Django: getting started

- To start a `django` project, you will create a directory (say `myproject`)
  - Create a virtual environment under it and name it as `venv`
  - Activate virtual environment (`source venv/bin/activate`) and install `django` in it (`pip install django`)
  - Finally, run `django-admin startproject myproject` to create a skeleton of project **as follows**.

```
myproject/                  <-- higher level folder
 |-- myproject/             <-- django project folder
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
 |    |    |-- wsgi.py
 |    +-- manage.py
 +-- venv/                  <-- virtual environment folder
```

Command `django-admin startproject <project-name>` is kind of a `cookiecutter` that creates project structure for us.

`Cookiecutter` is basically project template. There is famous library to create such templates - (https://cookiecutter.readthedocs.io/en/stable/).

The project structure at this stage is composed of five files:

- **manage.py**: This is a command line application. It's a shortcut to use the `django-admin` command-line utility. It's used to run management commands related to our

project. We will use it to run the development server, run tests, create migrations and much more.

- **__init__.py**: this empty file tells Python that this folder is a Python package.
- **settings.py**: this file contains all the project's configuration. We will refer to this file all the time!
- **urls.py**: this file is responsible for mapping the routes and paths in our project. For example, if you want to show something in the URL `/about/`, you have to map it here first.
- **wsgi.py**: this file is a simple gateway interface used for deployment. You don't have to bother about it. Just let it be for now. Required for deployment on application servers that follow `wsgi specifications`.
- **wsgi.py**: this file is a simple gateway interface used for deployment. You don't have to bother about it. Just let it be for now. Required for deployment on application servers. Required for deployment on application servers that follow `asgi specifications`.

Django comes with a simple web server installed. This lightweight server is used while project is in development phase. It is not meant for production environment. We can start it by executing the command:

`python manage.py runserver`

- By default `django` uses port number 8000 and you can check `http://127.0.0.1:8000`
- Use `ctrl + c` to stop development server.

NOTE

- In the world of `django` the term `project` is used for collection of configurations and apps. One project can be composed of multiple apps, or a single app.
- In the world of `django` the term `app` is composed of a set of models (database tables), views, templates, tests.
- It's important to note that you can't run a Django **app** without a **project**.

Now, let's create first `django-app`

To create our first app, go to the directory where the **manage.py** file is and execute one of the following commands:

```
# using django-admin
django-admin startapp app1

# using manage.py
python manage.py startapp app1
```

This will give us following directory structure.

```
myproject/
 |-- myproject/
 |    |-- app1/                <-- our new django app!
 |    |    |-- migrations/
 |    |    |    +-- __init__.py
 |    |    |-- __init__.py
 |    |    |-- admin.py
 |    |    |-- apps.py
 |    |    |-- models.py
 |    |    |-- tests.py
 |    |    +-- views.py
 |    |-- myproject/
 |    |    |-- __init__.py
 |    |    |-- settings.py
 |    |    |-- urls.py
 |    |    |-- wsgi.py
 |    +-- manage.py
 +-- venv/
```

So, let's first explore what each file does:

- **migrations/**: here Django store some files to keep track of the changes you create in the **models.py** file, so to keep the database and the **models.py** synchronized.
- **admin.py**: this is a configuration file for a built-in Django app called **Django Admin**.
- **apps.py**: this is a configuration file of the app itself.
- **models.py**: To write DDLs in ORM way. We define the entities of our Web application. The models are translated automatically by Django into database tables.
- **tests.py**: Used to write unit tests for the app.
- **views.py**: Contains business logic. This is the file where we handle the request/response cycle of our Web application.

Now that we created our first app, let's configure our project to *use* it.

To do that, open the **settings.py** and try to find the `INSTALLED_APPS` variable:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app1'
]
```

Hello world!

our first **view**. We will explore it in great detail in the next tutorial. But for now, let's just experiment how it looks like to create a new page with Django.

Open the **views.py** file inside the **boards** app, and add the following code:

views.py

```
from django.http import HttpResponse

def home(request):
    return HttpResponse('Hello, World!')
```

- Views are Python functions that receive an `HttpRequest` object and returns an `HttpResponse` object.
- Receive a *`request`* as a parameter and returns a *`response`* as a result.
- So, here we defined a simple view called **home** which simply returns a message saying **Hello, World!**.
- Now we have to tell Django *when* to serve this view. It's done inside the **urls.py** file:

urls.py

```
from django.conf.urls import url
from django.contrib import admin

from boards import views

urlpatterns = [
    url(r'^$', views.home, name='home'),
    url(r'^admin/', admin.site.urls),
]
```

- If you compare the snippet above with your **urls.py** file, you will notice I added the following new line: `url(r'^$', views.home, name='home')`
- and imported the **views** module from our app **boards** using `from boards import views`.

- But for now, Django works with **regex** to match the requested URL. For our **home** view, I'm using the `^$` regex, which will match an empty path, which is the homepage (this url: **http://127.0.0.1:8000**).

- If I wanted to match the URL **http://127.0.0.1:8000/homepage/**, my url would be: `url(r'^homepage/$', views.home, name='home')`.

- Let’s see what happens:

```
python manage.py runserver
```

In a Web browser, open the http://127.0.0.1:8000

That’s it!