p>
<p>{{ post.content }}</p>
{% endfor %}
```

The code here written inside the {% code %}, the code is written in jinja format. It's a full-featured template engine for Python.

We created a base file to avoid writing boilerplate code for both the home and about HTML file. The base file has the common code that these other files need.

To run and make this code work we need to add this app (blog) to the settings of the main project as

```
INSTALLED_APPS = [
'blog.apps.BlogConfig',
]
```

Follow this series of commits for the upgrades.
Next added bootstrap fonts and colors link in the HTML files. At this point, the webpage looks good but not more modern. The side navigation bar and top bar are added in the main.css file. Other further changes are made. As you follow through with these commits you'll know the changes applied. Finally, instead of hardcoding the routes of the home and blog in navigation, we use the code. This makes our app more versatile to future changes of the URL paths.