



## BackEnd Workshop-1

---

Clarusway



### Subject: First App (Welcome to BackEnd)

---

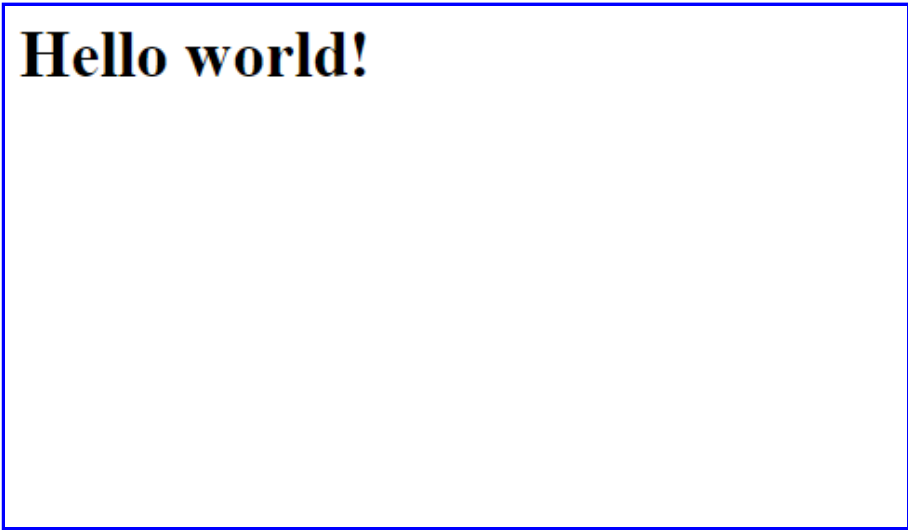
#### Learning Goal:

- Practice creating a basic Django project and app.

#### Introduction

Let's get our hands dirty! Create a Django project from scratch. This will be a basic Django project with a simple HTTP response.

The page will seem like:



**Hello world!**

## Nice to have VSCode Extentions:

Djaneiro - Django Snippets

## Needs

- Python
- pip
- virtualenv

---

## Code:

---

### Part 1 - Create first project and app

1. Create a working directory, cd to new directory.
2. Create virtual environment as a best practice:

```
python3 -m venv env # for Windows or  
python -m venv env # for Windows  
virtualenv env # for Mac/Linux or;  
virtualenv env -p python3 # for Mac/Linux
```

3. Activate scripts:

```
.\env\Scripts\activate # for Windows, may need to switch powershell on this  
operation.  
source env/bin/activate # for MAC/Linux
```

See the (env) sign before your command prompt.

4. Install django:

```
pip install django # or  
py -m pip install django
```

5. See installed python packages:

```
pip freeze
```

```
# you will see:
```

```
# asgiref==3.5.0
```

```
# Django==4.0.4
```

```
# sqlparse==0.4.2
```

```
# tzdata==2022.1
```

```
# If you see lots of things here, that means there is a problem with your virtual  
env activation.
```

```
# Activate scripts again!
```

6. Create the requirements.txt on your working directory, send your installed packages to this file, requirements file must be up to date:

```
pip freeze > requirements.txt
```

```
# In this project we will not use this file. But this is a standard procedure to  
learn.
```

7. Create project:

```
django-admin startproject clarusway .
```

```
# With . it creates a single project folder.
```

```
# Avoiding nested folders.
```

```
django-admin startproject clarusway
```

```
# Without . at the end it will create nested folders.
```

```
# Another naming as "main":
```

```
django-admin startproject main .
```

```
# Naming depends on the company, team and the project.
```

Various files has been created inside project folder "clarusway"!

- manage.py - A command-line utility that allows you to interact with your Django project
- \_\_init\_\_.py - An empty file that tells Python that the current directory should be considered as a Python package
- settings.py - Comprises the configurations of the current project like DB connections.
- urls.py - All the URLs of the project are present here
- wsgi.py - This is an entry point for your application which is used by the web servers to serve the project you have created.

## What is pycache? (Optional)

When you run a program in python, the interpreter compiles it to bytecode first (this is an oversimplification) and stores it in the `__pycache__` folder. If you look in there you will find a bunch of files sharing the names of the .py files in your project's folder, only their extensions will be either .pyc or .pyo. These are bytecode-compiled and optimized bytecode-compiled versions of your program's files, respectively.

As a programmer, you can largely just ignore it... All it does is make your program start a little faster. When your scripts change, they will be recompiled, and if you delete the files or the whole folder and run your program again, they will reappear (unless you specifically suppress that behavior).

When you're sending your code to other people, the common practice is to delete that folder, but it doesn't really matter whether you do or don't. When you're using version control (git), this folder is typically listed in the ignore file (.gitignore) and thus not included.

8. (Optional) If you created your project without "." at the end, change the name of the project main directory as src to distinguish from subfolder with the same name:

```
# Be careful to use your own folder names.
mv clarusway src
```

9. Lets create first application:

Go to the same level with manage.py file if you create your project with nested folders at step 7:

```
cd clarusway # or, if you changed;
cd src
```

Start app

```
# Using app name as "firstapp"
python manage.py startapp firstapp # or
py -m manage.py startapp firstapp

# Another naming:
python manage.py startapp home
py -m manage.py startapp home
```

10. Go to settings.py and add another line as below under INSTALLED\_APPS:

```
'firstapp',
```

11. Run your project:

```
python manage.py runserver # or  
py -m manage.py runserver
```

Go to <http://localhost:8000/> in your browser, and you should see Django rocket, which means you successfully created project.

The install worked successfully! Congratulations!

## Part 2 - Add Simple View and Say Hello World!

12. Go to `views.py` under "firstapp" directory, and create first view as "home" by adding:

```
from django.http import HttpResponse  
  
def home(request):  
    return HttpResponse("<h1>Hello world!</h1>")
```

13. Include URL path of the new app to the project url list, go to `urls.py` and add:

```
from django.urls import include  
  
urlpatterns = [  
    ...  
    path("", include("firstapp.urls")),  
]
```

14. Add `urls.py` file under firstapp directory and add:

```
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path('', views.home, name='home'),  
]
```

15. Run our project again to see our view:

```
python manage.py runserver  
py -m manage.py runserver
```

16. Go to <http://localhost:8000/> in your browser, and you should see the text "Hello, world!", which you defined in the home view.

To stop the server use "CTRL + C"

😊 **Thanks for Attending** 🙌

Clarusway

