Django Application Design Document

# Starting up and setting the structure for the application

In order to set up a folder with the right Django file structure, the following command needs to be executed in the terminal.

django-admin startproject ‘name of application’

After this has been done, a Django folder structure is created. Lets assume that the name of the application is ‘ii\_app’, then I would enter in:

django-admin startproject ii\_website

The folder structure is now created, however, inside the structure, a developer can create multiple applications. In order to start creating an application, you need to navigate to the ii\_website folder, and then enter the following in the terminal:

python3 manage.py startapp “name of app”

So lets assume that the name of an application that I am building is named ii\_app, then the statement would look like this:

python3 manage.py startapp ii\_app

Once this is done, the folder structure for the app gets imbedded inside here.

# Views

A view allows the user to view a webpage. It does this by processing user requests. The job of a view inside the views.py file is to take a particular request, analyse it and then send back an appropriate response. A view is written inside the views.py file by writing a python function which takes in a request, and then returns a http response. Each view needs to be linked to a particular URL. A webpage can only be viewed if a URL exists for the website (view). In order to assign a URL to each view, you have to create a new file in the directory named url.py and then define the urls there, and link them to each view, so these URL patterns need to be defined.

Django has no obligation to execute the url patterns defined in the newly created urls.py file within the app structure. Rather, it goes to the urls.py file within the overall website structure and executes those urls. Therefore, you need to define the following code inside the urls.py file which resides inside the overall website structure.

urlpatterns = [

path('admin/', admin.site.urls),

path('ii\_app/', include('ii\_app.urls')),

]

The following library also needs to be imported:

from django.urls import include,path

You are telling the URL patterns to go to the path ii\_app, and then look through the ii\_app urls.

# Settings.py

The databases dictionary contains information about the database engine (DBMS) being used to store the application data. In this instance, the database engine is SQLite, however, if you want to use a different database, you can install it and then name it in the databases dictionary. Secondly, the installed apps within the website are listed. Notice how the newly created app (i.e. ii\_app) is not listen in the settings.py file. Therefore, it needs to be added manually. It is added as the following format:

‘name\_of\_app.apps.name\_of`\_class\_in\_apps.py

# Models

Models allow us to create database tables. So models are the blueprints which can be used to create database tables.

These blueprints or models are created using Python classes and these models are created in models.py.

Every time an update has been made to the models, the following command updates the tables in the SQLite database:

Python3 manage.py makemigrations ii\_app

This will create the models in the databases that have been defined in the models.py file. You will get something like this:

Migrations for 'ii\_app':

ii\_app/migrations/0001\_initial.py

- Create model Employee

0001 is the name of the model that has been created. However, this model needs to be migrated to the database now. This can be done with the following two command:

Python3 manage.py sqlmigrate ii\_app 0001

Python3 manage.py migrate

After doing this, the physical table has been created in the SQLite database.

Let’s say that we create a class named employees. We have to create an object from the employees’ class and then save the object using the save () method. The data then gets stored into the table.

If you want to add data into the database, you have to do it via python shell. First stop the server and then type in the following to open the shell:

python3 manage.py shell

We want to import the employees table into the python shell by writing:

from ii\_app.models import Employee

in order to see all of the data inside the table, you type the following into the shell:

Employee.objects.all()

This will give an empty queryset if there is no data in the table. In the shell, if we want to add data to the table, we have to create a new object, defining the variables within the class. This can be done as follows:

romi = Employee(employee\_name = 'Romi Dhillon', employee\_position = 'Data Architect', employee\_cost = 753)

A new object named romi has been created, but this object has only been created and not actioned to create anything in the database table. In order to store the object in the database, the following statement needs to be executed in the database:

romi.save()

You can get the unique id of this object by tying in the following into the shell

romi.id() or

romi.pk