|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| DJANGO  Python web framework |
| |  |  |  | | --- | --- | --- | | Abhijith M | 1/1/20 | Python | |

Table of Contents

[Django 4](#_Toc61810735)

[Install django 4](#_Toc61810736)

[Create new project 4](#_Toc61810737)

[website design Basics 8](#_Toc61810738)

[HTML 8](#_Toc61810739)

[CSS 8](#_Toc61810740)

[JS 8](#_Toc61810741)

[Templates 8](#_Toc61810742)

[Create a template 8](#_Toc61810743)

[Step 1: Create the Templates Folder. In this step we'll create the templates directory. 9](#_Toc61810744)

[Step 2: Create new html file. 9](#_Toc61810745)

[Step 3: Render your template to Django app. 9](#_Toc61810746)

[The final step is to start server 9](#_Toc61810747)

[Passing Context to Templates 10](#_Toc61810748)

[Create a base template 12](#_Toc61810749)

[Generic view 14](#_Toc61810750)

[For loop in tempalte 15](#_Toc61810751)

[Adding static files 15](#_Toc61810752)

[URL 16](#_Toc61810753)

[Database 17](#_Toc61810754)

[How to create superuser in Django? 18](#_Toc61810755)

[How does Django connect to MySQL database? 18](#_Toc61810756)

[Step 1 — Create the Initial Django Project Skeleton. 19](#_Toc61810757)

[Step 2 — Initialize MySQL Database. 19](#_Toc61810758)

[Step 3 — Create the Database. 19](#_Toc61810759)

[STEP 5 – APPLY MIGRATIONS 20](#_Toc61810760)

[Step 4 — Create superuser & Add database to admin view. 21](#_Toc61810761)

[Start the development server 21](#_Toc61810762)

[Make the poll app modifiable in the admin 22](#_Toc61810763)

[Explore the free admin functionality 22](#_Toc61810764)

[Models 23](#_Toc61810765)

[Fileds 23](#_Toc61810766)

[Meta 25](#_Toc61810767)

[Combination of two field unique 25](#_Toc61810768)

[Foreign Key 26](#_Toc61810769)

[Django Database API 27](#_Toc61810770)

[Sort items in Db 30](#_Toc61810771)

[Custom user registration 30](#_Toc61810772)

[Namespace and HTTP 404 shortcut 34](#_Toc61810773)

[Simple Login Form 35](#_Toc61810774)

[Model Form 37](#_Toc61810775)

[Authentication 38](#_Toc61810776)

[Form 43](#_Toc61810777)

[HTML Form 47](#_Toc61810778)

[Django Form 48](#_Toc61810779)

[Validation in email 49](#_Toc61810780)

[Login page 50](#_Toc61810781)

[Check authenticated 51](#_Toc61810782)

[Register Form 52](#_Toc61810783)

[Session 55](#_Toc61810784)

[Session key 56](#_Toc61810785)

[Stream video to webpage 57](#_Toc61810786)

[Video streaming using Javascript 59](#_Toc61810787)

[Stream video Xframe option error 60](#_Toc61810788)

[Upload Video 61](#_Toc61810789)

[Enabling https 64](#_Toc61810790)

[Method 1 65](#_Toc61810791)

[method 2 65](#_Toc61810792)

[Publish your site in a network or public domain – 0.0.0.0:80 65](#_Toc61810793)

[Django payment gateway – razopay 66](#_Toc61810794)

[Integrating Razorpay with django 66](#_Toc61810795)

[***#1. Open razorpay Dashboard and collect the KeyId and SecretKey.*** 67](#_Toc61810796)

[***#2. Create Client Instance at Server-side for communicating with Razorpay.*** 67](#_Toc61810797)

[***#3. Create order by using ‘client.order.create’ method.*** 67](#_Toc61810798)

[***#4. Store the ID and Status returned by the razorpay*** 68](#_Toc61810799)

[***#5. Pass the necessary parameters and order\_id to html page.*** 68](#_Toc61810800)

[***#6. Copy the javascript from here!!! and paste it to the html page where you want the payment to be done!!!*** 69](#_Toc61810801)

[***#7. Verifying the signature!!!*** 71](#_Toc61810802)

[Part 2 71](#_Toc61810803)

[REST API 72](#_Toc61810804)

[Create new project 72](#_Toc61810805)

[Cmd http client 74](#_Toc61810806)

[Generic view 75](#_Toc61810807)

[Basics of arguments 76](#_Toc61810808)

[Mixin 77](#_Toc61810809)

[CRUD without REST framework 78](#_Toc61810810)

[Read 79](#_Toc61810811)

[Create 85](#_Toc61810812)

[csrf token disabling 85](#_Toc61810813)

[store database 87](#_Toc61810814)

[Update 88](#_Toc61810815)

[Delete 90](#_Toc61810816)

[By using single baseurl 90](#_Toc61810817)

[With REST Frame work 95](#_Toc61810818)

[Reference 97](#_Toc61810819)

# Django

What is Django used for?

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. A web application framework is a toolkit of all components need for application development.

Is Django backend or frontend?

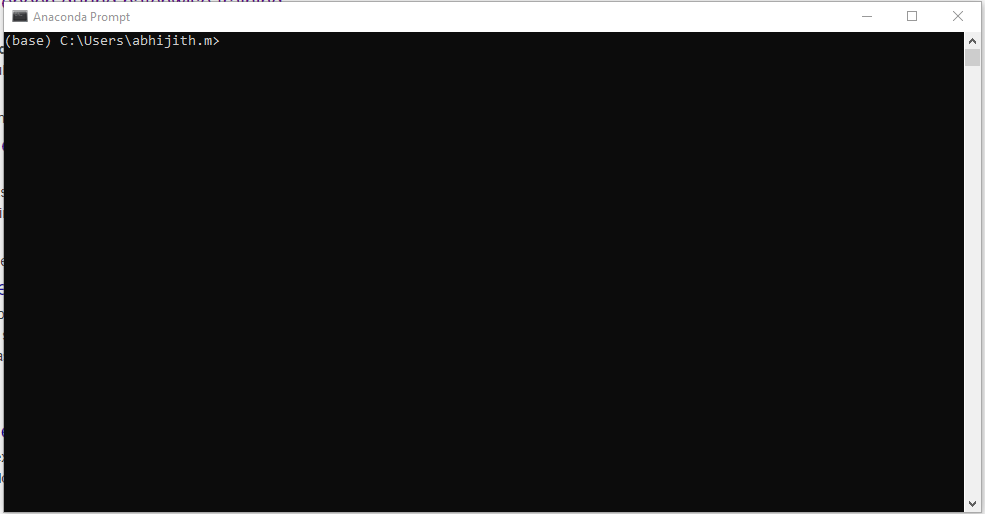
Django is a framework, not a language. Python is the language in which Django is written. Django is a collection of Python libs allowing you to quickly and efficiently create a quality Web application, and is suitable for both frontend and backend.

## Install django

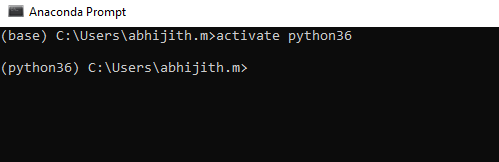
* Open anaconda prompt
* Create a new environment in anaconda prompt
* conda create -n python36 python=3.6
* This command will create a new environment in anaconda. Name of environment is python36
* then activate your newly created environment using following command.
* activate python36
* Now the environment is set. You can do all of Django projects in this environment.
* Simply an environment is a folder containing all necessary packages of your project
* You should need to install Django package in this environment.
* pip install django

## Create new project

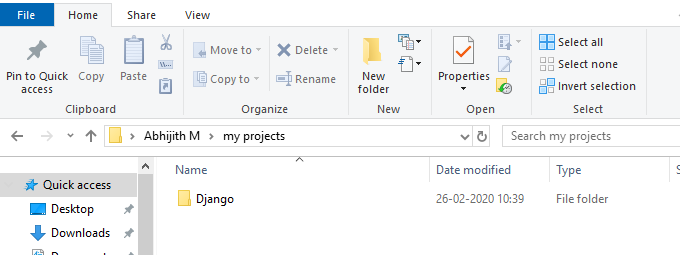
* First step is to activate environment in anaconda prompt
* Open anaconda prompt



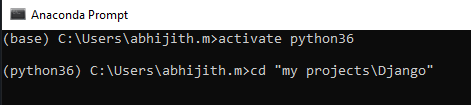
* activate python36



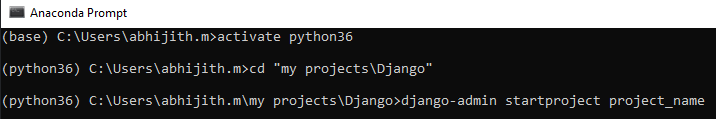
* Create a folder for Django projects in user directory

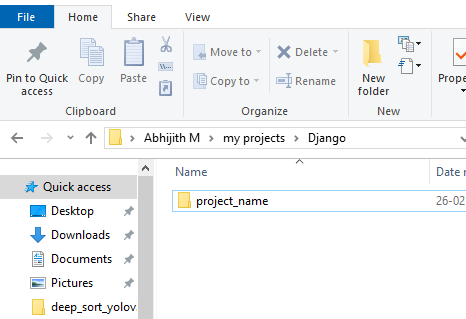


* Move to project directory
* To change directory in command prompt use the following code
* cd “path to your project directory”

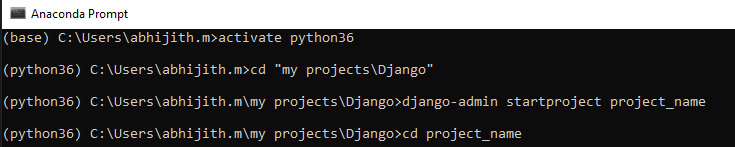


* *(python36) C:\Users\abhijith.m\Django>django-admin startproject project\_name*
* “django-admin startproject project\_name “command will create a new project with name project\_name. A new folder named “project\_name” will create to the current directory.

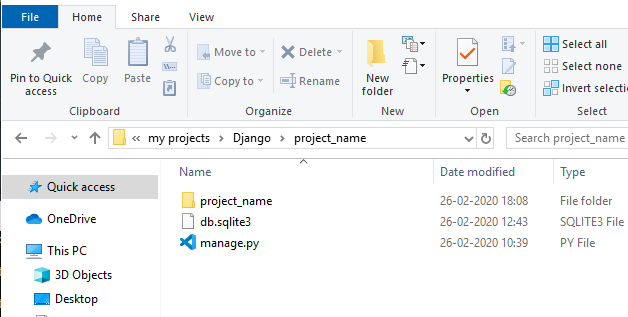




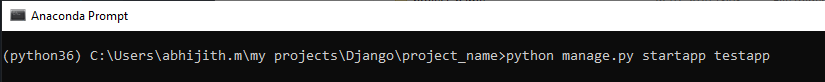
* *(python36) C:\Users\abhijith.m\Django>cd project\_name*
* Cd project\_name will change your current directory in prompt to project\_name

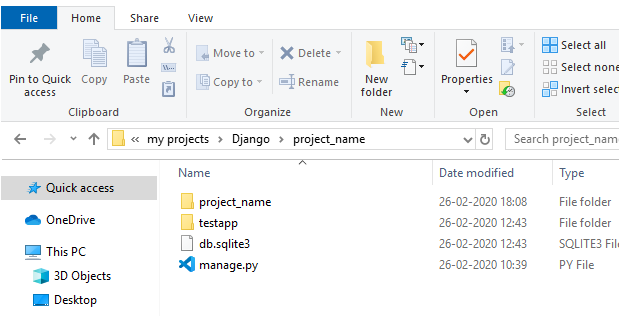


* When we first initialize the project these files are automatically created in project folder.

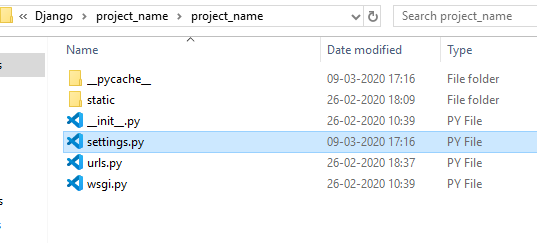


* *(python36) C:\Users\abhijith.m\Django\withoutrest>python manage.py startapp testapp*
* Manage.py is a Django controller file. “python manage.py startapp testapp” this command will create a new app with name testapp





* Open project folder - ../project\_name/project\_name/
* Open settings.py file



* Add your app name to INSTALLED\_APPS

# Application definition

INSTALLED\_APPS = [

    'count\_people.apps.CountPeopleConfig',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'testapp', # <-- your app name

]

* Now the project creation is almost completed. But the view is not created. For that we should need some website design knowledge.

# website design Basics

## HTML

## CSS

## JS

# Templates

What is Jinja template in Django?

**Template** Engines. **Template** engines take in tokenized strings and produce rendered strings with values in place of the tokens as output. **Templates** are typically used as an intermediate format written by developers to programmatically produce one or more desired output formats, commonly HTML, XML or PDF.

What is template engine in Django?

**Django's template engine** provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. **Templates** can be maintained by anyone with an understanding of HTML; no knowledge of Python is required.

What is Jinja template in Django?

**Jinja** is a web **template** engine for the Python programming language and is licensed under a BSD License created by Armin Ronacher. It is similar to the **Django template** engine but provides Python-like expressions while ensuring that the **templates** are evaluated in a sandbox.

Does Django need HTML?

But when you ask about **Django** it **is** a web development framework, So you should learn **html**, css and javascript **is** very essential. ... But when you ask about **Django** it **is** a web development framework, So you should learn **html**, css and javascript **is** very essential.

What is a template used for?

A **template** is a form, mold, or pattern **used** as a guide to making something. ... A document in which the standard opening and closing parts are already filled in is a **template** that you can copy and then fill in the variable parts.

How do I add a template to Django project?

## Create a template

Step 1: Create the Templates Folder. In this step we'll create the templates directory.

Step 2: Create new html file.

Step 3: Render your template to Django app.

* After creating a new project do the following steps to create new template.

### Step 1: Create the Templates Folder. In this step we'll create the templates directory.

* 1. Open the project folder (../project\_name)
  2. open “testapp” folder
  3. In this folder you need to create new folder named “templates” (location:- ../project\_name/testapp/)

### Step 2: Create new html file.

2.1 Create a new “index.html” file in (../project\_name/testapp/templates)

- index.html file

<html>

    <head>

    </head>

    <body>

        <h1>Django template</h1>

    </body>

</html>

### Step 3: Render your template to Django app.

* To show this index page in website do the following steps,
  1. Create new view in views.py (location:- ../project\_name/testapp/views.py)

from django.shortcuts import render

# Create your views here.

def MyView(request):

    return render(request,template\_name="index.html", context={})

* 1. Open urls.py file (location:- ../project\_name/project\_name/urls.py)
* Import views to this file

from testapp import views

* Add url path to map you view
* urls.py file >

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyView),

]

### The final step is to start server

* open anaconda prompt
* move to project folder
* (python36) C:\Users\abhijith.m > cd my projects\\Django\\project\_name
* This is my project location you can replace this location with your project path
* Then start running your server
* (python36) C:\Users\abhijith.m\my projects\Django\project\_name>python manage.py runserver

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

You have 17 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.

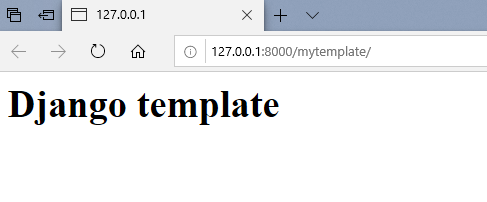
February 26, 2020 - 12:43:57

Django version 2.2.3, using settings 'project\_name.settings'

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

Quit the server with CTRL-BREAK.

* Open your web browser
* Search this url (<http://127.0.0.1:8000/mytemplate/>) in browser



## Passing Context to Templates

What is context in Django?

When you use a Django Template, it is compiled once (and only once) and stored for future use, as an optimization. A template can have variable names in double curly braces, such as {{ myvar1 }} and {{ myvar2 }}.

A Context is a dictionary with variable names as the key and their values as the value. Hence, if your context for the above template looks like: {myvar1: 101, myvar2: 102}, when you pass this context to the template render method, {{ myvar1 }} would be replaced with 101 and {{ myvar2 }} with 102 in your template. This is a simplistic example, but really a Context object is the context in which the template is being rendered.

What are views in Django?

Django views are a key component of applications built with the framework. At their simplest they are a Python function or class that takes a web request and return a web response. Views are used to do things like fetch objects from the database, modify those objects if needed, render forms, return HTML, and much more.

What is render in Python?

Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text. render() is the same as a call to render\_to\_response() with a context\_instance argument that forces the use of a RequestContext.

Why we use render in Django?

render() Combines a given template with a given context dictionary and returns an HttpResponse object with that rendered text. Django does not provide a shortcut function which returns a TemplateResponse because the constructor of TemplateResponse offers the same level of convenience as render() .

How to pass context to template?

* views.py >

from django.shortcuts import render

# Create your views here.

def MyView(request):

    myvar1 = "id1"

    myvar2 = "id2"

    context = {myvar1: 101, myvar2: 102}

    return render(request,template\_name="index.html", context=context)

* Here we passing two variables *myvar1* and *myvar2*. Value of *myvar1* is “id1” so in template page we use *id1* for accessing the value of *id1.* The {{ varname }} is the syntax used to read the value of Django variable.
* In index.html page update the code >

<html>

    <head>

    </head>

    <body>

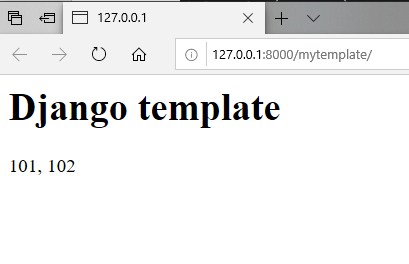
        <h1>Django template</h1>

        {{id1}}, {{id2}}

    </body>

</html>

* Open your browser and refresh the webpage (<http://127.0.0.1:8000/mytemplate/>)



## Create a base template

What is base template in Django?

A base template is the most basic template that you extend on every page of your website. You used the template tag {% block %} to make an area that will have HTML inserted in it. That HTML will come from another template that extends this template ( base. html ).

What is Template inheritance in Django?

Template inheritance. The most powerful – and thus the most complex – part of Django's template engine is template inheritance. Template inheritance allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override.

How can I create base template in Django?

1. Create a new base.html page in template folder

project\_name

└───testapp

└───templates

base.html

index.html

1. In base.html page add the following code.

<html>

    <head>

        <title> {% block title %} {% endblock %}</title>

    </head>

    <body>

        {% block index\_page %} {% endblock %}

    </body>

</html>

1. In index.html update the code >

{% extends 'base.html' %}

{% block title %}

Index

{% endblock %}

{% block index\_page %}

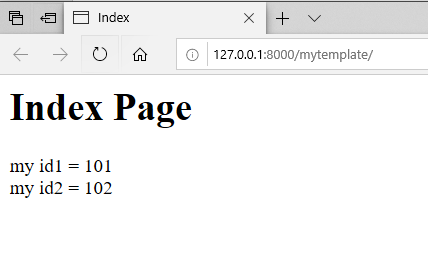
    <h1>Index Page</h1>

    my id1 = {{id1}} <br>

    my id2 = {{id2}}

{% endblock %}

* Refresh browser



* (music/templates/music) > create a new file ‘base.html’ inside music

<!DOCTYPE html>

<html>

<head>

<title>{% block title %} {% endblock %}</title>

</head>

<body>

<nav class="navbar navbar-inverse">…

</nav>

{% block body %}

{% endblock %}

</body>

</html>

* (music/templates/music/index.html) >

{% extends 'music/base.html' %}

{% block title %} Album title {% endblock %}

{% block body %}

{% if all\_albums %}…

{% endif %}

{% endblock %}

* (music/templates/music/details.html) > also like index.html

## Generic view

What is generic Django?

Django offers an easy way to set those simple views that is called generic views. Unlike classic views, generic views are classes not functions. Django offers a set of classes for generic views in Django. views. generic, and every generic view is one of those classes or a class that inherits from one of them.

What are class-based views?

Class-based views. A view is a callable which takes a request and returns a response. This can be more than just a function, and Django provides an example of some classes which can be used as views. These allow you to structure your views and reuse code by harnessing inheritance and mixins.

What is get and post method in Django?

Django's login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response. GET , by contrast, bundles the submitted data into a string, and uses this to compose a URL.

What is as\_view in Django?

as\_view is the function(class method) which will connect my MyView class with its url. From django docs: classmethod as\_view(\*\*initkwargs) Returns a callable view that takes a request and returns a response: You just can't use class-based views like you could in normal function-based views.

How can I create a generic view in Django?

* Open views.py, delete *MyView* function and create new class-based view
* from django.views.generic import View – is a default function in Django for Generic view.

from django.shortcuts import render

from django.views.generic import View

# Create your views here.

class MyGenericView(View):

    def get(self, request, \*args, \*\*kwargs):

        myvar1 = "id1"

        myvar2 = "id2"

        context = {myvar1: 101, myvar2: 102}

        return render(request, template\_name="index.html", context=context)

* Now the function name is changed so we need to map this function in urls.py page
* In urls.py update path

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyGenericView.as\_view()),

]

* Refresh browser. No changes will be found. But the code works fine.

## For loop in tempalte

## Adding static files

* Create a ‘static’ folder in parent directory

project\_name

└───static

└───testapp

└───templates

base.html

index.html

* Open settings.py (location:- ../project\_name/project\_name/settings.py)
* Add this code to the bottom of setting.py file

STATIC\_URL = '/static/'

# Add these new lines

STATICFILES\_DIRS = (

os.path.join(BASE\_DIR, 'static'),

)

STATIC\_ROOT = os.path.join(BASE\_DIR, 'staticfiles')

* In static folder create folder css and js
* Inside this folder you can add your javascript and css files

project\_name

└───static

└───css

style.css

└───js

myscript.js

* {% load static %} will load static file link to html page

{% load static %}

<link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" >

* Complete code of base.html page

<html>

    <head>

        <title> {% block title %} {% endblock %}</title>

        {% load static %}

        <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" >

    </head>

    <body>

        {% block index\_page %} {% endblock %}

    </body>

</html>

* style.css (location:- ../project\_name/static/css/style.css)

body {

    background-color: antiquewhite;

}

* refresh your browser

## URL

What is url naming in Django?

Django offers a way to name urls so it's easy to reference them in view methods and templates. The most basic technique to name Django urls is to add the name attribute to url definitions in urls.py .

How do I name a url in Django?

* Add the name attribute to url definitions in urls.py .

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

    path('admin/', admin.site.urls),

    path('mytemplate/', views.MyGenericView.as\_view(), name="mytemplate"

),

]

Using {% url ??? %} in django templates

* If you add a name attribute to the path, you can refer this name in html page.

<form action="{% url ‘logout\_view’ %}">

# Database

What is Database explain?

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Computer databases typically contain aggregations of data records or files, containing information about sales transactions or interactions with specific customers.

What is database example?

A Microsoft Excel spreadsheet or Microsoft Access are good examples of desktop database programs. These programs allow users to enter data, store it, protect it, and retrieve it when needed. Relational databases are the most common database systems. They include databases like SQL Server, Oracle Database, Sybase, Informix, and MySQL.

What is SQL and why it is used?

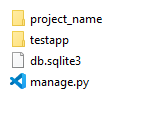
SQL is used to communicate with a database. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

What is called in SQL?

SQL is an abbreviation for structured query language, and pronounced either see-kwell or as separate letters. SQL is a standardized query language for requesting information from a database. The original version called SEQUEL (structured English query language) was designed by an IBM research center in 1974 and 1975.

What is DB(DataBase) sqlite3 in Django?

By default, when we made our first app and started the server you must have seen a new file in your project directory, named as ‘db.sqlite3’. The file is database file where all the data that you will be generating will be stored. It is a local file as Django is a server-side framework and it treats your computer as the host when you actually run the server in command line/terminal.



Does Django use SQL?

Yes and no. To actually use Django, you don't need to use SQL. In fact, Django makes it really really hard for you to use SQL directly with it. It comes with an ORM (Object Relational Mapper), which does most of the behind the scenes work for you.

What is ORM how it's important Django?

One of the most powerful features of Django is its Object-Relational Mapper (ORM), which enables you to interact with your database, like you would with SQL. In fact, Django's ORM is just a pythonical way to create SQL to query and manipulate your database and get results in a pythonic fashion.

How connect Django to xampp?

1. Create a database in phpmyadmin.
2. In your settings.py edit the DATABASES DATABASES = { 'default': { 'ENGINE': 'django.db.backends.mysql', 'NAME': 'yourDbName', 'USER': 'root', 'PASSWORD': '', 'HOST': 'localhost', 'PORT': '3306', #my port is 3306 } }
3. Finally, in your terminal.

## How to create superuser in Django?

* Create admin

(python36) C:\Users\abhijith.m\Desktop\website>python manage.py createsuperuser

Username (leave blank to use 'abhijith.m'): admin

Email address: admin@example.com

Password:

Password (again):

Superuser created successfully.

* Open browser
* Go to: <http://127.0.0.1:8000/admin/>
* Enter username and password
* Open music/admin.py

from django.contrib import admin

from .models import Album

admin.site.register(Album)

* Again Go to: <http://127.0.0.1:8000/admin/>

## How does Django connect to MySQL database?

Step 1 — Create the Initial Django Project Skeleton.

Step 2 — Initialize MySQL Database.

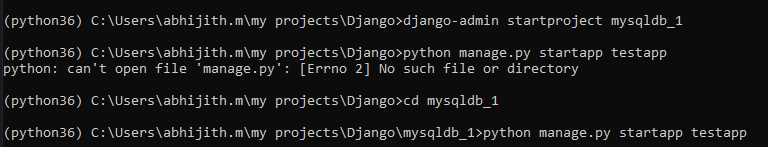
Step 3 — Create the Database.

Step 4 — Create superuser & Add database to admin view.

Step 5 — Test MySQL Connection to Application.

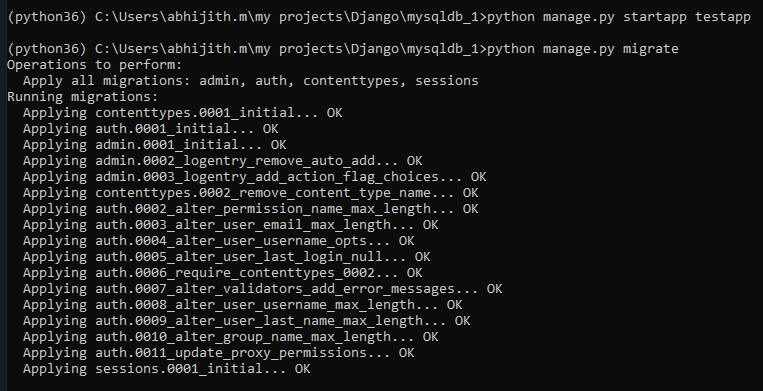
### Step 1 — Create the Initial Django Project Skeleton.

* Create a new project – project name = mysqldb\_1



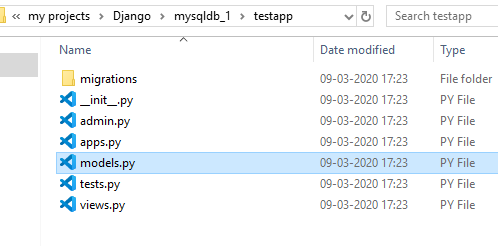
### Step 2 — Initialize MySQL Database.

* Python manage.py migrate
* This command will create all migrations for Django DB initialization.



### Step 3 — Create the Database.

* Open models.py in testapp folder



* Add the following code to create new table in database.

from django.db import models

# Create your models here.

class UserDB(models.Model):

    id = models.AutoField(primary\_key=True)

    name = models.CharField(max\_length=500)

    email = models.EmailField(max\_length=254)

### STEP 5 – APPLY MIGRATIONS

* Edit the testapp/settings.py file and add that dotted path to the INSTALLED\_APPS setting. It’ll look like this:

INSTALLED\_APPS = [

    'testapp',

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

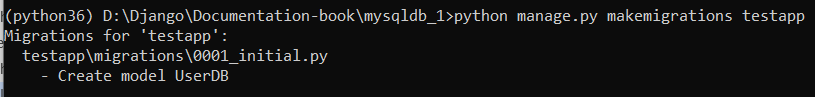
    'django.contrib.messages',

    'django.contrib.staticfiles',

]

* Now we need to apply migrations to db

$ python manage.py makemigrations testapp



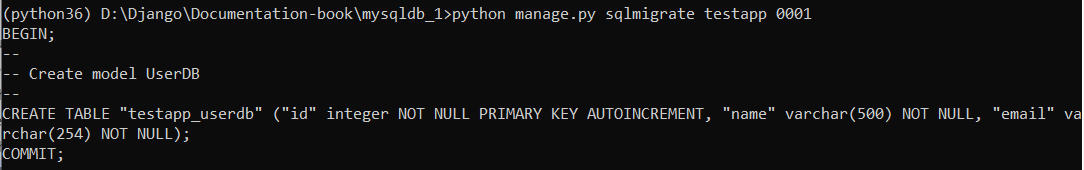
$ python manage.py migrate testapp

What is the difference between Makemigrations and migrate in Django?

migrate, which is responsible for applying migrations, as well as unapplying and listing their status. makemigrations, which is responsible for creating new migrations based on the changes you have made to your models. ... You have unapplied migrations; your app may not work properly until they are applied.

There’s a command that will run the migrations for you and manage your database schema automatically - that’s called migrate, and we’ll come to it in a moment - but first, let’s see what SQL that migration would run. The sqlmigrate command takes migration names and returns their SQL:

$ python manage.py sqlmigrate testapp 0001



### Step 4 — Create superuser & Add database to admin view.

* In order to log into the admin site, we need a user account with Staff status enabled. In order to view and create records we also need this user to have permissions to manage all our objects. You can create a "superuser" account that has full access to the site and all needed permissions using manage.py.

$ python3 manage.py createsuperuser

Enter your desired username and press enter.

Username: admin

You will then be prompted for your desired email address:

Email address: admin@example.com

The final step is to enter your password. You will be asked to enter your password twice, the second time as a confirmation of the first.

Password: \*\*\*\*\*\*\*\*\*\*

Password (again): \*\*\*\*\*\*\*\*\*

Superuser created successfully.

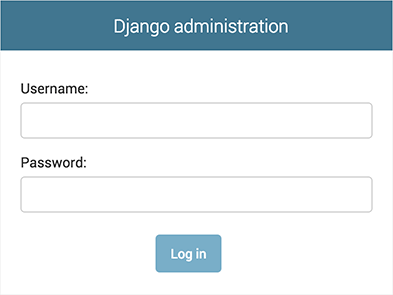
### Start the development server

If the server is not running start it like so:

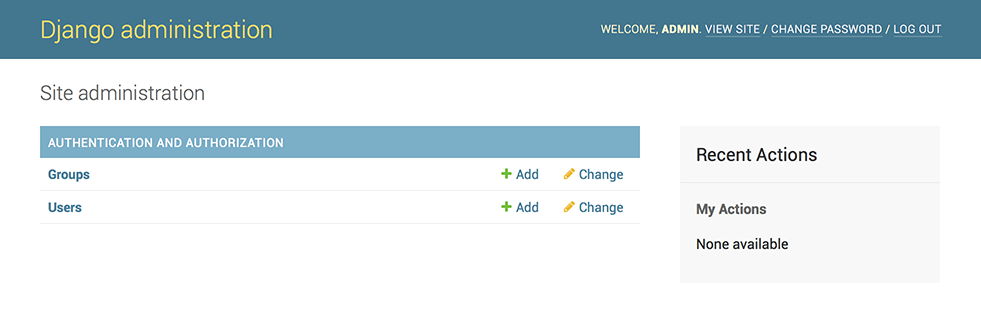
$ python manage.py runserver

Now, open a Web browser and go to “/admin/” on your local domain – e.g., http://127.0.0.1:8000/admin/. You should see the admin’s login screen:

Django admin login screen



After successful login admin site would be like this,



### Make the poll app modifiable in the admin

Only one more thing to do: we need to tell the admin that Users objects have an admin interface. To do this, open the testapp/admin.py file, and edit it to look like this:

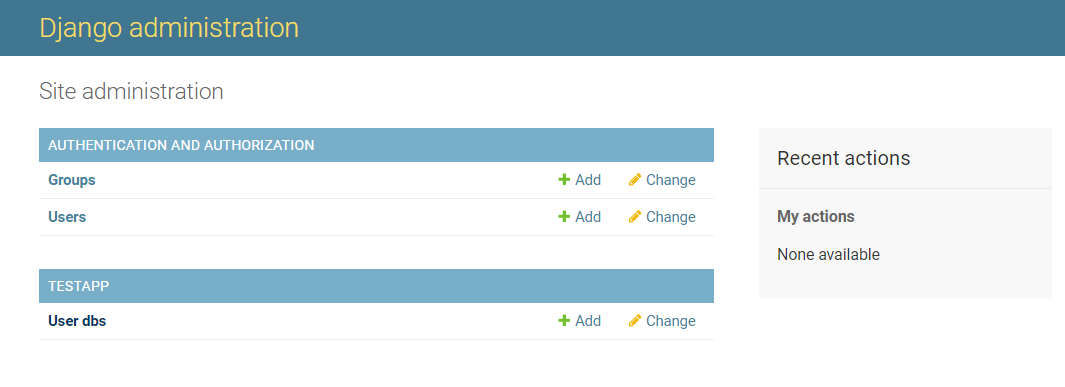
from django.contrib import admin

from .models import UserDB

admin.site.register(UserDB)

### Explore the free admin functionality

Now that we’ve registered Question, Django knows that it should be displayed on the admin index page:



## Models

WHAT IS models model in Django?

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table. With all of this, Django gives you an automatically-generated database-access API.

What is the use of models in Django?

Django web applications access and manage data through **Python** objects referred to as models. Models define the structure of stored data, including the field types and possibly also their maximum size, default values, selection list options, help text for documentation, label text for forms, etc.

How do Django models work?

In Django, the model is the object mapped to the database. When you create a model, Django executes SQL to create a corresponding table in the database (Figure 4-2) without you having to write a single line of SQL. Django prefixes the table name with the name of your Django application.

Where is Django database stored?

In the settings.py file, there is a variable called DATABASES . It is a dict, and one of its keys is default , which maps to another dict. This subdict has a key, NAME , which has the path of the SQLite database.

what is model fields in Django?

Model and each field of the model class represents a database field (column). Django provides us a database-abstraction API which allows us to create, retrieve, update and delete a record from the mapped table. Model is defined in Models.py file.

How do I get model fields in Django?

To get the value of each field from the instance, use getattr(instance, field.name) . True by default. Recursively includes fields defined on parent classes. If set to False, get\_fields() will only search for fields declared directly on the current model.

### Fileds

The most important part of a model – and the only required part of a model – is the list of database fields it defines. Fields are specified by class attributes.

Field types¶

Each field in your model should be an instance of the appropriate Field class. Django uses the field class types to determine a few things:

The column type, which tells the database what kind of data to store (e.g. INTEGER, VARCHAR, TEXT).

The default HTML widget to use when rendering a form field (e.g. <input type="text">, <select>).

The minimal validation requirements, used in Django’s admin and in automatically-generated forms.

Django ships with dozens of built-in field types; you can find the complete list in the model field reference. You can easily write your own fields if Django’s built-in ones don’t do the trick; see Writing custom model fields.

What is CharField in Django?

CharField is a commonly-defined field used as an attribute to reference a text-based database column when defining Model classes with the Django ORM. The Django project has wonderful documentation for CharField and all of the other column fields.

What is AutoField in Django?

According to documentation, An AutoField is an IntegerField that automatically increments according to available IDs. One usually won't need to use this directly because a primary key field will automatically be added to your model if you don't specify otherwise.

IS NOT NULL in Django?

In most cases, it's redundant to have two possible values for “no data;” the Django convention is to use the empty string, not NULL . One exception is when a CharField has both unique=True and blank=True set.

AutoField

id = models.AutoField(primary\_key=True)

ForeignKey

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

FileField

video = models.FileField()

BooleanField

train\_status = models.BooleanField(default=True)

Text field

field\_name = models.TextField()

Email field

email = models.EmailField(max\_length=254)

Recursive relation

next\_item\_id = models.ForeignKey('self', on\_delete=models.CASCADE, blank=True)

DateTimeField

from datetime import date

date\_time = models.DateTimeField(auto\_now\_add=True, blank=True)

or

date = models.DateTimeField(default=datetime.now, blank=True)

File field

image\_url =  models.FileField()

in view

album\_logo = request.FILES['album\_logo']

## Meta

What is Meta inner class in Django models?

This is just a class container with some options (metadata) attached to the model. It defines such things as available permissions, associated database table name, whether the model is abstract or not, singular and plural versions of the name etc.

What is class meta Python?

A metaclass in Python is a class of a class that defines how a class behaves. A class is itself an instance of a metaclass. A class in Python defines how the instance of the class will behave. In order to understand metaclasses well, one needs to have prior experience working with Python classes.

What is self \_\_ class \_\_ in Python?

self. \_\_class\_\_ is a reference to the type of the current instance. For instances of abstract1 , that'd be the abstract1 class itself, which is what you don't want with an abstract class.

How do I make a field unique in Django?

Of course, you should have an id tag, which is always unique, because it's a primary key. However, certain fields may also need to be unique, because certain fields may need to be unique without any repetition. So, to make a field unique in Django is very basic. You just set the unique attribute to true.

What is unique together in Django?

In this article, we show how to make fields of a database table (model) unique together in Django. This means that, for a given database table, there cannot be multiple rows of the same data for the fields that are unique together.

### Combination of two field unique

class ItemsDB(models.Model):

    item\_id = models.CharField( max\_length=200)

    shop = models.ForeignKey(ShopsDB, on\_delete=models.CASCADE)

    item\_name = models.CharField(max\_length=200)

    brand = models.CharField(max\_length=200)

    image = models.CharField(max\_length=500)

    price = models.IntegerField()

    measurement\_unit = models.CharField(max\_length=50)

    description = models.TextField()

    validity = models.DateField()

    class Meta:

        unique\_together = ('item\_id','shop')

    def \_\_str\_\_(self):

        return self.item\_name

### Foreign Key

What is Django foreign key?

ForeignKey is a Django ORM field-to-column mapping for creating and working with relationships between tables in relational databases. ForeignKey is defined within the django. related module but is typically referenced from django.

What is On\_delete models cascade in Django?

The on\_delete method is used to tell Django what to do with model instances that depend on the model instance you delete. (e.g. a ForeignKey relationship). The on\_delete=models. CASCADE tells Django to cascade the deleting effect i.e. continue deleting the dependent models as well. Here's a more concrete example.

Can foreign key be null?

A foreign key containing null values cannot match the values of a parent key, since a parent key by definition can have no null values. However, a null foreign key value is always valid, regardless of the value of any of its non-null parts. A table can have many foreign keys.

Is foreign key one to many?

A foreign key relationship could be one-to-one (a record in one table is linked to one and only one record in another table) or one-to-many (a record in one table is linked to multiple records in another table). Foreign keys are only supported on InnoDB tables.

How do I create a one to many relationship in Django?

To handle One-To-Many relationships in Django you need to use ForeignKey . The current structure in your example allows each Dude to have one number, and each number to belong to multiple Dudes (same with Business).

Does a foreign key have to be a primary key?

If you really want to create a foreign key to a non-primary key, it MUST be a column that has a unique constraint on it. A FOREIGN KEY constraint does not have to be linked only to a PRIMARY KEY constraint in another table; it can also be defined to reference the columns of a UNIQUE constraint in another table.

Why foreign keys are not redundant?

Primary and foreign keys are a way in which to constrain related data together to ensure data in your database remains consistent and to ensure no redundant data is in the database as a result of deleting a table or row in one table that affects data in other tables that may perhaps rely on that information.

What is foreign key violation?

Occurs when an update or delete would violate a check constraint on a column. Dependent foreign key constraint violation in a referential integrity constraint. Occurs when an insert or update on a foreign key table is performed without a matching value in the primary key table.

## Django Database API

* Create new project and edit the following files
* Project name: Songs

from django.db import models

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

class Song(models.Model):

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

file\_type = models.CharField(max\_length=10)

song\_title = models.CharField(max\_length=250)

* Open terminal
* Goto project directory
* *python manage.py makemigrations testapp*
* *python manage.py migrate*
* *python manage.py runserver*

First of all, here we import, fetch and update values in Django database through command prompt. After that we will do the same thing in python script.

* Open cmd
* Go to project directory
* Type the following command in cmd.

$ python manage.py shell

(python36) C:\Users\abhijith.m\Desktop\website>python manage.py shell

Python 3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]

Type 'copyright', 'credits' or 'license' for more information

IPython 7.6.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from music.models import Album, Song

In [2]: Album.objects.all()

Out[2]: <QuerySet []>

In [3]: a = Album(artist="Taylor swift", album\_title="Red", genre="Country", album\_logo="https://previews.123rf.com/ima

...: ges/queezzard/queezzard1601/queezzard160100009/50339428-equalizer-icon-can-be-used-as-logo-to-music-album-dj-se

...: t-concert-banner-vector-illustration-.jpg")

In [4]: a.save()

In [5]: a.artist

Out[5]: 'Taylor swift'

In [6]: a.album\_title

Out[6]: 'Red'

In [7]: a.id

Out[7]: 1

In [8]: a.pk

Out[8]: 1

In [9]: b = Album()

In [10]: b.artist = "Myth"

In [11]: b.album\_title = "High School"

In [12]: b.genre = "Punk"

In [13]: b.album\_logo = "https://www.kidsdiscover.com/wp-content/uploads/2014/06/Kid\_Smiling-e1401721146279.

...: jpg"

In [14]: b.save()

In [15]: b.artist

Out[15]: 'Myth'

1. When check all objects in album. It will display like this

In [16]: Album.objects.all()

Out[16]: <QuerySet [<Album: Album object (1)>, <Album: Album object (2)>]>

* Open testapp/models.py
* Include this function inside Artist function

def \_\_str\_\_(self):

return self.album\_title + ' - ' +self.artist

* Full code

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

def \_\_str\_\_(self):

return self.album\_title + ' - ' +self.artist

* *exit() # exit from shell*
* *python manage.py shell*

In [1]: from music.models import Album, Song

In [2]: Album.objects.all()

Out[2]: <QuerySet [<Album: Red - Taylor swift>, <Album: High School - Myth>]>

Filter

In [3]: Album.objects.filter(id=1)

Out[3]: <QuerySet [<Album: Red - Taylor swift>]>

In [5]: Album.objects.filter(artist\_\_startswith='Taylor')

Out[5]: <QuerySet [<Album: Red - Taylor swift>]>

## Sort items in Db

How do I sort data in Django?

In Django 1.4 and newer you can order by providing multiple fields. By default, results returned by a QuerySet are ordered by the ordering tuple given by the ordering option in the model's Meta. You can override this on a per-QuerySet basis by using the order\_by method.

How do you sort a list in Python?

The sort() method sorts the elements of a given list in a specific ascending or descending order. The syntax of the sort() method is: list. sort(key=..., reverse=...)

What is Django filter?

Django-filter is a generic, reusable application to alleviate writing some of the more mundane bits of view code. Specifically, it allows users to filter down a queryset based on a model's fields, displaying the form to let them do this.

User.objects.all().order\_by('artist') # For ascending

User.objects.all().order\_by('-artist') # For descending; Not '-' sign in order\_by method

## Custom user registration

Here we moving to an advanced level of Django database. By default Django provide user registration through createsuperuser command that we discussed before. But that database have some limitations. We cannot add our custom attributes to that model. So in this section we discussing about how to create our custom user model by inheriting django’s default user model.

What is Django user model?

The default User model in Django uses a username to uniquely identify a user during authentication. If you'd rather use an email address, you'll need to create a custom User model by either subclassing AbstractUser or AbstractBaseUser .

Is user authenticated Django?

Django comes with a user authentication system. It handles user accounts, groups, permissions and cookie-based user sessions. This section of the documentation explains how the default implementation works out of the box, as well as how to extend and customize it to suit your project's needs.

How do I override a user model in Django?

Ways to Extend the Existing User Model

1. Using a Proxy Model.
2. Using One-To-One Link With a User Model (Profile)
3. Creating a Custom User Model Extending AbstractBaseUser.
4. Creating a Custom User Model Extending AbstractUser.

How do I authenticate username and password in Django?

Django provides two functions in django.contrib.auth : authenticate() and login() . To authenticate a given username and password, use authenticate() .

Is Django authentication secure?

Django is as secure as any web framework can be. It provides tools and doc to prevent common mistakes causing security problems (csrf, xss, etc.) However, a tool in itself cannot be "secure". The whole platform security depends on the proper use of the tools you choose, and thus is more a matter of developer skills.

* Create new Django project and edit the following files
* (Models.py)>

from django.db import models

from django.contrib.auth.models import User

class UserProfile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

location = models.CharField(max\_length=30)

age = models.IntegerField()

def \_\_str\_\_(self):

return self.user.username

* (admin.py)>

from django.contrib import admin

from .models import UserProfile

admin.site.register(UserProfile)

* Include *testapp* in INSTALLED\_APPS
* (Setings.py)> INSTALLED\_APPS

'testapp',]

* Create file (testapp/forms.py)>

from django import forms

from django.forms import ModelForm

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

from .models import UserProfile

class ExtendedUserCreationForm(UserCreationForm):

email = forms.EmailField(required=True)

first\_name = forms.CharField(max\_length=30)

last\_name = forms.CharField(max\_length=50)

class Meta:

model = User

fields = ('username', 'email', 'first\_name', 'last\_name', 'password1', 'password2')

def save(self, commit=True):

user = super().save(commit=True)

user.email = self.cleaned\_data['email']

user.first\_name = self.cleaned\_data['first\_name']

user.last\_name = self.cleaned\_data['last\_name']

if commit:

user.save()

return user

class UserProfileForm(ModelForm):

class Meta:

model = UserProfile

fields = ("location", "age")

* (views.py)>

from django.shortcuts import render, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib.auth import authenticate, login

from .forms import ExtendedUserCreationForm, UserProfileForm

def index(request):

if request.user.is\_authenticated:

username = request.user.username

else:

username = 'not logged in'

context = {'username' : username}

return render(request, 'example/index.html', context)

def register(request):

if request.method == 'POST':

form = ExtendedUserCreationForm(request.POST or None)

profile\_form = UserProfileForm(request.POST)

if form.is\_valid() and profile\_form.is\_valid():

user = form.save()

profile = profile\_form.save(commit=False)

profile.user = user

profile.save()

username = form.cleaned\_data.get('username')

password = form.cleaned\_data.get('password1')

user = authenticate(username=username, password=password)

login(request, user)

return redirect('index')

else:

form = ExtendedUserCreationForm()

profile\_form = UserProfileForm()

context = {'form': form,

'profile\_form':profile\_form

}

return render(request, "auth/register.html", context)

* Create a html page to display login and registration page
* (testapp/templates/example/index.html)>

{{username}}

{% if user.is\_authenticated %}

Email: {{ user.email }}

Location: {{ user.userprofile.location }}

{% endif %}

* (testapp/templates/auth/register.html)>

<form method="POST" action="{% url 'register' %}">

{% csrf\_token %}

{{ form.as\_p }}

{{profile\_form.as\_p}}

<button type="submit" >Create new account</button>

</form>

* (urls.py)>

from testapp import views

urlpatterns = [

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

path('register/',views.register, name='register'),

path('index/',views.index, name='index'),

]

* <http://127.0.0.1:8000/register/>
* <http://127.0.0.1:8000/admin/> (add user to superuser and staff permission)
* Logout
* <http://127.0.0.1:8000/admin/> login with new user id
* <http://127.0.0.1:8000/index/>

## Namespace and HTTP 404 shortcut

What is the 404 error mean?

A 404 error message is a Hypertext Transfer Protocol (HTTP) status code indicating the server could not find the requested website. In other words, your web browser can connect with the server, but the specific page you're trying to access can't be reached.

How do I change 404 in Django?

First create a templates folder and then add a 404. html file to it. Update settings.py so Django will look for this new templates folder. And finally add some basic text to the file.

How do I stop 404 error?

The simplest and easiest way to fix your 404 error code is to redirect the page to another one. You can perform this task using a 301 redirect. What's 301, you may ask? It's a redirect response code that signals a browser that the content has been transferred to another URL.

* Open *Songs* project
* (music/views.py)> only need to import the following library.

from .models import Album

from django.shortcuts import render, get\_object\_or\_404

* Update

def detail(request, album\_id):

# album = Album.objects.get(pk=album\_id)

album = get\_object\_or\_404(Album, pk=album\_id)

return render(request, 'music/detail.html', {'album':album})

## Simple Login Form

* (music/models.py) Add the following line to song model

is\_favorite = models.BooleanField(default=False)

* Full code

class Song(models.Model):

album = models.ForeignKey(Album, on\_delete=models.CASCADE)

file\_type = models.CharField(max\_length=10)

song\_title = models.CharField(max\_length=250)

is\_favorite = models.BooleanField(default=False)

def \_\_str\_\_(self):

return self.song\_title

* Open cmd
* *python manage.py makemigrations music*
* *python manage.py migrate*
* *python manage.py runserver (Restart server)*
* (music/urls.py)> include the path inside ‘urlpatterns’

path('<int:question\_id>/favorite/', views.favorite, name='favorite'),

* (music/templates/music/details.html) >

<img src="{{ album.album\_logo }}">

<h1>{{ album.album\_title }}</h1>

<h3> {{ album.artist }} </h3>

{% if error\_message %}

<p><strong>{{ error\_message }}</strong></p>

{% endif %}

<form action="{% url 'favorite' album.id %}" method="POST">

{% csrf\_token %}

{% for song in album.song\_set.all %}

<input type="radio" id="song{{ forloop.counter }}" name="song" value="{{ song.id }}">

<label for="song{{ forloop.counter }}">

{{ song.song\_title }}

{% if song.is\_favorite %}

<img href="http://icons.iconarchive.com/icons/icons-land/vista-elements/256/Favorites-icon.png"

width="15" />

{% endif %}

</label><br>

{% endfor %}

<input type="submit" value="Favorite" >

</form>

* (music/views.py)> import ‘Song’

from .models import Album, Song

* Create a new method
* (music/views.py) >

def favorite(request, album\_id):

album = get\_object\_or\_404(Album, pk=album\_id)

try:

selected\_song = album.song\_set.get(pk=request.POST['song'])

except (KeyError, Song.DoesNotExist):

return render(request, 'music/detail.html', {

'album':album,

'error\_message': "You did not select a valid song",

})

else:

selected\_song.is\_favorite = True

selected\_song.save()

return render(request, 'music/detail.html', {'album':album})

## Model Form

* (music/models.py)>

from django.core.urlresolvers import reverse

* Add inside Album

def get\_absolute\_url(self):

return reverse('music:detail', kwargs={'pk': self.pk})

* Album

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.CharField(max\_length=1000)

def get\_absolute\_url(self):

return reverse('music:detail', kwargs={'pk': self.pk})

def \_\_str\_\_(self):

return self.album\_title + ' - ' +self.artist

views.py

from django.http import Http404

from .models import Album, Song

from django.shortcuts import render

def index(request):

#connecting to db

all\_albums = Album.objects.all()

context = {

'all\_albums': all\_albums,

}

return render(request, 'music/index.html', context)

def favorite(request, pk):

try:

album = Album.objects.get(pk=pk)

except Album.DoesnotExist:

raise Http404("Album does not exist")

try:

selected\_song = album.song\_set.get(pk=request.POST['song'])

except (KeyError, Song.DoesNotExist):

return render(request, 'music/detail.html', {

'album':album,

'error\_message': "You did not select a valid song",

})

else:

selected\_song.is\_favorite = True

selected\_song.save()

return render(request, 'music/detail.html', {'album':album})

def detail(request, pk):

try:

album = Album.objects.get(pk=pk)

except Album.DoesnotExist:

raise Http404("Album does not exist")

return render(request, 'music/detail.html', {'album':album})

## Authentication

* Start new project
  1. *django-admin startproject authenticate*
  2. *cd authenticate*
  3. *python manage.py startapp user\_example*
  4. *python manage.py migrate*
  5. *python manage.py createsuperuser*

*Username (leave blank to use 'abhijith.m'): admin*

*Email address: admin@example.com*

*Password:*

*Password (again):*

*Superuser created successfully.*

* (authenticate/urls.py)> inside ‘urlpatterns’

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

path('', include('user\_example.urls')),

path('accounts/', include('django.contrib.auth.urls'))

* (authenticate/settings.py)> inside ‘INSTALLED\_APPS’

'user\_example',

* (user\_example/)> Create another file ‘urls.py’

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name = 'index'),

]

* (user\_example/views.py)>

from django.shortcuts import render

def index(request):

return render(request, 'user\_example/index.html')

* (user\_example/templates/user\_example)> create another file ‘index.html’

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>This is my index page</h1>

</body>

</html>

* (user\_example/templates) create folder registration
* (user\_example/templates/registration) create file ‘login.html’

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<body>

{% if form.errors %}

<p>There's something wrong with what you entered!</p>

{% endif %}

{% if next %}

<p>Hey, you can't access that page.</p>

{% endif %}

<form action="{% url 'login' %}" method="post">

{% csrf\_token %}

<p> Username: {{ form.username }} </p>

<p> Password: {{ form.password }} </p>

<input type="submit" value="login">

<input type="hidden" name= value="{{ next }}">

</form>

</body>

</html>

* (authenticate/settings.py)> add ‘LOGIN\_REDIRECT\_URL’ end of the file

STATIC\_URL = '/static/'

LOGIN\_REDIRECT\_URL = '/'

* In browser <http://127.0.0.1:8000/accounts/login/> login with (username:admin password:\*\*\*\*\*)
* (user\_example/views.py)> add

from django.contrib.auth.forms import UserCreationForm

def register(request):

form = UserCreationForm()

context = {'form' : form}

return render(request, 'registration/register.html', context)

* (userexample/templates/registration/)> create file ‘’registration.html’’

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Register</title>

</head>

<body>

<form method="POST" action="{% url 'register' %}"></form>

{% csrf\_token %}

{% if form.errors %}

<p>There are errors in the form!</p>

{% endif %}

{{ form }}

<input type="submit" value="Register">

</body>

</html>

* (user\_example/urls.py)> add path

path('register', views.register, name = 'register'),

* In browser <http://127.0.0.1:8000/register>
* (user\_example/views.py)> complete code

from django.shortcuts import render, redirect

def index(request):

return render(request, 'user\_example/index.html')

from django.contrib.auth import authenticate, login

from django.contrib.auth.forms import UserCreationForm

def register(request):

if request.method == 'POST':

form = UserCreationForm(request.POST)

if form.is\_valid():

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user = authenticate(username= username, password=password)

login(request, user)

return redirect('index')

context = {'form' : form}

return render(request, 'registration/register.html', context)

* (user\_example/templates/user\_example/index.html)> update

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<h1>This is my index page</h1>

{% if user.is\_authenticated %}

<h2>Your name is: {{ user.username }}</h2>

{% else %}

<h2>You are not logged in.</h2>

{% endif %}

</body>

</html>

# Form

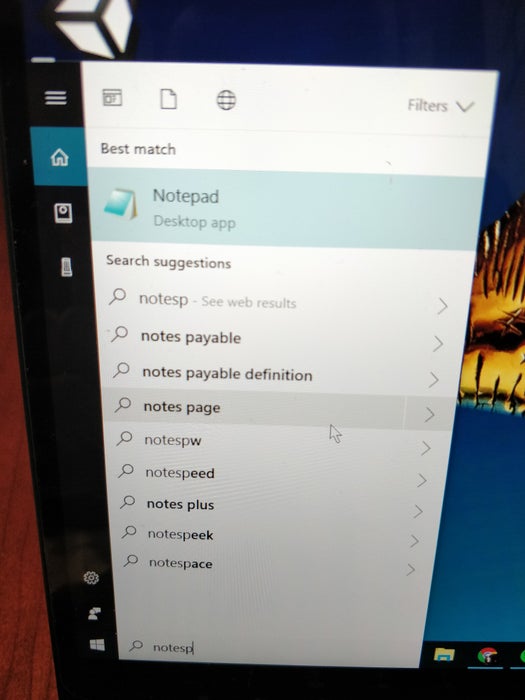
What does form mean in HTML?

An HTML form is a section of a document containing normal content, markup, special elements called controls (checkboxes, radio buttons, menus, etc.), and labels on those controls.

How do I create a Web form?

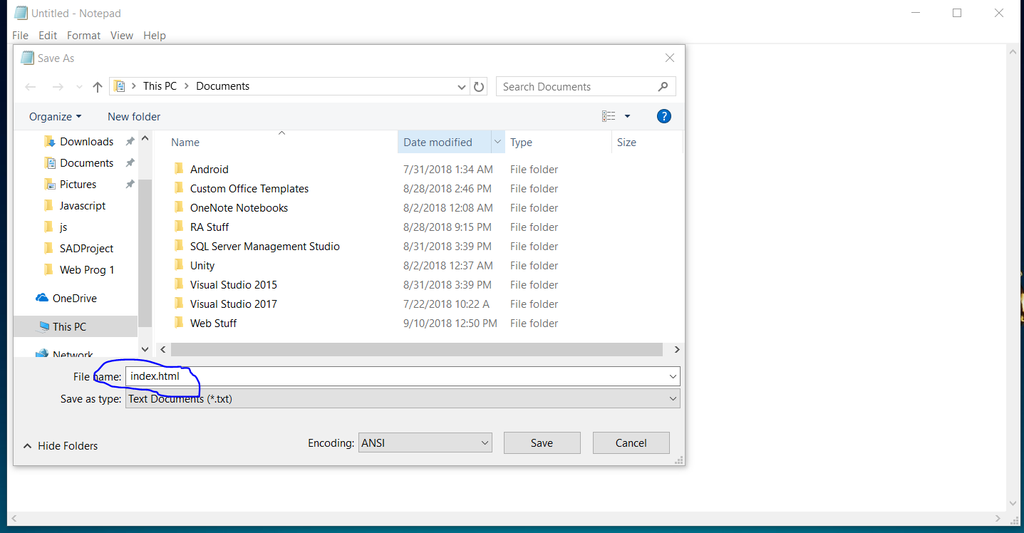
This is a simple instruct on how to make a web form. This would be a small intro on how to make website and how to put content on them and what can be expanded on in the future.

Step 1: Open Notepad

[](https://content.instructables.com/ORIG/FLY/GG23/JM6KPS3J/FLYGG23JM6KPS3J.jpg?auto=webp&frame=1&width=1024&height=1024&fit=bounds&md=a4cee9a579ce51b1c25d03164eb8cea7)

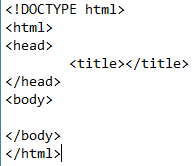
On the search on the task bar, type in notepad and open the application.

Step 2: Save the File As Index.html

[](https://content.instructables.com/ORIG/FOV/NCEV/JM6KPTXS/FOVNCEVJM6KPTXS.png?auto=webp&frame=1&width=1024&fit=bounds&md=185014871234010e1d1185bd50240295)

On notepad, click the "File", then "Save As." When the new window pops up type in "index.html" and make sure the save type is under "All Files." Save this file in your documents folder.

Step 3: Type a Standard Html Page's Format

[](https://content.instructables.com/ORIG/F45/M4I7/JMAV563D/F45M4I7JMAV563D.png?auto=webp&frame=1&fit=bounds&md=f708889bbdab0bdc33e9aa52fcf1f7d5)

Type in the following:

<!DOCTYPE html>

<html>

<head>

<title></title>

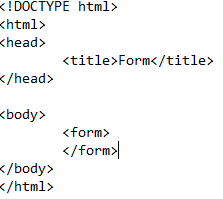
</head>

<body>

</body>

</html>

Step 4: Give the Page a Name and Create the Form

[](https://content.instructables.com/ORIG/F1E/9M46/JMAV67DV/F1E9M46JMAV67DV.png?auto=webp&frame=1&fit=bounds&md=7991122dea5778f77ffc83c27573ed48)

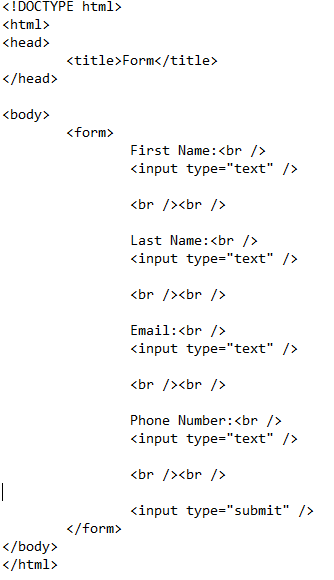
Within the title tag, give the page a name (maybe FORM)

To create the form, type in the following inside the body tag:

<form>

</form>

Step 5: Add Fields to the Form

[](https://content.instructables.com/ORIG/FL4/QQ47/JMAV67KW/FL4QQ47JMAV67KW.png?auto=webp&frame=1&fit=bounds&md=8a4e1d23824c552a6b5acf9712cd8769)

After you type the form tag, add the following inside of it:

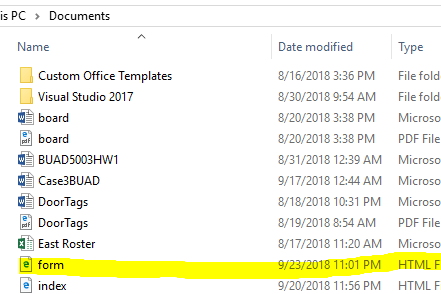
**First Name:**

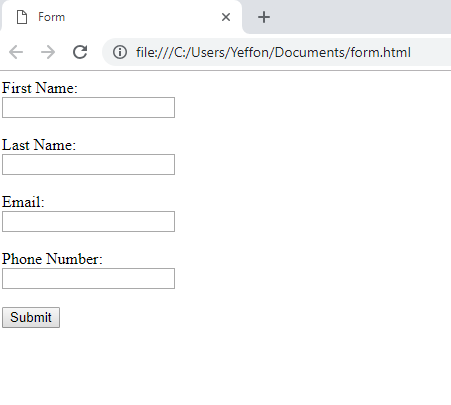
**Last Name:**

**Email:**

**Phone Number:**

Step 6: Go to Your Documents Folder and Open the Web Page

[](https://content.instructables.com/ORIG/FUA/A66Q/JMAV67QE/FUAA66QJMAV67QE.png?auto=webp&frame=1&fit=bounds&md=b473d0b21f02977d97d959639c15d979)

[](https://content.instructables.com/ORIG/FO3/J1OR/JMAV67Q5/FO3J1ORJMAV67Q5.png?auto=webp&frame=1&fit=bounds&md=778934c2fa4f6578da6af7e3e9fad211)

Open your file explorer and open the documents folder. Find the document and open it in a browser of your choice.

What is form and its uses?

Form (HTML) A webform, web form or HTML form on a web page allows a user to enter data that is sent to a server for processing. Forms can resemble paper or database forms because web users fill out the forms using checkboxes, radio buttons, or text fields.

## HTML Form

Are HTML forms still used?

HTML forms were invented and mostly standardized before the advent of asynchronous Javascript and complex web applications. Today, we use form inputs, buttons, and other interaction mechanisms in lots of different ways; but underlying that is a system based on the HTTP request-response paradigm.

What is input in HTML?

The HTML <input> element is used to create interactive controls for web-based forms in order to accept data from the user; a wide variety of types of input data and control widgets are available, depending on the device and user agent.

How to create form in Django and save data to server?

Here we create a login form with database connection. But not saving data to database through this form. If you want to check this, you can create a superuser as we discussed earlier.

* Create a project and an app named testapp
* (testapp/views.py)>

def login\_page(request):

return render(request, "auth/login.html", {})

* (testapp/templates/auth/login.html)>

<form method="POST" >

{% csrf\_token %}

<input type="text" placeholder="Name" name="fullname">

<button type="submit" >Submit</button>

</form>

* (urls.py)>

path('testapp/',views.login\_page),

* Update (views.py)>

def login\_page(request):

if request.method == "POST":

print(request.POST)

print(request.POST.get('fullname'))

return render(request, "auth/login.html", {})

* In browser <http://127.0.0.1:8000/testapp/>

## Django Form

what is django form?

Django provides a Form class which is used to create HTML forms. It describes a form and how it works and appears. Each field of the form class map to the HTML form <input> element and each one is a class itself, it manages form data and performs validation while submitting the form.

* Create file (testapp/forms.py)

from django import forms

class ContactForm(forms.Form):

fullname = forms.Charfield()

email = forms.EmailField()

* In (views.py)>

from .forms import ContactForm

def login\_page(request):

contact\_form = ContactForm()

if request.method == "POST":

print(request.POST)

print(request.POST.get('fullname'))

context = {

'form':contact\_form,

}

return render(request, "auth/login.html", context)

* In (templates/auht/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/testapp/>

## Validation in email

What is a Django widget?

A widget is Django's representation of an HTML input element. The widget handles the rendering of the HTML, and the extraction of data from a GET/POST dictionary that corresponds to the widget. The HTML generated by the built-in widgets uses HTML5 syntax, targeting <!

Is valid form Django?

Django forms submit only if it contains CSRF tokens. It uses a clean and easy approach to validate data. The is\_valid() method is used to perform validation for each field of the form, it is defined in Django Form class. It returns True if data is valid and place all data into a cleaned\_data attribute.

* (Testapp/forms.py)>

from django import forms

class ContactForm(forms.Form):

fullname = forms.CharField(

widget=forms.TextInput(

attrs={

"id":"form\_full\_name",

"placeholder":"Your full name"

}

)

)

email = forms.EmailField()

def clean\_email(self):

email = self.cleaned\_data.get("email")

if not "gmail.com" in email:

raise forms.ValidationError("Email has to be gmail.com")

return email

* (testapp/views.py)>

def login\_page(request):

contact\_form = ContactForm(request.POST or None)

if contact\_form.is\_valid():

print(contact\_form.cleaned\_data)

# if request.method == "POST":

# print(request.POST)

# print(request.POST.get('fullname'))

context = {

'form':contact\_form,

}

return render(request, "auth/login.html", context)

* (testapp/templates/auth/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/testapp/>
* Check email validation with and without ‘gmail.com’

## Login page

* (testapp/forms.py)>

class LoginForm(forms.Form):

username = forms.CharField()

password = forms.CharField(widget=forms.PasswordInput())

* (testapp/views.py)>

from .forms import ContactForm, LoginForm

def login\_page(request):

login\_form = LoginForm(request.POST or None)

print(request.user.is\_authenticated)

if login\_form.is\_valid():

print(login\_form.cleaned\_data)

context = {

'form':login\_form,

}

return render(request, "auth/login.html", context)

* Create (templates/auth/login.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* In browser <http://127.0.0.1:8000/login/>
* In (views.py)>

from django.contrib.auth import authenticate, login

from django.shortcuts import render, redirect

def login\_page(request):

login\_form = LoginForm(request.POST or None)

context = {

'form':login\_form,

}

print("\*" \* 10)

print(request.user.is\_authenticated)

if login\_form.is\_valid():

print(login\_form.cleaned\_data)

username = login\_form.cleaned\_data.get("username")

password = login\_form.cleaned\_data.get("password")

user = authenticate(request, username=username, password=password)

print(user)

if user is not None:

login(request, user)

#context['form'] = LoginForm()

return redirect("/login")

else:

print("Error")

return render(request, "auth/login.html", context)

* In cmd
  + python manage.py createsuperuser
* in browser <http://127.0.0.1:8000/login/>

## Check authenticated

What is authenticate in Django?

The Django authentication system handles both authentication and authorization. Briefly, authentication verifies a user is who they claim to be, and authorization determines what an authenticated user is allowed to do. Here the term authentication is used to refer to both tasks.

Is Django authentication secure?

Django is as secure as any web framework can be. It provides tools and doc to prevent common mistakes causing security problems (csrf, xss, etc.) However, a tool in itself cannot be "secure". The whole platform security depends on the proper use of the tools you choose, and thus is more a matter of developer skills.

How do I know if a user is authenticated Django?

* in (urls.py)>

path('home/',views.home\_page),

* (views.py>

def home\_page(request):

# if not request.user.is\_authenticated():

# return Login

print(request.session.get("first\_name", "unknown"))

print(request.session.get("user", "unknown"))

context = {}

if request.user.is\_authenticated:

context["premium\_content"] = "User logged in"

return render(request, "testapp/home.html",context)

* (testapp/templates/testapp/home.html)>

<h1>Cart</h1>

{% if request.user.is\_authenticated%}

<h1>Premium</h1>

{{ premium\_content }}

{% endif %}

* In browser <http://127.0.0.1:8000/home/>

## Register Form

What is the purpose of a registration form?

Your registration form actually serves three purposes. It is: A way for people to sign up for your event. One of the first impressions people get about your event brand.

How do I create a registration form?

* (forms.py)>

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

password = forms.CharField(widget=forms.PasswordInput())

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

* In (views.py)>

from .forms import ContactForm, LoginForm, RegisterForm

def register\_page(request):

register\_form = RegisterForm(request.POST or None)

context = {

'form':register\_form,

}

if register\_form.is\_valid():

print(register\_form.cleaned\_data)

return render(request, "auth/register.html", context)

* Create file (testapp/templates/auth/register.html)>

<form method="POST">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

* (urls.py)>

path('register/',views.register\_page),

* (forms.py)>

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

password = forms.CharField(widget=forms.PasswordInput())

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

def clean(self):

data = self.cleaned\_data

password = self.cleaned\_data.get('password')

password2 = self.cleaned\_data.get('password2')

if password2 != password:

raise forms.ValidationError("password must match")

return data

* (views.py)>

from django.contrib.auth import authenticate, login, get\_user\_model

User = get\_user\_model()

def register\_page(request):

register\_form = RegisterForm(request.POST or None)

context = {

'form':register\_form,

}

if register\_form.is\_valid():

print(register\_form.cleaned\_data)

username = register\_form.cleaned\_data.get("username")

email = register\_form.cleaned\_data.get("email")

password = register\_form.cleaned\_data.get("password")

new\_user = User.objects.create\_user(username, email, password)

print(new\_user)

return render(request, "auth/register.html", context)

* (forms.py)>

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

class RegisterForm(forms.Form):

username = forms.CharField()

email = forms.EmailField()

password = forms.CharField(widget=forms.PasswordInput())

password2 = forms.CharField(label='Confirm password' , widget=forms.PasswordInput())

def clean\_username(self):

username = self.cleaned\_data.get("username")

qs = User.objects.filter(username = username)

if qs.exists():

raise forms.ValidationError("Username is taken")

return username

def clean(self):

data = self.cleaned\_data

password = self.cleaned\_data.get('password')

password2 = self.cleaned\_data.get('password2')

if password2 != password:

raise forms.ValidationError("password must match")

return data

# Session

what is session in Django?

Sessions are the mechanism used by Django (and most of the Internet) for keeping track of the "state" between the site and a particular browser. Sessions allow you to store arbitrary data per browser, and have this data available to the site whenever the browser connects.

How does Django session work?

Django provides a session framework that lets you store and retrieve data on a per-site-visitor basis. Django abstracts the process of sending and receiving cookies, by placing a session ID cookie on the client side, and storing all the related data on the server side. So the data itself is not stored client side.

How do you check session is set or not in Django?

get('orderId', None) will return None and the if not request. session. get('orderId', None) will direct you to populate the session, which (probably) is what you wanted. this will give you orderId from the session if exists, otherwise it will populate the session with ts and return you that.

How does Django store data in a session?

* Create a new project

1. django-admin startproject session
2. cd session
3. python manage.py startapp testapp

* (testapp) views.py >

from django.shortcuts import render

# Create your views here.

def cart\_home(request):

print(request.session)

print(dir(request.session))

return render(request, "testapp/home.html",{})

* Urls.py

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

path('testapp/',views.cart\_home),

]

* Settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'testapp',

]

* Create folder:- testapp/templates/home.html

<h1>Cart</h1>

## Session key

* Testapp/views.py

from django.shortcuts import render

# Create your views here.

def cart\_home(request):

# print(request.session)

# print(dir(request.session))

key = request.session.session\_key

print(key)

return render(request, "testapp/home.html",{})

* In cmd
  + python manage.py migrate
  + python manage.py runserver
* in (testapp/views.py)>

def cart\_home(request):

# print(request.session)

# print(dir(request.session))

# key = request.session.session\_key

# print(key)

request.session['first\_name'] = "Abhijith"

return render(request, "testapp/home.html",{})

def home\_page(request):

print(request.session.get("first\_name", "unknown"))

return render(request, "testapp/home.html",{})

* (urls.py)>

urlpatterns = [

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

path('testapp/',views.cart\_home),

path('testapp1/',views.home\_page),

]

* Refresh browser >
* <http://127.0.0.1:8000/testapp/>
* <http://127.0.0.1:8000/testapp1/>

# Stream video to webpage

What is video streaming?

Streaming media is multimedia that is constantly received by and presented to an end-user while being delivered by a provider. The verb to stream refers to the process of delivering or obtaining media in this manner. Streaming refers to the delivery method of the medium, rather than the medium itself.

What is StreamingHttpResponse in Django?

The StreamingHttpResponse class is used to stream a response from Django to the browser. You might want to do this if generating the response takes too long or uses too much memory. For instance, it's useful for generating large CSV files. Performance considerations. Django is designed for short-lived requests.

* Views.py

from django.shortcuts import render

from django.views.decorators import gzip

from django.shortcuts import render

from django.http import HttpResponse,StreamingHttpResponse

import cv2

import time

class VideoCamera(object):

def \_\_init\_\_(self):

self.video = cv2.VideoCapture(0)

def \_\_del\_\_(self):

self.video.release()

def get\_frame(self):

ret,image = self.video.read()

ret,jpeg = cv2.imencode('.jpg',image)

return jpeg.tobytes()

def gen(camera):

while True:

frame = camera.get\_frame()

yield(b'--frame\r\n'

b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@gzip.gzip\_page

def index(request):

try:

return StreamingHttpResponse(gen(VideoCamera()),content\_type="multipart/x-mixed-replace;boundary=frame")

except:

print("aborted")

* urls.py

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

path('index/', views.index, name='index'),

]

# Video streaming using Javascript

* index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta content="stuff, to, help, search, engines, not" name="keywords">

<meta content="What this page is about." name="description">

<meta content="Display Webcam Stream" name="title">

<title>Display Webcam Stream</title>

<style>

#container {

margin: 0px auto;

width: 500px;

height: 375px;

border: 10px #333 solid;

}

#videoElement {

width: 500px;

height: 375px;

background-color: #666;

}

</style>

</head>

<body>

<div id="container">

<video autoplay="true" id="videoElement">

</video>

</div>

<script>

var video = document.querySelector("#videoElement");

if (navigator.mediaDevices.getUserMedia) {

navigator.mediaDevices.getUserMedia({

video: true

})

.then(function(stream) {

video.srcObject = stream;

// myJson = JSON.stringify(stream)

})

.catch(function(err0r) {

console.log("Something went wrong!");

});

}

console.log(video)

</script>

</body>

</html>

## Stream video Xframe option error

from django.http import HttpResponse

from django.views.decorators.clickjacking import xframe\_options\_exempt

@xframe\_options\_exempt

def ok\_to\_load\_in\_a\_frame(request):

    return HttpResponse("This page is safe to load in a frame on any site.")

* views.py

def index(request):

return render(request, 'testapp/index.html', {})

# Upload Video

* create ‘media’ folder in parent directory (the directory which contains manage.py file)
* settings.py > bottom of page

MEDIA\_ROOT= os.path.join(BASE\_DIR, 'media/')

MEDIA\_URL= "/media/"

* model.py

from django.db import models

# Create your models here.

class Album(models.Model):

artist = models.CharField(max\_length=250)

album\_title = models.CharField(max\_length=500)

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

album\_logo = models.FileField()

def \_\_str\_\_(self):

return self.album\_title + " - " + self.artist

* admin.py

from .models import Album

admin.site.register(Album)

* setting.py

INSTALLED\_APPS = [

'upload.apps.UploadConfig',

* setting.py – bottom of page

STATIC\_URL = '/static/'

MEDIA\_ROOT= os.path.join(BASE\_DIR, 'media/')

MEDIA\_URL= "/media/"

* urls.py

from django.contrib import admin

from django.urls import path

from django.conf import settings

from django.conf.urls.static import static

from upload import views

urlpatterns = [

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

path('index/', views.IndexCBV.as\_view(), name='index'),

]

if settings.DEBUG:

urlpatterns += static(settings.STATIC\_URL, document\_root = settings.STATIC\_ROOT)

urlpatterns += static(settings.MEDIA\_URL, document\_root = settings.MEDIA\_ROOT)

* forms.py >

from django import forms

from .models import Album

class AlbumForm(forms.ModelForm):

class Meta:

model = Album

fields = ['artist', 'album\_title', 'genre', 'album\_logo']

* view.py

from django.shortcuts import render

from django.views.generic import View

from .forms import AlbumForm

from .models import Album as AlbumDB

# Create your views here.

class IndexCBV(View):

def get(self, request, \*args, \*\*kwargs):

uploadForm = AlbumForm()

all\_albums = AlbumDB.objects.all()

context = {

'form' : uploadForm,

'all\_albums' : all\_albums,

}

return render(request, "index.html", context)

def post(self, request, \*args, \*\*kwargs):

print(request.POST, request.FILES)

artist = request.POST['artist']

album\_title = request.POST['album\_title']

genre = request.POST['genre']

album\_logo = request.FILES['album\_logo']

formdata = {

'artist' : artist,

'album\_title' : album\_title,

'genre' : genre,

'album\_logo' : album\_logo,

}

form = AlbumForm(request.POST, request.FILES)

if form.is\_valid():

form.save(commit=True)

result = 'Resource created successfully'

if form.errors:

result = form.errors

context = {

'result' : result

}

return render(request, "index.html", context )

* (templates/index.html)

{% for album in all\_albums %}

<a href="#" >

<img src="{{ album.album\_logo.url }}" >

</a>

{{ album.album\_title }}

{% endfor %}

<form method="POST" enctype="multipart/form-data">

{% csrf\_token %}

{{ form }}

<button type="submit" >Submit</button>

</form>

{% if result %}

Result: {{result}}

{% endif %}

* Forms.py

from django import forms

from .models import Album

class AlbumForm(forms.ModelForm):

class Meta:

model = Album

fields = ['artist', 'album\_title', 'genre', 'album\_logo']

# Enabling https

What is HTTPS?

HTTPS (Hypertext Transfer Protocol Secure) is a secure version of the HTTP protocol that uses the SSL/TLS protocol for encryption and authentication. HTTPS is specified by RFC 2818 (May 2000) and uses port 443 by default instead of HTTP's port 80.

The HTTPS protocol makes it possible for website users to transmit sensitive data such as credit card numbers, banking information, and login credentials securely over the internet. For this reason, HTTPS is especially important for securing online activities such as shopping, banking, and remote work. However, HTTPS is quickly becoming the standard protocol for all websites, whether or not they exchange sensitive data with users.

Why use HTTPS?

There are multiple good reasons to use HTTPS on your website, and to insist on HTTPS when browsing, shopping, and working on the web as a user:

Integrity and Authentication: Through encryption and authentication, HTTPS protects the integrity of communication between a website and a user’s browsers. Your users will know that the data sent from your web server has not been intercepted and/or altered by a third party in transit. And, if you’ve made the extra investment in EV or OV certificates, they will also be able to tell that the information really came from your business or organization.

Privacy: Of course no one wants intruders scooping up their credit card numbers and passwords while they shop or bank online, and HTTPS is great for preventing that. But would you really want everything else you see and do on the web to be an open book for anyone who feels like snooping (including governments, employers, or someone building a profile to de-anonymize your online activities)? HTTPS plays an important role here too.

User Experience: Recent changes to browser UI have resulted in HTTP sites being flagged as insecure. Do you want your customers’ browsers to tell them that your website is “Not Secure” or show them a crossed-out lock when they visit it? Of course not!

Compatibility: Current browser changes are pushing HTTP ever closer to incompatibility. Mozilla Firefox recently announced an optional HTTPS-only mode, while Google Chrome is steadily moving to block mixed content (HTTP resources linked to HTTPS pages). When viewed together with browser warnings of “insecurity” for HTTP websites, it’s easy to see that the writing is on the wall for HTTP. In 2020, all current major browsers and mobile devices support HTTPS, so you won’t lose users by switching from HTTP.

SEO: Search engines (including Google) use HTTPS as a ranking signal when generating search results. Therefore, website owners can get an easy SEO boost just by configuring their web servers to use HTTPS rather than HTTP.

## Method 1

just install

sudo pip install django-sslserver

include sslserver in installed aps

INSTALLED\_APPS = (...

"sslserver",

...

)

now you can run

python manage.py runsslserver 0.0.0.0:8888

## method 2

Please go to this url

<https://gist.github.com/claudiosanches/7012524>

* stunnel4 dev\_https & python3 manage.py runserver 0.0.0.0:8444& HTTPS=1 python3 manage.py runserver 0.0.0.0:8001

# Publish your site in a network or public domain – 0.0.0.0:80

How do I access Django from another computer or mobile?

To make your development server viewable to other machines on the network, use its own IP address (e.g. 192.168. 2.1) or 0.0. 0.0 or :: (with IPv6 enabled). Run the application with IP address then access it in other machines.

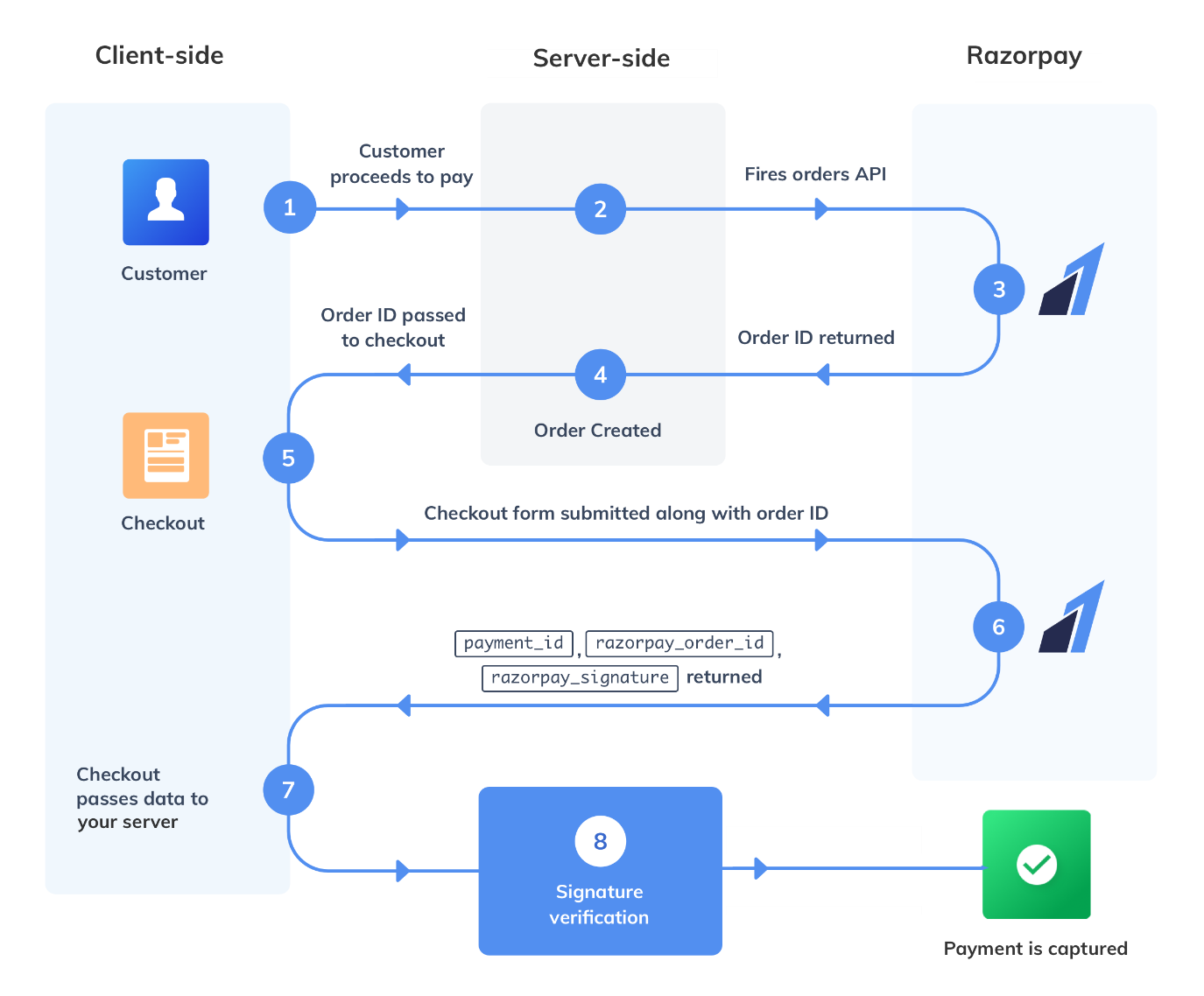
$ python manage.py runserver 0.0.0.0:2424

# Django payment gateway – razopay

## Integrating Razorpay with django

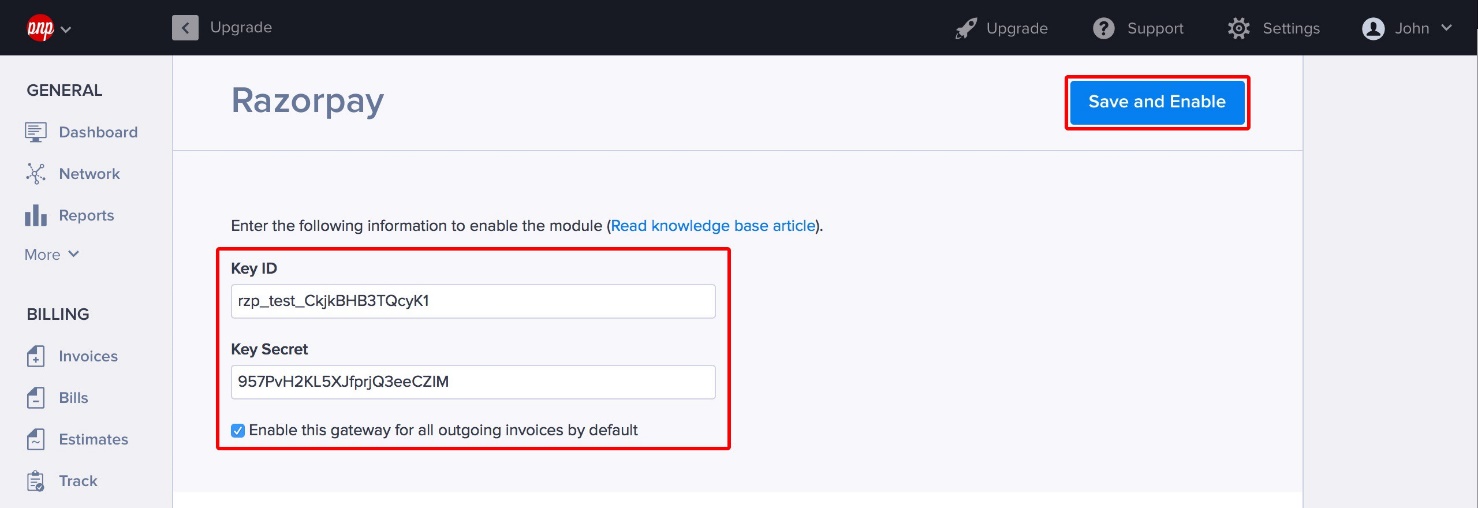


Before Integrating Razorpay payment gateway, let’s first have a look at how **Razorpay Payment flow works**.



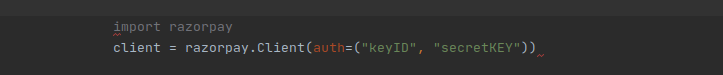
So here’s a story before we dive into the Integration part as this will help to understand what exactly happens at the backend.  
So let’s understand this, you have a Business(Website) and Clients(users). You need to take payments online from your Clients but you are wondering if you could hire someone who could collect the payments on your behalf and that someone are the **Payment Gateways**(eg. Paytm, Razorpay, Paypal etc.). So there are rules and expectation from Payment Gateways which you should follow.

### ***#1. Open razorpay Dashboard and collect the KeyId and SecretKey.***



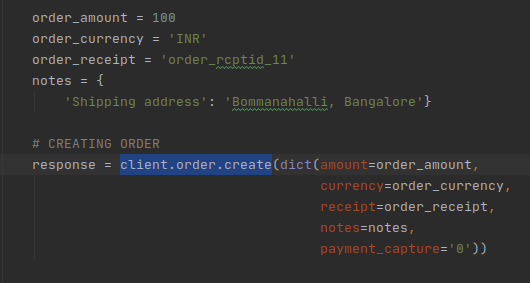
### ***#2. Create Client Instance at Server-side for communicating with Razorpay.***

$ pip install razorpay

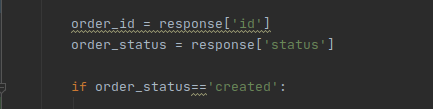


This ‘client’ Object will be used to communicate with razorpay.

### ***#3. Create order by using ‘client.order.create’ method.***

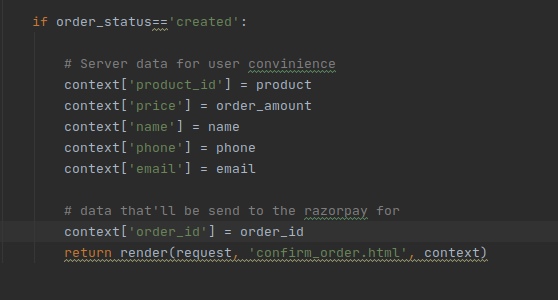


### ***#4. Store the ID and Status returned by the razorpay***



Make sure that status returned by the razorpay is — created. This means creating order was successful and it is being acknowledged by razorpay.

### ***#5. Pass the necessary parameters and order\_id to html page.***



### ***#6. Copy the javascript from here!!! and paste it to the html page where you want the payment to be done!!!***

<button id="rzp-button1">Pay</button>

<script src="https://checkout.razorpay.com/v1/checkout.js"></script>

<script>

var options = {

    "key": "YOUR\_KEY\_ID", // Enter the Key ID generated from the Dashboard

    "amount": "50000", // Amount is in currency subunits. Default currency is INR. Hence, 50000 refers to 50000 paise

    "currency": "INR",

    "name": "Acme Corp",

    "description": "Test Transaction",

    "image": "https://example.com/your\_logo",

    "order\_id": "order\_9A33XWu170gUtm", //This is a sample Order ID. Pass the `id` obtained in the response of Step 1

    "handler": function (response){

        alert(response.razorpay\_payment\_id);

        alert(response.razorpay\_order\_id);

        alert(response.razorpay\_signature)

    },

    "prefill": {

        "name": "Gaurav Kumar",

        "email": "gaurav.kumar@example.com",

        "contact": "9999999999"

    },

    "notes": {

        "address": "Razorpay Corporate Office"

    },

    "theme": {

        "color": "#3399cc"

    }

};

var rzp1 = new Razorpay(options);

rzp1.on('payment.failed', function (response){

        alert(response.error.code);

        alert(response.error.description);

        alert(response.error.source);

        alert(response.error.step);

        alert(response.error.reason);

        alert(response.error.metadata.order\_id);

        alert(response.error.metadata.payment\_id);

});

document.getElementById('rzp-button1').onclick = function(e){

    rzp1.open();

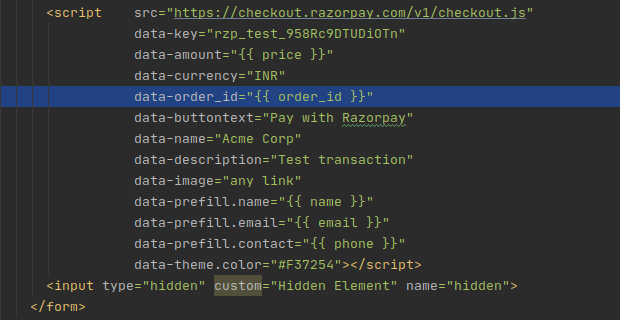
    e.preventDefault();

}

</script>

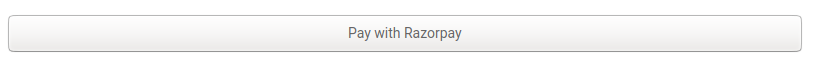
Image for post

**NOTE**: ‘{% url ‘**payment\_status**’ %}’ is the **redirect url** where razorpay will redirect the user after the process.



Now pay attention to the **data-order\_id**, this is the same parameter which we received from the razorpay while we were creating the order.

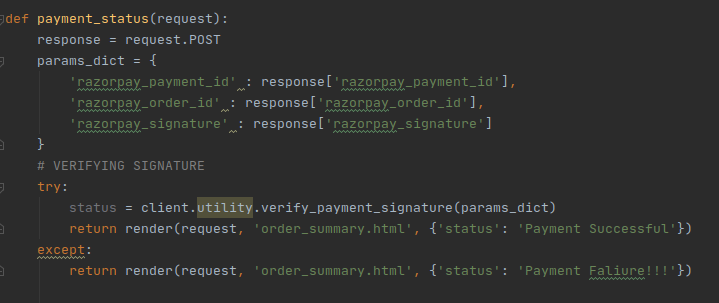
After this the button on page will look like:



Now, when User clicks on this button!!!

When this is all Done! Its not Over yet. There is one more thing we need to take care of and that is

### ***#7. Verifying the signature!!!***



When the razorpay makes a successful payment it redirects to the url which was mentioned in the form action. Then we need to extract three keywords which are ‘**razorpay\_payment\_id**’, ‘**razorpay\_order\_id**’ and **razorpay\_signature** and then pass it to the '**client.utility.verify\_payment\_signature()**'.

***Note:****if the payment was successful then it would return****none,****else it would throw an error.*

# Part 2

# REST API

What is REST API in Django?

Django REST framework is a powerful and flexible toolkit for building Web APIs. The Web browsable API is a huge usability win for your developers. ... Authentication policies including packages for OAuth1a and OAuth2. Serialization that supports both ORM and non-ORM data sources.

Is Django GOOD FOR REST API?

Django is an incredibly robust web application solution that isn't suited to building REST APIs alone.

What is the use of REST API in Django?

A typical Django application that uses React as a front end. It needs an API to allow React to consume data from the database. For example, in web development, many applications rely on REST APIs to allow the front end to talk to the back end.

Does Instagram still use Django?

Instagram, the world's biggest online photo-sharing app, uses Python on its backend. According to Instagram's engineering team, Instagram currently features the world's largest deployment of the Django web framework, which is written entirely in Python.

When should I use RESTful API?

REST allows for building APIs with any method, including HTTP, URLs, and JSON. SOAP only uses XML for sending data. REST is considered a simpler, more efficient alternative to SOAP because it requires writing less code to complete tasks and follows a less rigid structure and logic than SOAP.

What is HttpResponse in Django?

HttpResponse (source code) provides an inbound HTTP request to a Django web application with a text response. This class is most frequently used as a return object from a Django view.

What is the difference between render and HttpResponse?

render is basically a simple wrapper around a HttpResponse which renders a template, though as said in the previous answer you can use HttpResponse to return others things as well in the response, not just rendering templates.

## Create new project

* Cmd activate environment

1. *(python36) C:\Users\abhijith.m\0 zerone\Django>django-admin startproject withoutrest*
2. *(python36) C:\Users\abhijith.m\0 zerone\Django>cd withoutrest*
3. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>python manage.py startapp testapp*

* (testapp/views.py)>

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def emp\_data\_view(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

resp = 'Employee Number:{}Employee Name:{}\

Employee Salary:{}Employee Address:{}'.format(emp\_data['eno'],

emp\_data['ename'],

emp\_data['esal'],

emp\_data['eaddr'])

return HttpResponse(resp)

* (withoutrest/urls.py)>

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

path('api/', views.emp\_data\_view),

]

* Runserver
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>python manage.py runserver*
  + <http://127.0.0.1:8000/api>
* (testapp/views.py)>

import json

def emp\_data\_jsonview(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

resp = json.dumps(emp\_data)

return HttpResponse(resp, content\_type='application/json')

* (withoutrest/urls.py)>

path('apijs/', views.emp\_data\_jsonview),

* <http://127.0.0.1:8000/apijs/>

## Cmd http client

* + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>pip install httpie*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>http* [*http://127.0.0.1:8000/apijs/*](http://127.0.0.1:8000/apijs/)

HTTP/1.1 200 OK

Content-Length: 64

Content-Type: application/json

Date: Thu, 18 Jul 2019 12:42:52 GMT

Server: WSGIServer/0.2 CPython/3.6.8

X-Frame-Options: SAMEORIGIN

{

"eaddr": "Mumbai",

"ename": "Sunny",

"eno": 100,

"esal": 10000

}

* Json response without converting dict to json
* (testapp/views.py)> add

from django.http import JsonResponse

def emp\_data\_jsonview2(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

return JsonResponse(emp\_data)

* (withoutrest/urls.py)> add

path('apijs2/', views.emp\_data\_jsonview2),

* Create file ‘test.py’ in any other location other than project directory (anywhere in your PC)

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'apijs'

resp = requests.get(BASE\_URL+ENDPOINT)

print(resp.json())

* Run test.py file
* Open cmd in test.py directory

$ python test.py

## Generic view

What is post and get method?

By design, the POST request method requests that a web server accepts the data enclosed in the body of the request message, most likely for storing it. It is often used when uploading a file or when submitting a completed web form. In contrast, the HTTP GET request method retrieves information from the server.

What is put method in HTTP?

In general the HTTP PUT method replaces the resource at the current URL with the resource contained within the request. PUT is used to both create and update the state of a resource on the server.

What is Delete method?

The HTTP DELETE method is used to delete a resource from the server. Unlike GET and HEAD requests, the DELETE requests may change the server state. Sending a message body on a DELETE request might cause some servers to reject the request.

* (testapp/views.py)>

from django.views.generic import View

class jsonCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp\_data={

'eno':100,

'ename': 'Sunny',

'esal':1000,

'eaddr':'Mumbai',

}

return JsonResponse(emp\_data)

* (withoutrest/urls.py)> add

path('apijscbv/', views.jsonCBV.as\_view()),

* In browser <http://127.0.0.1:8000/apijscbv/>
* (testapp/views.py)> update

from django.views.generic import View

class jsonCBV(View):

def get(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from get method'})

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return HttpResponse(json\_data, content\_type='application/json')

def put(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from put method'})

return HttpResponse(json\_data, content\_type='application/json')

def delete(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from delete method'})

return HttpResponse(json\_data, content\_type='application/json')

* Test.py

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'apijscbv'

resp = requests.post(BASE\_URL+ENDPOINT)

print(resp.json())

* It should return an error msg. because the csrf token is not defined.
* The middleware setting reject this request
* (withoutrest/settings.py)> inside ‘MIDDLEWARE’ >
* Comment the following line

# 'django.middleware.csrf.CsrfViewMiddleware',

* Test.py request ‘’resp = requests.delete(BASE\_URL+ENDPOINT)’’

### Basics of arguments

Here we going to do some python basics

What is an argument in python give an example?

Arguments in Functions

The argument is a value that is passed to the function when it's called. In other words on the calling side, it is an argument and on the function side it is a parameter. Let see how Python Args works - Step 1) Arguments are declared in the function definition.

What are the types of arguments in Python?

5 Types of Arguments in Python Function Definition:

1. default arguments.
2. keyword arguments.
3. positional arguments.
4. arbitrary positional arguments.
5. arbitrary keyword arguments.

Give an example of argument?

def f1(\*\*kwargs):

print(kwargs)

f1(name='abhi', rollno=100, mark=90)

{'name': 'abhi', 'rollno': 100, 'mark': 90}

def f1(\*args):

print(args)

f1()

f1(90)

f1(90,20,30)

()

(90,)

(90, 20, 30)

### Mixin

What is Mixins in Django REST framework?

The generics and mixin modules are indeed different, yet they are inter-related. DRF separates ReSTful API / HTTP verb behaviour from Django model operations and organises a set of abstract/base classes for each. A ViewSet class is simply a type of class-based View, that does not provide any method handlers such as .

* (testapp/)> create file mixins.py

from django.http import HttpResponse

class HttpResponseMixin(object):

def render\_to\_http\_response(self, json\_data):

#1000 lines of code

return HttpResponse(json\_data, content\_type='application/json')

* (testapp/views.py)> update

from django.views.generic import View

from testapp.mixins import HttpResponseMixin

class jsonCBV(HttpResponseMixin, View):

def get(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from get method'})

return self.render\_to\_http\_response(json\_data)

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return self.render\_to\_http\_response(json\_data)

def put(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from put method'})

return self.render\_to\_http\_response(json\_data)

def delete(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from delete method'})

return self.render\_to\_http\_response(json\_data)

## CRUD without REST framework

What is SQL CRUD?

CRUD represents an acronym for the database operations Create, Read, Update, and Delete. The communication between two layers could be in the form of ad hoc SQL statements such as INSERT, SELECT, UPDATE, and DELETE.

What is crud REST API?

CRUD stands for Create, Read, Update, and Delete. But put more simply, in regards to its use in RESTful APIs, CRUD is the standardized use of HTTP Action Verbs. ... Keep in mind there are several different HTTP Action Verbs available, and it's easy to want to incorporate these new verbs and make your API new and different.

* In cmd

1. *(python36) C:\Users\abhijith.m\0 zerone\Django>django-admin startproject withoutrestm*
2. *(python36) C:\Users\abhijith.m\0 zerone\Django>cd withoutrestm*
3. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py startapp testapp*

* (withoutrestm/settings.py)> inside ‘INSTALLED\_APPS’ add

'testapp',

* (testapp/models.py)>

class Employee(models.Model):

id = models.AutoField(primary\_key=True)

eno = models.IntegerField()

ename = models.CharField(max\_length=64)

esal = models.FloatField()

eaddr = models.CharField(max\_length=64)

* (testapp/admin.py).

from testapp.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['id','eno', 'ename', 'esal', 'eaddr']

admin.site.register(Employee, EmployeeAdmin)

* In cmd
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py makemigrations*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py migrate*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py createsuperuser*

*Username (leave blank to use 'abhijith.m'): admin*

*Email address: admin@example.com*

*Password:*

*Password (again):*

*Superuser created successfully.*

* Start server
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py runserver*

### Read

#### Based on id

* (testapp/views.py)>

from django.shortcuts import render

from django.views.generic import View

from testapp.models import Employee

import json

from django.http import HttpResponse

# Create your views here.

class EmployeeDetailCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.get(id=1)

emp\_data={

'eno': emp.eno,

'ename': emp.ename,

'esal': emp.esal,

'eaddr': emp.eaddr,

}

json\_data = json.dumps(emp\_data)

return HttpResponse(json\_data, content\_type='application/json')

* (withoutrestm/urls.py)> add

from testapp import views

* (withoutrestm/urls.py)> inside ‘urlpatterns’

path('api/', views.EmployeeDetailCBV.as\_view())

* In browser <http://127.0.0.1:8000/api/>
* Provide id explicitly
* (testapp/views.py)> update

class EmployeeDetailCBV(View):

def get(self, request, id, \*args, \*\*kwargs):

emp=Employee.objects.get(id=id)

* (withoutrestm/urls)> update

path('api/<int:id>', views.EmployeeDetailCBV.as\_view())

#### Serializer

What is serializer Django?

One family of classes DRF has is Serializers. They're used to convert the data sent in a HTTP request to a Django object and a Django object to a valid response data. It looks a lot like a Django Form, but it is also concerned in defining how the data will be returned to the user.

What does a serializer do?

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

Why do we use serializer in Django?

Django's serialization framework provides a mechanism for “translating” Django models into other formats. Usually these other formats will be text-based and used for sending Django data over a wire, but it's possible for a serializer to handle any format (text-based or not).

How would you create a serializer in Django?

To create a basic serializer one needs to import serializers class from rest\_framework and define fields for a serializer just like creating a form or model in Django. This way one can declare serializer for any particular entity or object based on fields required.

* (testapp/views.py)> add

from django.core.serializers import serialize

* (testapp/views.py)> update

# Create your views here.

class EmployeeDetailCBV(View):

def get(self, request, id, \*args, \*\*kwargs):

emp=Employee.objects.get(id=id)

json\_data = serialize('json', [emp,], fields=('eno', 'ename','eaddr'))

return HttpResponse(json\_data, content\_type='application/json')

* Get all
* (testapp/views.py)> add

class EmployeeListCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.all()

json\_data = serialize('json', emp)

return HttpResponse(json\_data, content\_type='application/json')

* (withoutrestm/urls)> add inside ‘urlpatterns’

path('api/', views.EmployeeListCBV.as\_view())

* Run test.py

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'api/'

def get\_resource(id):

resp = requests.get(BASE\_URL+ENDPOINT+id+'/')

print(resp.status\_code)

print(resp.json())

# id = input("Enter id:")

# get\_resource(id)

def get\_all():

resp = requests.get(BASE\_URL+ENDPOINT)

print(resp.status\_code)

print(resp.json())

get\_all()

* To exclude ‘’model and pk‘’ from out put
* (testapp/views.py)> add

class EmployeeListCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.all()

json\_data = serialize('json', emp)

pdict = json.loads(json\_data)

final\_list = []

for obj in pdict:

emp\_data = obj['fields']

final\_list.append(emp\_data)

json\_data = json.dumps(final\_list)

return HttpResponse(json\_data, content\_type='application/json')

* (testapp)> create file mixins.py

from django.core.serializers import serialize

import json

class SerializerMixin(object):

def serialize(self, qs):

json\_data = serialize('json', qs)

p\_data = json.loads(json\_data)

final\_list = []

for obj in p\_data:

emp\_data = obj['fields']

final\_list.append(emp\_data)

json\_data = json.dumps(final\_list)

return json\_data

* (testapp/views.py)> update

from testapp.mixins import SerializerMixin

# Create your views here.

class EmployeeDetailCBV(SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

qs=Employee.objects.get(id=id)

json\_data = self.serialize([qs,])

return HttpResponse(json\_data, content\_type='application/json')

# Create your views here.

class EmployeeListCBV(SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

#### Exception: Avoid DoesNotExist error

* (testapp/views.py)> update

class EmployeeDetailCBV(SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

try:

qs=Employee.objects.get(id=id)

except :

json\_data = json.dumps({'Error': "The requested resource is not available"})

else:

json\_data = self.serialize([qs,])

return HttpResponse(json\_data, content\_type='application/json')

* Add httpResponse in mixin
* (testapp/mixin.py)> add

from django.http import HttpResponse

class HttpresponseMixin(object):

def render\_to\_http\_response(self, json\_data, status=200):

return HttpResponse(json\_data, content\_type='application/json', status=status)

* (testapp/views.py)> update

class EmployeeDetailCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

try:

qs=Employee.objects.get(id=id)

except :

json\_data = json.dumps({'Error': "The requested resource is not available"})

return self.render\_to\_http\_response(json\_data, status=404)

else:

json\_data = self.serialize([qs,])

return self.render\_to\_http\_response(json\_data)

#### Save or Display all data in table by console code

* In cmd
  1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee*

**indented format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --indent 4*

**indented format with json**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format json --indent 4*

**XML format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format xml --indent 4*

**Save file to emp.xml**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format xml >emp.xml --indent 4*

**Save in yaml format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format yaml >emp.yaml --indent 4*

### Create

* (testapp/views.py)> update

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

self.render\_to\_http\_response(json\_data)

* Test.py > add

def create\_resource():

new\_emp = {

'eno':500,

'ename':'Shiva',

'esal': 5000,

'eaddr': 'Chennai',

}

new\_emp = json.dumps(new\_emp)

resp = requests.post(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

create\_resource()

### csrf token disabling

1. Class Level

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return self.render\_to\_http\_response(json\_data)

1. Project Level

* Run test.py
* **To check client was send a valid json data or not**
* (testapp)> create new file ‘utils.py’

import json

def is\_json(data):

try:

p\_data = json.loads(data)

valid = True

except ValueError:

valid = False

return valid

* (testapp/views.py)> add

from testapp.utils import is\_json

* (testapp/views.py)> update

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return self.render\_to\_http\_response(json\_data)

def post(self, request, \*args, \*\*kwargs):

data = request.body

valid\_jason = is\_json(data)

if not valid\_jason:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

json\_data = json.dumps({'msg':'You provided valid json data'})

return self.render\_to\_http\_response(json\_data, status=400)

### store database

* (testapp)> create new file forms.py

from django import forms

from testapp.models import Employee

class EmployeeForm(forms.ModelForm):

def clean\_esal(self):

inputsal = self.cleaned\_data['esal']

if inputsal<5000:

raise forms.ValidationError('The minimum salary should be 5000')

return inputsal

class Meta:

model = Employee

fields = '\_\_all\_\_'

* (testapp/views.py)> update inside ‘class EmployeeListCBV’

def post(self, request, \*args, \*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

empdata = json.loads(data)

form = EmployeeForm(empdata)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource created successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

* test.py

def create\_resource():

new\_emp = {

'eno':500,

'ename':'Shiva',

'esal': '50000',

'eaddr': 'Chennai',

}

new\_emp = json.dumps(new\_emp)

resp = requests.post(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

create\_resource()

* Run test.py

## Update

* (testapp/views.py)> add

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeDetailCBV(HttpresponseMixin, SerializerMixin, View):

* (testapp/views.py)> add inside ‘class EmployeeDetailCBV’

def get\_object\_by\_id(self,id):

try:

emp = Employee.objects.get(id=id)

except Employee.DoesNotExist:

emp = None

return emp

def put(self,request, id, \*args, \*\*kwargs):

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found for updation"})

return self.render\_to\_http\_response(json\_data, status=404)

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

provided\_data = json.loads(data)

original\_data = {

'eno':emp.eno,

'ename':emp.ename,

'esal':emp.esal,

'eaddr': emp.eaddr

}

original\_data.update(provided\_data)

form = EmployeeForm(original\_data, instance=emp)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource Updated successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

* Test.py > add and run

def update\_resource(id):

new\_emp = {

'esal': '70000',

'eaddr': 'Delhi',

}

new\_emp = json.dumps(new\_emp)

resp = requests.put(BASE\_URL+ENDPOINT+str(id)+'/', data=new\_emp)

print(resp.status\_code)

print(resp.json())

update\_resource(6)

## Delete

* (testapp/views.py)> add

def delete(self, request,id, \*args, \*\*kwargs):

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found for deletion"})

return self.render\_to\_http\_response(json\_data, status=404)

status,deleted\_item = emp.delete()

if status == 1:

json\_data = json.dumps({'Msg': "Resource deleted successfully"})

return self.render\_to\_http\_response(json\_data)

json\_data = json.dumps({'Error': "Unable to delete. Please try again"})

return self.render\_to\_http\_response(json\_data, status=404)

* Test.py > add and run

def delete\_resource(id):

resp = requests.delete(BASE\_URL+ENDPOINT+str(id)+'/')

print(resp.status\_code)

print(resp.json())

delete\_resource(6)

## By using single baseurl

* (testapp/views.py)> add

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeCRUDCBV(HttpresponseMixin, SerializerMixin, View):

def get\_object\_by\_id(self,id):

try:

emp = Employee.objects.get(id=id)

except Employee.DoesNotExist:

emp = None

return emp

def get(self, request, \*args,\*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = self.serialize([emp,])

return self.render\_to\_http\_response(json\_data)

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return self.render\_to\_http\_response(json\_data)

* Test.py > add and run

def get\_resource(id=None):

data={}

if id is not None:

data= {

'id': id

}

resp = requests.get(BASE\_URL+ENDPOINT, data=json.dumps(data))

print(resp.status\_code)

print(resp.json())

get\_resource()

get\_resource(2)

* (testapp/views.py)> add

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeCRUDCBV(HttpresponseMixin, SerializerMixin, View):

def get\_object\_by\_id(self,id):

try:

emp = Employee.objects.get(id=id)

except Employee.DoesNotExist:

emp = None

return emp

def get(self, request, \*args,\*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = self.serialize([emp,])

return self.render\_to\_http\_response(json\_data)

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return self.render\_to\_http\_response(json\_data)

def post(self, request, \*args, \*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

empdata = json.loads(data)

form = EmployeeForm(empdata)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource created successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

def put(self, request, \*args, \*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is None:

json\_data = json.dumps({'msg':'please provide valid id for updation'})

return self.render\_to\_http\_response(json\_data, status=400)

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found. Not possible to update"})

return self.render\_to\_http\_response(json\_data, status=404)

provided\_data = json.loads(data)

original\_data = {

'eno':emp.eno,

'ename':emp.ename,

'esal':emp.esal,

'eaddr': emp.eaddr

}

original\_data.update(provided\_data)

form = EmployeeForm(original\_data, instance=emp)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource Updated successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

def delete(self, request, \*args, \*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

status,deleted\_item = emp.delete()

if status == 1:

json\_data = json.dumps({'Msg': "Resource deleted successfully"})

return self.render\_to\_http\_response(json\_data)

json\_data = json.dumps({'Error': "Unable to delete. Please try again"})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = json.dumps({'Error': "Unable to delete. Please provide valid id"})

return self.render\_to\_http\_response(json\_data, status=404)

* Test.py > add

# create\_resource()

def update\_resource\_b(id):

new\_emp = {

'id': id,

'esal': 10000,

'eaddr': 'Delhi',

}

new\_emp = json.dumps(new\_emp)

resp = requests.put(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

# update\_resource\_b(7)

def delete\_resource\_b(id):

data= {

'id': id

}

resp = requests.delete(BASE\_URL+ENDPOINT, data=json.dumps(data))

print(resp.status\_code)

print(resp.json())

delete\_resource\_b(7)

# With REST Frame work

Here we do the same rest api with django’s default rest framework.

* Create project > in cmd
  1. *(python36) E:\django projects>django-admin startproject withrestc4*
  2. *(python36) E:\django projects>cd withrestc4*
  3. *(python36) E:\django projects\withrestc4>python manage.py startapp testapp*
* (withrestc4/settings.py)> add inside ’ INSTALLED\_APPS’

'rest\_framework',

'testapp'

* (testapp/models.py)> add

class Employee(models.Model):

id = models.AutoField(primary\_key=True)

eno = models.IntegerField()

ename = models.CharField(max\_length=64)

esal = models.FloatField()

eaddr = models.CharField(max\_length=64)

* (testapp/admin.py)> add

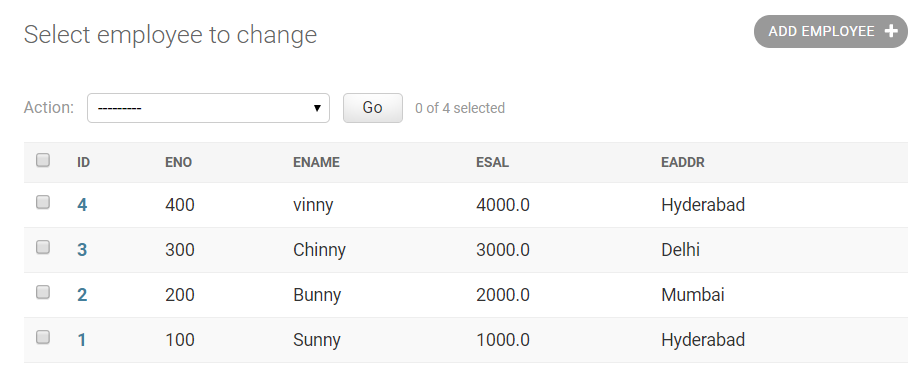
from testapp.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['id','eno', 'ename', 'esal', 'eaddr']

admin.site.register(Employee, EmployeeAdmin)

* Install package
  1. *(python36) E:\django projects\withrestc4>pip install djangorestframework*
* In cmd>
  1. *(python36) E:\django projects\withrestc4>python manage.py makemigrations*
  2. *(python36) E:\django projects\withrestc4>python manage.py migrate*
  3. *(python36) E:\django projects\withrestc4>python manage.py createsuperuser*
     + *Username (leave blank to use 'user'): admin*
     + *Email address: admin@example.com*
     + *Password:*
     + *Password (again):*
     + *Superuser created successfully.*
  4. *(python36) E:\django projects\withrestc4>python manage.py runserver*
* In browser <http://127.0.0.1:8000/admin/> add the following employees**
* (testapp/)> create new file ‘serializers.py’

from testapp.models import Employee

from rest\_framework.serializers import ModelSerializer

class EmployeeSerializer(ModelSerializer):

class Meta:

model = Employee

fields = '\_\_all\_\_'

* (withrestc4/urls.py)> add

from django.contrib import admin

from django.urls import path, include

from rest\_framework import routers

from testapp import views

router = routers.DefaultRouter()

#router.register('api', views.EmployeeCRUDCBV, base\_name='api')

router.register('api', views.EmployeeCRUDCBV)

urlpatterns = [

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

path("", include(router.urls))

]

* (testapp/views.py)> add

from django.shortcuts import render

from rest\_framework.viewsets import ModelViewSet

from testapp.models import Employee

from testapp.serializers import EmployeeSerializer

# Create your views here.

class EmployeeCRUDCBV(ModelViewSet):

queryset = Employee.objects.all()

serializer\_class = EmployeeSerializer

* <http://127.0.0.1:8000/api/> , <http://127.0.0.1:8000/api/1/> and add an employee

# Reference

1. <https://docs.djangoproject.com/en/2.2/>