

PYTHON PROGRAMMING

UNIT - 5:

Django: Installing Django – Building an Application – Project Creation –Designing the Data Schema - Creating an administration site for models -Working with QuerySets and Managers – Retrieving Objects – Building List and Detail Views.

Installing Django :

Python Django

Django is used in many popular sites like Disqus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, etc. There are more than 5k online sites based on the Django framework. (Source) Sites like Hot Frameworks assess the popularity of a framework by counting the number of GitHub projects and StackOverflow questions for each platform, here Django is in 6th position. Web frameworks often refer to themselves as “opinionated” or “un-opinionated” based on opinions about the right way to handle any particular task. Django is somewhat opinionated, hence delivering the in both worlds(opinionated & un-opinionated).

Django gives you ready-made components to use such as:

1. It's very easy to switch databases in the Django framework.
2. It has a built-in admin interface which makes it easy to work with it.
3. Django is a fully functional framework that requires nothing else.
4. It has thousands of additional packages available.
5. It is very scalable.

Features of Django

- **The versatility of Django:** Django can build almost any type of website. It can also work with any client-side framework and can deliver content in any format such as HTML, JSON, XML, etc. Some sites which can be built using Django are wikis, social networks, new sites etc.
- **Security:** Since the Django framework is made for making web development easy, it has been engineered in such a way that it automatically do the right things to protect the

website. For example, In the Django framework instead of putting a password in cookies, the hashed password is stored in it so that it can't be fetched easily by hackers.

- **Scalability:** Django web nodes have no stored state, they scale horizontally – just fire up more of them when you need them. Being able to do this is the essence of good scalability. Instagram and Disqus are two Django based products that have millions of active users, this is taken as an example of the scalability of Django.
- **Portability:** All the codes of the Django framework are written in Python, which runs on many platforms. Which leads to run Django too in many platforms such as Linux, Windows and Mac OS.

Django Installation

To install Django, first visit to [django official site \(https://www.djangoproject.com\)](https://www.djangoproject.com) and download django by clicking on the download section. Here, we will see various options to download The Django.

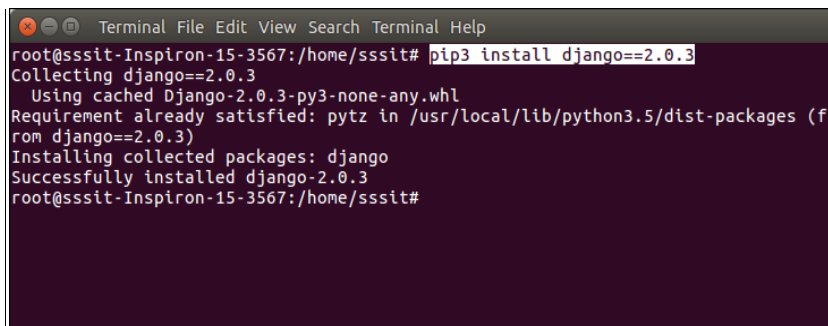
Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions **pip3** is used to manage packages.

In this tutorial, we are installing Django in Ubuntu operating system.

The complete installation process is described below. Before installing make sure **pip is installed** in local system.

Here, we are installing Django using pip3, the installation command is given below.

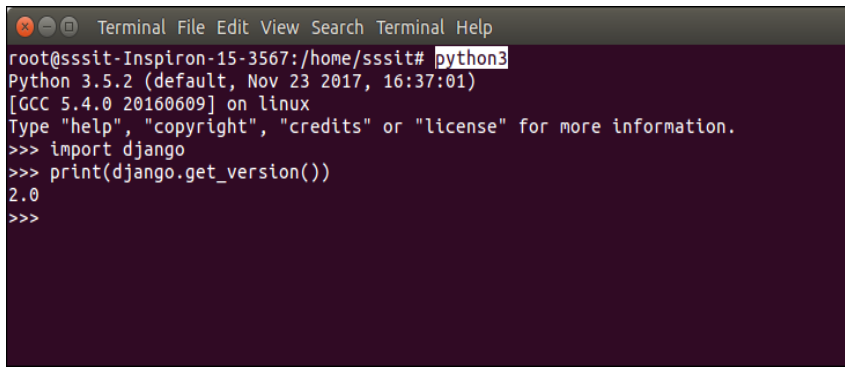
1. \$ pip3 install django==2.0.3



```
root@sssit-Inspiron-15-3567:/home/sssit# pip3 install django==2.0.3
Collecting django==2.0.3
  Using cached Django-2.0.3-py3-none-any.whl
Requirement already satisfied: pytz in /usr/local/lib/python3.5/dist-packages (from django==2.0.3)
Installing collected packages: django
Successfully installed django-2.0.3
root@sssit-Inspiron-15-3567:/home/sssit#
```

Verify Django Installation

After installing Django, we need to verify the installation. Open terminal and write **python3** and press enter. It will display python shell where we can verify django installation.

A terminal window with a dark background and light text. The window title is "Terminal File Edit View Search Terminal Help". The prompt is "root@sssit-Inspiron-15-3567:/home/sssit#". The user has entered "python3", which has started a Python 3.5.2 shell. The prompt is now ">>>". The user has entered "import django", and the prompt is ">>>". The user has entered "print(django.get_version())", and the output is "2.0". The prompt is ">>>".

```
root@sssit-Inspiron-15-3567:/home/sssit# python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> print(django.get_version())
2.0
>>>
```

Look at the Django version displayed by the print method of the python. Well, Django is installed successfully. Now, we can build Django web applications.

Building an application

How to Make an App in Python

These days, there's a mobile app for everything. From health to education, there is no one field an app isn't in. There comes a thought in mind, and once checked, there is already an app for it. The thing is that there are a lot of apps to do a task. There are some characteristics of an app that makes it unique from the other:

1. The background internet and battery consumption
2. Speed
3. The outlook
4. Bugs, errors, and glitches
5. User data

App development is a beautiful domain to work in because all the work the developer puts in always is visible. A good app always makes its way into the competition sooner or later. But building a good app isn't that easy. It starts with a good idea and ends with a good UI, but everything in the middle does take a lot of the app's performance.

This tutorial is designed to help young students and developers make a checklist when building an app or wanting to.

1. Idea

A good idea is a wonderful asset anyone can ever possess. For a person wanting to start something new, there is always something that drives him. It is the business idea for a start-up founder; it is the app idea for an app developer.

- It can be a fresh idea that no one has ever touched or
- It can even be just a simple thought in your brain when using an app.

- A simple feature can turn the whole app around.
- If you already have a good thought in your mind about what you want to build, then you are ready already.
- If you are not sure whether the idea is going to click that well, it's okay too:
 1. Dig up on the topic
 2. Do all the research; you can check out all the related apps.
 3. There will be a point you will reach a position where you'll know what you intend to create.
- If you don't have any idea but want to develop an app, think about all the disadvantages you faced while using other apps. Then, find some already existing apps and get into using them. Try doing something new, like:
 1. Create remixes of the apps. That'll give the best combinations creating a good byproduct.
 2. Not having an idea is also pretty good because you can keep building features you want, and once you figure out what you want to create, you'll have all the features you developed right in your hand.

2. Market Research

Now, you have an idea of what you want to create. Now, it's time to check the competition your idea has. If your idea is already implemented, check all those apps and note the best and worst parts of using those apps. You have to ensure that all the best parts are covered in your app, and all the worst parts are resolved.

- If you have an idea no one has ever tried before, check for related topics and create statistics on what feature had the most growth.
- If you are trying to improve an existing app trend, find all the features that add value to your app and give the best UI to attract the user.

Developing an app is as cool as it sounds; it attracts developers and users to try out new types of apps. There are billions of apps already. But, in the billion apps, not all are legit, and not all are up to expectations.

If you're serious about your app and excited about it, you have to give the best to get the best growth.

3. List out everything

Tons of research and brainstorming will make you reach a point where you feel like you know what exactly you are going to do. Don't just start doing it; take a moment and write all the features you don't want to miss in your app because your brain will not stay in the same active state. Create a checklist and check it off once you have included a feature. This way, you can track your progress, giving you confidence amid all the errors and bugs you'll face in the middle of development.

4. Design the App

Now, you have an idea, an approximate measure of how well the app might perform in the market, and everything is listed. You close your eyes, and you should be able to see your app in your head. You have to ensure that is how your app should turn out to be, and for that, you have to create a basic design of all the features and the arrangements you have to set up in the app. It is like a house plan before you build your house.

5. The UI and the GUI

Find your target audience. There's always a target audience for an app. If it is an education-based app, it's always students and sometimes parents. Some people use apps just because they find them aesthetic, and some won't use an app unless they find it safe. Keep in mind all of these different perspectives and build the UI of the app. There are a lot of online options giving many frameworks. Check them and find the best one.

1. Build it the way you'll like it as a user.
2. But, don't limit it to yourself; think like your target audience and think about how they want the app to be like.

6. Build the app

It is the most important and the core part of developing your app:

1. You can use Java and other languages to code it.
2. You can use an existing API and add more features.
3. You can use an online tool-make sure you find the right one.
4. Find your language and start the process.

It is like building your house. You took inspiration from existing models. You have the plan, know what you want to include, and design it all on paper. It is the right time to get into the actual process.

Never rush on the process. Be patient and check everything twice. If you are coding, think like a hacker and a user and be a developer who can satisfy the user by blocking out the hacker.

Think about all the possible scenarios and problems the app might encounter and create a solution for them. When the app is in the market, you might get a lot of reviews, but if the user finds all the errors in the starting stage itself, a bad review in the initial stage will not do any good for the growth of the app.

7. Ads

It is the money stage. It is your choice, though. Advertising is good and sometimes bad if you do it too much. But in the end, everything depends on the keystrokes and how the app turns out.

1. If you are confident about what you built, try out a few ads and make sure ads don't interrupt the user.
2. As mentioned above, always think like you are a user. Would you like it if the app you are using had too many ads that interrupt your usage of the app?

8. Marketing strategies

There are a lot of bakeries in the street, and you just opened a brand new one. How does the customer know the taste of your cake if he doesn't even taste it, and why will he even taste it if he doesn't even know your bakery exists? You need to market your app. simply put, you need to let people know that you created something and want them to try it out. You can try:

1. Email marketing
2. SMS marketing
3. Paid marketing
4. Build an attached website and create ads
5. Use social media
6. Journal about your journey and try blogging

9. Make your way into the play store/ App store.

10. Check the reviews and act on them and update the app

In the initial stage of the app launch, there will not be many people trying it out. It all starts with 10's and then 100's users, and eventually, it grows. But, as a developer, you have to put the users in a position they feel comfortable using and then review it.

1. Respond to the reviews.
2. Thank the good reviews
3. Evaluate the bad reviews.

The world is evolving, and so is technology. We use an app, and a better one comes along; we replace the old one. It is always what happens. If people like your app, they'll choose your app over all the old apps they've been using. Now, what if another app with better features comes along?

Don't let your app grow old. Keep updating and adding new features so that people never get bored of it, and even if they see a new feature in some other app, it should already be in the app.

It makes your app stay and make room for itself in the domain.

Django Project

In the previous topic, we have installed Django successfully. Now, we will learn step by step process to create a Django application.

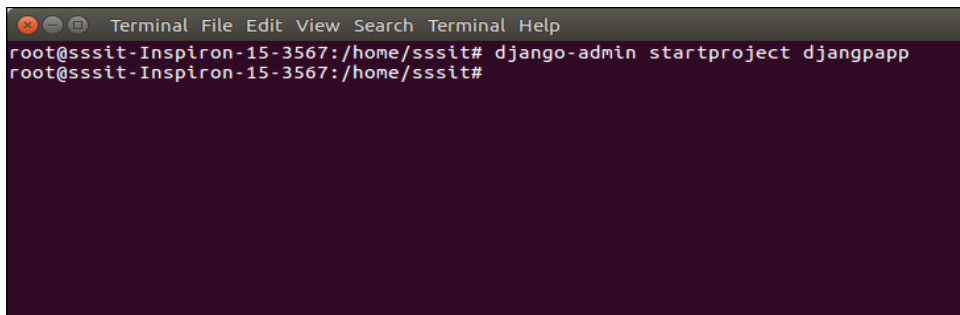
To create a Django project, we can use the following command. *projectname* is the name of Django application.

1. `$ django-admin startproject projectname`

Django Project Example

Here, we are creating a project **djangpapp** in the current directory.

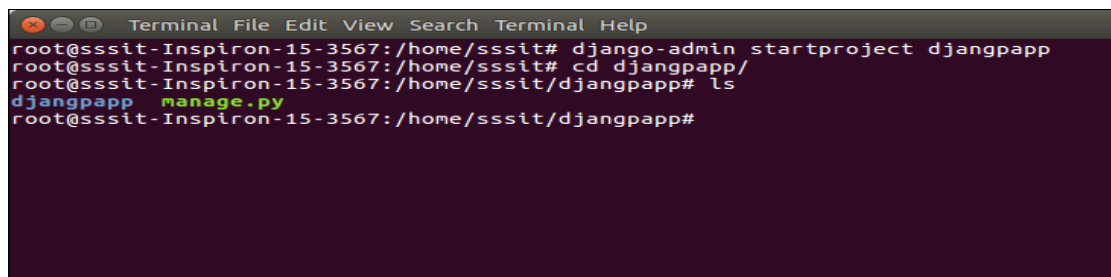
1. `$ django-admin start project djangpapp`

A terminal window with a dark purple background and a menu bar at the top containing 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command 'django-admin startproject djangpapp' being executed. The prompt is 'root@sssit-Inspiron-15-3567:/home/sssit#'. The output is 'root@sssit-Inspiron-15-3567:/home/sssit#'.

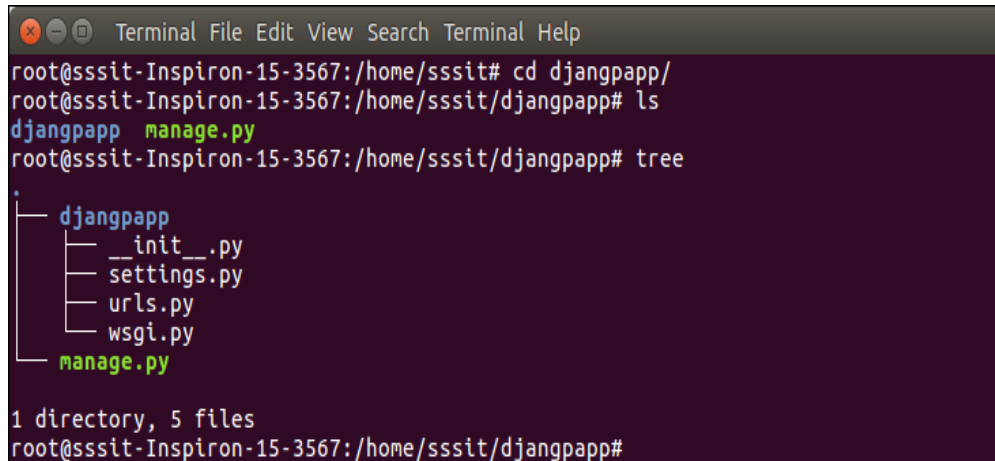
Locate into the Project

Now, move to the project by changing the directory. The Directory can be changed by using the following command.

`cd djangpapp`

A terminal window with a dark purple background and a menu bar at the top containing 'Terminal', 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command 'cd djangpapp' being executed. The prompt is 'root@sssit-Inspiron-15-3567:/home/sssit#'. The output is 'root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#'. The terminal also shows the command 'ls' being executed, and the output is 'djangpapp manage.py'.

To see all the files and subfolders of django project, we can use **tree** command to view the tree structure of the application. This is a utility command, if it is not present, can be downloaded via **apt-get install tree** command.

A terminal window with a dark background and light-colored text. The window title is "Terminal File Edit View Search Terminal Help". The prompt is "root@sssit-Inspiron-15-3567:/home/sssit#". The user enters "cd.djangpapp/" and the prompt changes to "root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#". The user enters "ls" and the output is ".djangpapp manage.py". The user enters "tree" and the output is a tree structure: ".djangpapp" with sub-items "__init__.py", "settings.py", "urls.py", "wsgi.py", and "manage.py". Below the tree structure, it says "1 directory, 5 files". The prompt is "root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#".

```
root@sssit-Inspiron-15-3567:/home/sssit# cd.djangpapp/
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp# ls
.djangpapp manage.py
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp# tree
.
├── .djangpapp
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   └── manage.py
└── manage.py

1 directory, 5 files
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp#
```

A Django project contains the following packages and files. The outer directory is just a container for the application. We can rename it further.

- **manage.py:** It is a command-line utility which allows us to interact with the project in various ways and also used to manage an application that we will see later on in this tutorial.
- A directory (djangpapp) located inside, is the actual application package name. Its name is the Python package name which we'll need to use to import module inside the application.
- **__init__.py:** It is an empty file that tells to the Python that this directory should be considered as a Python package.
- **settings.py:** This file is used to configure application settings such as database connection, static files linking etc.
- **urls.py:** This file contains the listed URLs of the application. In this file, we can mention the URLs and corresponding actions to perform the task and display the view.
- **wsgi.py:** It is an entry-point for WSGI-compatible web servers to serve Django project.

Initially, this project is a default draft which contains all the required files and folders.

Running the Django Project

Django project has a built-in development server which is used to run application instantly without any external web server. It means we don't need of Apache or another web server to run the application in development mode.

To run the application, we can use the following command.

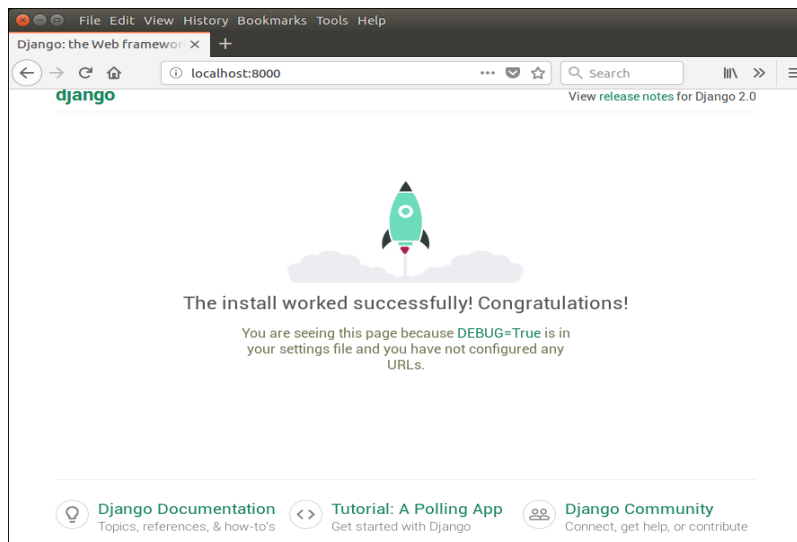
1. `$ python3 manage.py runserver`

```
Terminal File Edit View Search Terminal Help
root@sssit-Inspiron-15-3567:/home/sssit/djangpapp# python3 manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 14 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.

March 13, 2018 - 07:21:03
Django version 2.0, using settings 'djangpapp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```



DESIGNING THE DATA SCHEMA

You will start designing your blog data schema by defining the data models for your blog. A model is a Python class that subclasses `django.db.models.Model` in which each attribute represents a database field. Django will create a table for each model defined in the `models.py` file. When you create a model, Django will provide you with a practical API to query objects in the database easily.

First, you need to define a Post model. Add the following lines to the `models.py` file of the blog application:

```
from django.db import models
from django.utils import timezone
```

```

from django.contrib.auth.models import User
class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250,
                           unique_for_date='publish')
    author = models.ForeignKey(User,
                              on_delete=models.CASCADE,
                              related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10,
                             choices=STATUS_CHOICES,
                             default='draft')

    class Meta:
        ordering = ('-publish',)
    def __str__(self):
        return self.title

```

Activating the application

In order for Django to keep track of your application and be able to create database tables for its models, you have to activate it. To do this, edit the settings.py file and add `blog.apps.BlogConfig` to the `INSTALLED_APPS` setting. It should look like this:

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog.apps.BlogConfig',
]

```

The `BlogConfig` class is your application configuration. Now Django knows that your application is active for this project and will be able to load its models.

Creating and applying migrations

Now that you have a data model for your blog posts, you will need a database table for it. Django comes with a migration system that tracks the changes made to models and enables them to propagate into the database. As mentioned, the migrate command applies migrations for all applications listed in `INSTALLED_APPS`; it synchronizes the database with the current models and existing migrations.

First, you will need to create an initial migration for your Post model. In the root directory of your project, run the following command:

```
python manage.py makemigrations blog
```

You should get the following output:

```
Migrations for 'blog':
  blog/migrations/0001_initial.py
    - Create model Post
```

Django just created the `0001_initial.py` file inside the migrations directory of the blog application. You can open that file to see how a migration appears. A migration specifies dependencies on other migrations and operations to perform in the database to synchronize it with model changes.

Let's take a look at the SQL code that Django will execute in the database to create the table for your model. The `sqlmigrate` command takes the migration names and returns their SQL without executing it. Run the following command to inspect the SQL output of your first migration:

```
python manage.py sqlmigrate blog 0001
```

The output should look as follows:

```
BEGIN;
--
-- Create model Post
--
CREATE TABLE "blog_post" ("id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "title" varchar(250) NOT NULL, "slug" varchar(250) NOT NULL,
"body" text NOT NULL, "publish" datetime NOT NULL, "created" datetime NOT NULL,
"updated" datetime NOT NULL, "status" varchar(10) NOT NULL, "author_id" integer NOT
NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "blog_post_slug_b95473f2" ON "blog_post" ("slug");
CREATE INDEX "blog_post_author_id_dd7a8485" ON "blog_post" ("author_id");
COMMIT;
```

The exact output depends on the database you are using. The preceding output is generated for SQLite. As you can see in the output, Django generates the table names by combining the application name and the lowercase name of the model (blog_post), but you can also specify a custom database name for your model in the Meta class of the model using the db_table attribute.

Django creates a primary key automatically for each model, but you can also override this by specifying primary_key=True in one of your model fields. The default primary key is an id column, which consists of an integer that is incremented automatically. This column corresponds to the id field that is automatically added to your models.

How to Add Model to Django Admin Site [Step-by-step]

Django has a special admin dashboard feature. In Django, models are defined to store the user data.

As part of the [Django tutorial](#), we are going to see how we can add model to Django admin so that we can see all the model data entries from users.

Step 1: Create model in Django

If you have already created model, you can skip this step.

As an example, let's take a model called ContactModel which has three fields-

- name to store the name of the user
- mobile to store the mobile number of the user
- and email to store the email ID of the user to contact
- You can define the model in the models.py as below.

```
1 class ContactModel(models.Model):  
2     name = models.CharField(max_length=120)  
3     mobile = models.IntegerField()  
4     email = models.EmailField()
```

- In an earlier tutorial, we have learned about [creating a form in Django](#). Similarly you can create the contact form to allow users to enter the data that will be saved in this model.

- Being admin, we might want to see who all have contacted us by submitting the custom contact form.
- Django has admin dashboard feature by which we can see or track all the data models or information.
- Once user share the contact detail by filling the contact form, let's see how we can check the information.

Step 2: Add model to Django admin

- You can check the admin.py file in your project app directory. If it is not there, just create one.
- Edit admin.py and add below lines of code to register model for admin dashboard.
- `from django.contrib import admin`
- `from .models import ContactModel`
-
- `@admin.register(ContactModel)`
- `class RequestDemoAdmin(admin.ModelAdmin):`
- `list_display = [field.name for field in`
- `ContactModel._meta.get_fields()]`
- Here, we are showing all the model fields in the admin site. You can also customize the list_display list by specifying selected fields.
- For example, we want to display only a name and an email ID (and not a mobile number) in the Django admin site.

```
1 from django.contrib import admin
```

```
2 from .models import ContactModel
```

```
3
```

```
4 @admin.register(ContactModel)
```

```
5 class RequestDemoAdmin(admin.ModelAdmin):
```

```
6 list_display = ['name', 'email']
```

- That's all. Now run your Django project.

Step 3: Execute and run project

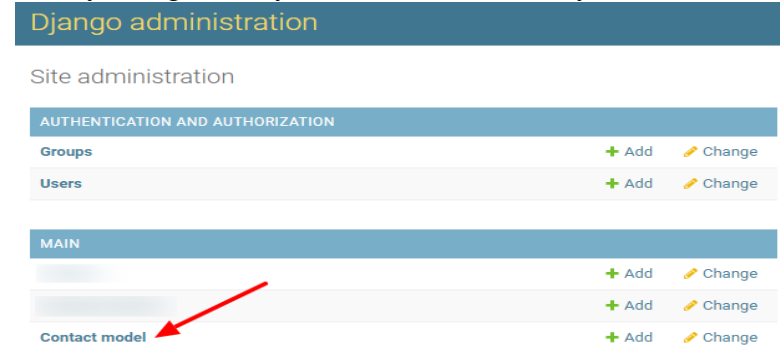
- If you have created the new model, you have to migrate the models into the database. Execute the below command in the console using manage.py script.
- **Make Migrations:**
- `python manage.py makemigrations`
- **Migrate:**

- python manage.py migrate
- Now let's start your Django project by running Django server.

1python manage.py runserver

- Once it is started, open the Django admin dashboard in the browser.
- <http://127.0.0.1:8000/admin>

Once you login into your admin dashboard, you can see the contact model in your dashboard.



Click on the “Contact model”. You will see all the model data entries.

Select contact model to change

Action: Go 0 of 16 selected

<input type="checkbox"/>	ID	NAME	MOBILE	EMAIL
<input type="checkbox"/>	16	Ani	5656	aniruddha@gmail.com
<input type="checkbox"/>	15	Ani	354545	aniruddha@gmail.com
<input type="checkbox"/>	14	Ani	354545	aniruddha@gmail.com
<input type="checkbox"/>	13	fgfg	4546656	aniruddha@gmail.com
<input type="checkbox"/>	12	fgfg	4546656	aniruddha@gmail.com
<input type="checkbox"/>	11	Ani	4656565656	aniruddha@gmail.com

Working with Query Sets and Managers

Django QuerySet

A QuerySet is a collection of data from a database.

A QuerySet is built up as a list of objects.

QuerySets makes it easier to get the data you actually need, by allowing you to filter and order the data at an early stage.

In this tutorial we will be querying data from the Member table.

Member:

id	firstname	lastname	phone	joined_date
1	Emil	Refsnes	5551234	2022-01-05
2	Tobias	Refsnes	5557777	2022-04-01
3	Linus	Refsnes	5554321	2021-12-24
4	Lene	Refsnes	5551234	2021-05-01
5	Stalikken	Refsnes	5559876	2022-09-29

Querying Data

In views.py, we have a view for testing called testing where we will test different queries.

In the example below we use the .all() method to get all the records and fields of the Member model:

View

Get your own Django Server

views.py:

```
from django.http import HttpResponse
from django.template import loader
from .models import Member

def testing(request):
    mydata = Member.objects.all()
    template = loader.get_template('template.html')
    context = {
        'mymembers': mydata,
    }
    return HttpResponse(template.render(context, request))
```

The object is placed in a variable called mydata, and is sent to the template via the context object as mymembers, and looks like this:

```
<QuerySet [
  <Member: Member object (1)>,
```

```
<Member: Member object (2)>,
<Member: Member object (3)>,
<Member: Member object (4)>,
<Member: Member object (5)>
]>
```

Template

templates/template.html:

```
<table border='1'>
  <tr>
    <th>ID</th>
    <th>Firstname</th>
    <th>Lastname</th>
  </tr>
  {% for x in mymembers %}
    <tr>
      <td>{{ x.id }}</td>
      <td>{{ x.firstname }}</td>
      <td>{{ x.lastname }}</td>
    </tr>
  {% endfor %}
</table>
```

Output:

ID	Firstname	Lastname
1	Emil	Refsnes
2	Tobias	Refsnes
3	Linus	Refsnes
4	Lene	Refsnes
5	Stalikken	Refsnes

In views.py you can see how to import and fetch members from the database.

Managers

***class* Manager**

A Manager is the interface through which database query operations are provided to Django models. At least one Manager exists for every model in a Django application.

The way Manager classes work is documented in [Making queries](#); this document specifically touches on model options that customize Manager behavior.

Manager names

By default, Django adds a Manager with the name `objects` to every Django model class. However, if you want to use `objects` as a field name, or if you want to use a name other than `objects` for the Manager, you can rename it on a per-model basis. To rename the Manager for a given class, define a class attribute of type `models.Manager()` on that model.

```
from django.db import models
```

```
class Person(models.Model):  
    # ...  
    people = models.Manager()
```

Custom managers

You can use a custom Manager in a particular model by extending the base Manager class and instantiating your custom Manager in your model.

There are two reasons you might want to customize a Manager: to add extra Manager methods, and/or to modify the initial QuerySet the Manager returns.

Adding extra manager methods

Adding extra Manager methods is the preferred way to add “table-level” functionality to your models. (For “row-level” functionality – i.e., functions that act on a single instance of a model object – use [Model methods](#), not custom Manager methods.)

For example, this custom Manager adds a method `with_counts()`:

```
from django.db import models
```

```

from django.db.models.functions import Coalesce

class PollManager(models.Manager):
    def with_counts(self):
        return self.annotate(num_responses=Coalesce(models.Count("response"), 0))

class OpinionPoll(models.Model):
    question = models.CharField(max_length=200)
    objects = PollManager()

class Response(models.Model):
    poll = models.ForeignKey(OpinionPoll, on_delete=models.CASCADE)
    # ...

```

Retrieving all objects in Django

The Django framework makes it easy to retrieve objects from the database. In this article, we'll look at how to retrieve all objects from a database in Django. We'll also look at how to filter objects based on certain criteria.

```
all_entries = Entry.objects.all();
```

If you want to retrieve all objects in Django, you can use the above query set. For example, if there is a Model named as 'Person' and it has fields like first_name and last_name and you want to get all entries made in this table then you can use below query set.

```
Person.objects.all()
```

Building List and Detail Views.

Django Class Based Generic Views

Django is the most popular web framework of Python, which is used in rapid web application development. It provides a built-in interface that makes it easy to work with it. It is also known as the **Batteries** included framework because it offers built-in facilities for each operation.

Most of us may be already familiar with the function-based views and know how to handle the requests using the function-based view. If you don't familiar with them, visit our [Django tutorials](#).

In this tutorial, we will introduce the **Class-Based Generic views**. These are the advanced set of Built-in views and are used to implement the selective CRUD (create, retrieve, update, and delete) operations. Using Class-Based views, we can easily handle the GET, POST requests for a view.

These do not substitute for a function-based view but provide some additional facilities over the function-based view.

Current Time 0:10

/

Duration 18:10

^

Function-Based Views

Function-based views are beginner-friendly; beginners can easily understand them. It is quite easy to understand in comparison to class-based views.

- It is easy to understand and easy to use.
- It provides the explicit code flow.
- Straightforward usage of decorators.

But function-based view can't be extended and also leads to code redundancy.

Class-Based Views

Class-based views can be used in place of function-based views. All the operations handle using Python objects instead of functions. They provide some excellent example over the function-based views. Class-based views can implement CRUD operation in easy manner.

- It follows the DRY convention of Django.
- We can extend class-based views and can add more functionality according to a requirement using Mixin.
- It allows to inherit another class, can be modified for various use cases.

But these are complex to understand and hard to read. It has implicit code flow.

Perform CRUD (Create, Retrieve, Update, Delete) Using Class Based Views

We will create the project name **Hello** which includes an app named **sampleapp**. If you don't know how to create app in django, visit [Django App](#) tutorial.

In the application, we will create an Employee model in the **model.py** file.

Example -

```

1. from django.db import models
2.
3. # Create your models here.
4.
5. class Employee(models.Model):
6.     first_name = models.CharField(max_length=30)
7.     last_name = models.CharField(max_length=30)
8.     mobile = models.CharField(max_length=10)
9.     email = models.EmailField()
10.
11.     def __str__(self):
12.         return "%s %s" % (self.first_name, self.last_name)

```

DetailView

DetailView is different from **ListView** as it displays the one instance of a table in the database. Django automatically assigns a primary key for each entry, and we need to specify the **<pk>** in the request. **DetailView** will automatically perform everything. The implementation of **DetailView** is the same as the **ListView**, and we need to create **modelname_detail.html**.

Let's understand the following implementation of **DetailView**.

View.py

```

1. from django.views.generic.detail import DetailView
2.
3. class EmployeeDetail(DetailView):
4.     model = Employee

```

url.py

```

1. from django.urls import path
2. from .views import EmployeeCreate, EmployeeDetail, EmployeeRetrieve
3.
4. urlpatterns = [
5.     path("", EmployeeCreate.as_view(), name = 'EmployeeCreate'),
6.     path('retrieve/', EmployeeRetrieve.as_view(), name = 'EmployeeRetrieve'),
7.     path('retrieve/<int:pk>', EmployeeDetail.as_view(), name = 'EmployeeDetail')
8. ]

```

sampleapp/template/employee_detail.html

```

1. {% extends 'base.html' %}
2.
3. {% block content %}
4.

```

5. `<h1>{{ object.first_name }} {{ object.last_name }}</h1>`
- 6.
7. `<p>{{ object.email }}</p>`
8. `<p>{{ object.mobile }}</p>`
- 9.
10. `{% endblock content %}`

We created a new employee and it assigned 2 as primary key. Running the server and provide the request with primary key.

Output:

