**Django**

Are you looking for a web framework that is open-source and uses model-view-template(MVT) architecture to create a database driven website?

Introducing Django , built using python with a principle known as Don’t Repeat yourself to suit your need. It provides an abstraction with reusable and pluggable components.

Django consists of

- Object-Relational Mapper(ORM) : mediates between data models.

- Relational Database Model : a system for processing web requests.

- View : A template system dealing with functional logic

- Controller : a regular-expression based URL dispatcher.

**Let’s Focus on the ORM and model in detail.**

**Why Django :**

- Ridiculously Fast

- Fully loaded

- Reassuringly Secure

- Exceedingly scalable

- Incredibly Versatile

Django is powered with inclusive battery – the Django ORM

**Django Installation**

Since Django is python based module, install using the pip installer

$ pip install django

To verify the installation, get the installed version using

$ python3 -c “import django; print(django.get\_version())”

1.11.7

**Database support**

Django comes with a default SQLite. However, packages are available for other database soures.

- PostgreSQL

- MySQL

- Oracle

**Your First Django Project:**

**Project Creation:** The Django environment is ready, and you can build your first web application now. It starts by creating a new Django project.

From command line, cd to a directory where the project code will be created.

Use the django-admin command to create a project myweb.

$ django-admin startproject myweb

**What Entails a Django project:**

Django creates a project structure like:

Container for the project

myweb/

A command-line utility to interact with the project myweb

manage.py

Python package for the project. Import it to use any code inside

myweb/

myweb directory is considered a Python package due to this empty file

\_\_init\_\_.py

Setting/Configuration for myweb project

settings.py

URL declarations similar to Table of contents for myweb project

urls.py

An entry-point for WSGI-compatible web servers to serve myweb project.

wsgi.py

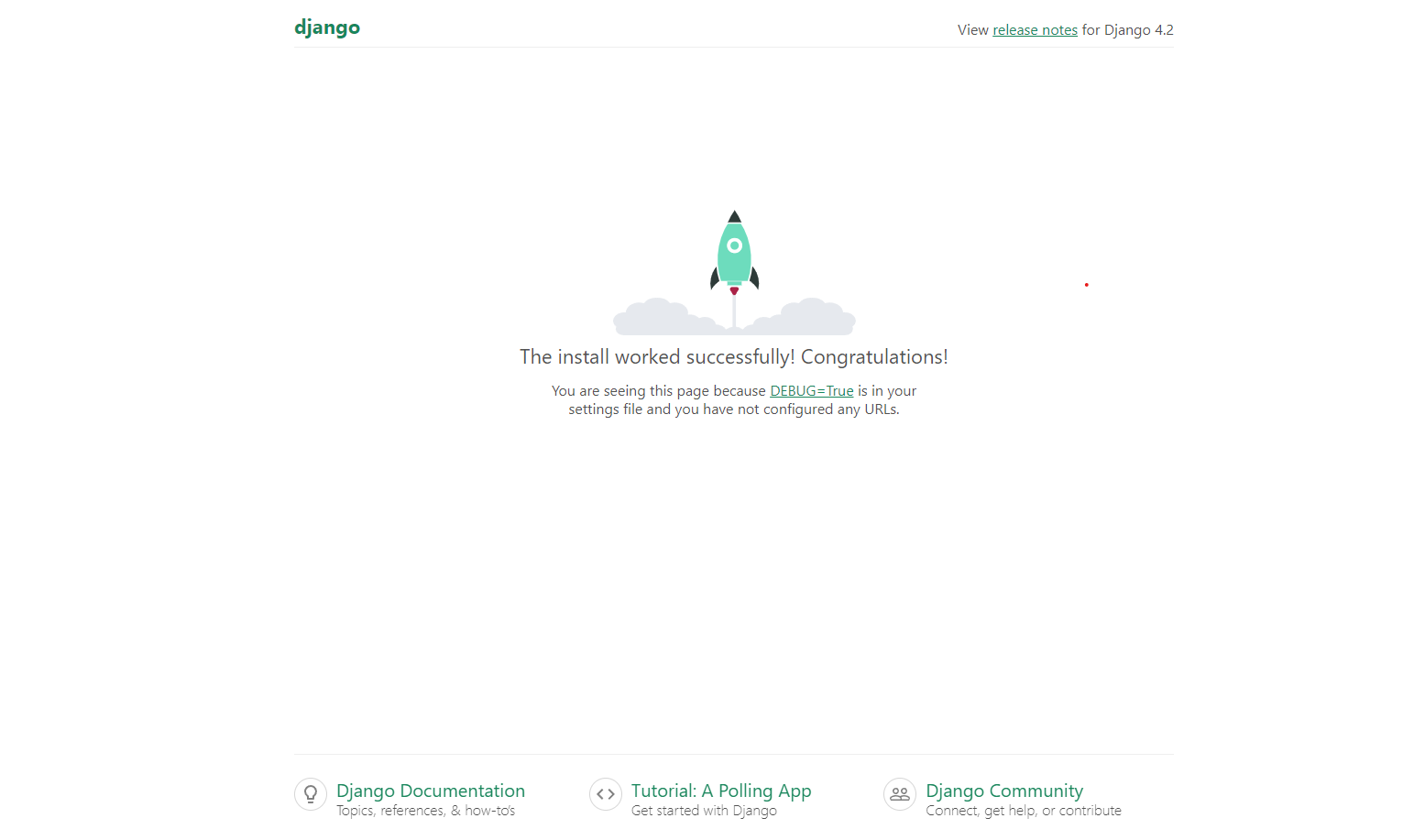
**Up and Running your Project:**

Lets start the project to see the Django web interface. On the command line, change to myweb root directory and execute the command

$ python manage.py runserver

You’ll see a output on the command line.

Open the web browser and provide the URL <http://127.0.0.1:8000/> to launch the Django project page.



**Create an App:**

To work with the project, you need a application or app that performs different actions.

To create an app poll. Go to the container directory myweb and execute the command.

$ python manage.py startapp poll

The poll app is created with all necessary components.

**What an App Entails:**

A Django app has the following:

admin.py reads model metadata and provides an interface to manage app content

views.py web based requests and response is configured in this file

apps.py application configuration details for the app are included. Eg: custom app name.

tests.py app unit test automation classes are included in this

models.py A class for each model is defined with the model structure layout

migrations/ contains migrated model details with the corresponding database table structure.

**Object – Relational Mapper**

Relational Database

|  |  |  |
| --- | --- | --- |
| Name | Profile photo | Employee Details |
|  |  |  |
|  |  |  |

We now have a fair understanding of the structure of Django project,

From this topic onwards we will focus on the key aspects of Django – ORM, model and migrations.

Object – represents the code objects of the programming language

Relational – The RDBMS database where the data is stored

Mapper – Connection between the object and the relational database

ORM – code library to automate data transfer from relational databases to code objects.

The object values are converted to groups of simpler values to store in the database.

**ORM Characteristics**

Create table automatically in the database using code, similar to hibernate in java, python has Django.

**Python ORM Implementations**

Python has many ORM implementations

Django ORM – Default built-in ORM for Django Web Framework

SQLAlchemy – Abstraction level is just right, which makes complex queries easier to implement

Peewee – A simple, small ORM with few concepts that are expressed in detail

Pony ORM – Provides a convenient syntax for writing queries with automatically query optimization

SQLObject – An Object-based query language that makes SQL more abstract and provides high

Database independence.

**Django’s ORM**

The Django web framework has its own built-in object-relational mapping module, referred as Django ORM.

<https://www.youtube.com/watch?v=eio1wDUHFJE> (Django Tutorial #8 – The Django ORM).

**A Sample ORM**

Lets consider a simple SQL query on a customer table to retrieve a customer name.

SELECT \* FROM CUSTOMER WHERE name= ‘fresco play’;

The equivalent Django ORM query is a Python code.

>>> customer = Customers.objects.filter(name= ‘fresco play’)

A python model class (Customers) is represented as a database table

The class instance(customer) is represented as a particular record on the table.

Lets dive right into models for deeper understanding.

**What’s a Django model:**

A model is a representation of all the information about your data.

Every model is equivalent to a database table and represent the behaviour of the stored data.

A Django model:

Is a Python class that subclasses django.db.models.Model

Has an attribute/field that represents a database column.

Has class and field objects that can be accessed through an automatically generated database-access API

**Understanding a Model with an Example**

Now, define a model Customer with firstname and lastname as CharField attributes limiting to 30 character length.

from django.db import models

class Customer(models.Model):

firstname = model.CharField(max\_length=30)

lastname = model.CharField(max\_length=30)

The Customer model is equivalent to a database table represented as

CREATE TABLE frescoapp\_customer (

“id” serial NOT NULL PRIMARY KEY,

“firstname” varchar(30) NOT NULL,

“lastname” varchar(30) NOT NULL

)

id is the auto-assigned primary key by Django with the type as AutoField.

frescoapp\_customer is the table name where frescoapp is the Django app name, and customer represents the Customer model name.

**Model Manager – Not the Usual Boss:**

A manager is an interface through which Django models are accessed.

Each model has at least one Manager.

Default manager objects is assigned to each model as we have seen before with the Customer model.

>>> customer = Customer.objects.filter(name=’fresco play’)

To rename a given class’ Manager, define a class attribute of type models.Manager() on that model.

from django.db import models

class Customer(models.Model):

#....

custname = models.Manager()

This creates a manager custname, hence using Customer.objects will generate an AttributeError exception.