**Object Relational Mapper**

Django is a web framework that is open-sourced and uses **model-view-template (MVT)** architecture to create a database driven website.

Django is 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.

**Features:**

* **Ridiculously fast** - Build your application from **concept to completion** as quick as possible.
* **Fully loaded** - User authentication, content administration, sitemaps, and RSS feeds — all packages included.
* **Reassuringly secure** - Other than providing a secure user account management, Django helps to tackle clickjacking, cross-site scripting, cross-site request forgery (CSRF), and SQL injection.
* **Exceedingly scalable** - Heavy traffic demands from websites are scaled quick and flexible.
* **Incredibly versatile** - Builds a content management portal, social network or scientific computing platform, works across multiple industries.

**Database Support**

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

* PostgreSQL database - psycopg2 package.
* MySQL database - mysqlclient DB API driver.
* Oracle database - cxOracle DB API driver.

**Database settings**

Every **Django project** will have a **settings** configuration in a file settings.py.

The file has a DATABASES section which is configurable.

Django adds **sqlite database** settings by default

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

**First Django Project:**

**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

**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 the 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 DjangoLearning . # period here means to create in current folder

Is that it? Yes! Now, let's look at what Django has created.

**What Entails a Django Project?**

Django creates a project structure like:

myweb/

manage.py

myweb/

\_\_init\_\_.py

settings.py

urls.py

wsgi.py

* myweb/: container for the project.
* myweb/manage.py: A command-line utility to interact with the project myweb
* myweb/myweb/: Python package for the project. Import it to use any code inside (e.g. myweb.urls).
* myweb/myweb/**init**.py: myweb directory is considered a Python package due to this empty file.
* myweb/myweb/settings.py: Settings/configuration for myweb project.
* myweb/myweb/urls.py: URL declarations similar to **Table of contents** for myweb project.
* myweb/myweb/wsgi.py: An entry-point for WSGI-compatible web servers to serve myweb project.

We will deal with urls and wsgi in next Django course(s).

**Up and Running Your Project**

Let's 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 the subsequent output on the command line:

Performing system checks...

System check identified no issues (0 silenced).

You have 13 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

December 04, 2017 - 11:39:43

Django version 1.11.7, using settings 'myweb.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

Open a 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.

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

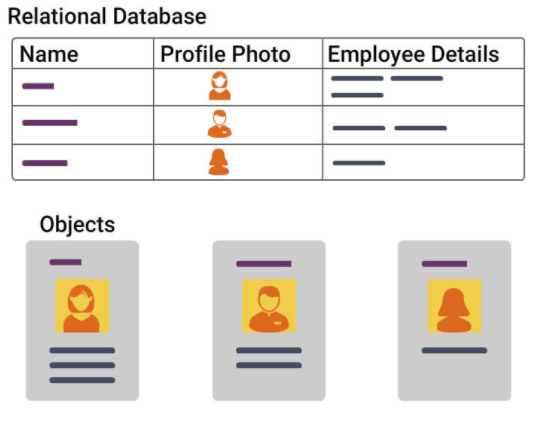
What an App Entails?

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. e.g., 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 –**

If a class name customer is created with 3 properties Name, profile Photo, Employee Details. Using ORM, without much concentrating on building database queries on how to create table and columns. ORM helps in automatic creation based on class definition. This concept is called ORM



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.
* Object-relational Mapper (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.

**Python ORM Implementations**

Python has many ORM implementations

* **Django ORM** - Default built-in ORM for the 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 automatic query optimization.
* **SQLObject** - An Object-based query language that makes SQL more abstract and provides high database independence.

**Vedio links:**

<https://www.youtube.com/watch?v=7E1M1W9o7PA&ab_channel=Telusko>

<https://www.youtube.com/watch?v=eio1wDUHFJE&list=PL4cUxeGkcC9ib4HsrXEYpQnTOTZE1x0uc&index=8&ab_channel=TheNetNinja>

**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 represents 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 = models.CharField(max\_length=30)

lastname = models.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 Customers model

>>> customer = Customer.objects.filter(name='freso 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.

**How to Access a Model?**

Consider the example

from django.db import models

class Customers(models.Model):

name = models.CharField(max\_length=100)

customersince = models.DateField()

phone = models.IntegerField()

To return all customers in the database, use,

>>> Customers.objects.all()

If you create a customized manager using custname = models.manager(), then, to get all customers,

>>> Customers.custname.all()

**Useful URL’s:**

Official Django documentation URL: <https://docs.djangoproject.com/en/3.1/>