O'REILLY: Django 3 – Full Stack Websites with Python Web Development

Django Basics

- Django can be installed using `pip install django` which will install the latest version of Django.
- Once install the command `django-admin` will show all available commands.
- To start a project use `django-admin startproject project_name`
- Once a project is started, files will be added to the folder. The file `manage.py` is located in the new project folder created.
- Running `python manage.py runserver` will start the Django server.
- The server is running on `localhost:8000`
- Tip: change the top-level folder created by Django and `-project` in its name since there is another subfolder with that name. This can be referred to as 'project folder'.

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

1) urls.py showing the new path for home page

Password Generator Project – Django

- Create project: `django-admin startproject password_generator`
- Rename the top-level folder as `password_generator-project`
- Create new app called generator: `python manage.py startapp generator`
- In the `settings.py` add the new app to the list of `INSTALLED_APPS`
- In the `urls.py` from `password_generator` folder, update the path for the home page. There is no need for admin page in this project.

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

# Create your views here.

def home(request):
    return HttpResponse("Hello Django!")
```

2) views.py with the function for home page

- In the `views.py` file of the newly created app folder create the function that will call the home page. Make sure to import the HttpResponse.
- HttpResponse can handle html code but is unpractical. Using templates will be much easier to return html code.
- Create a folder called `templates` inside the generator app folder, then another folder with the app name, in this case, `genenrator`. So it should end up with: `/templates/generator/` as folder structure.
- In this last generator folder we can create the templates for this app. Create a file called `home.html` and write a message in it.
- Going back to `views.py` file we'll replace the `HttpResponse` with `render` in order to return the html template just created.

```
def home(request):
    return render(request, 'generator/home.html')
```

- We can also return a dictionary (map) by adding it to the return as follows:

```
return render(request, 'generator/home.html', {'password':'asdf32s3fisx3'})
```

- Now when whenever we provide the tag `{{ password }}` in the template file (such as home.html) the asdf32s3fisx3 will be shown.

- Create the home.htlm page using forms to give the user choices of how the passoword to be generated. Here are the choices:
    - Length of the passowrd from 6 to 14
- Also include submit button which will create the password for the user by loading a new page called password.

```
path('password/', views.password),
```

3) the path to the localhost/password

- To create this new page we must add the new url to the urls.py file as well as creating the function in the views.py file of the app.
- Create the `password.html` file and make sure to add "password" to the form action in order to point to this file.

```
<form action="{% url 'password' %}">
```

- To make this url dynamic use the `{% url 'password' %}` tag where the 'password' is the name given in the `urls.py` file for the path that points to the password page.

```
path('password/', views.password, name='password'),
```

- Using dynamic url makes it easier when changing path since name will be used in order to point to the path. (which can be changed as many times as needed)