Create virtual environment

virtualenv myenv -p python3

goto scripts folder n run activate.bat file

source ./myenv/bin/activate

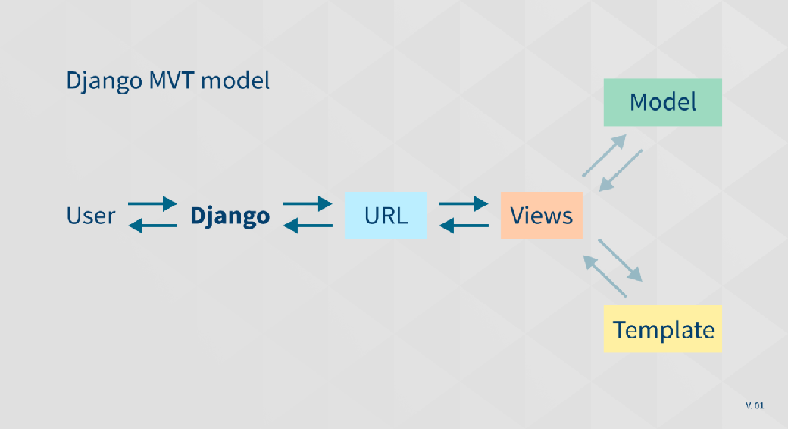
pip install django

Create project

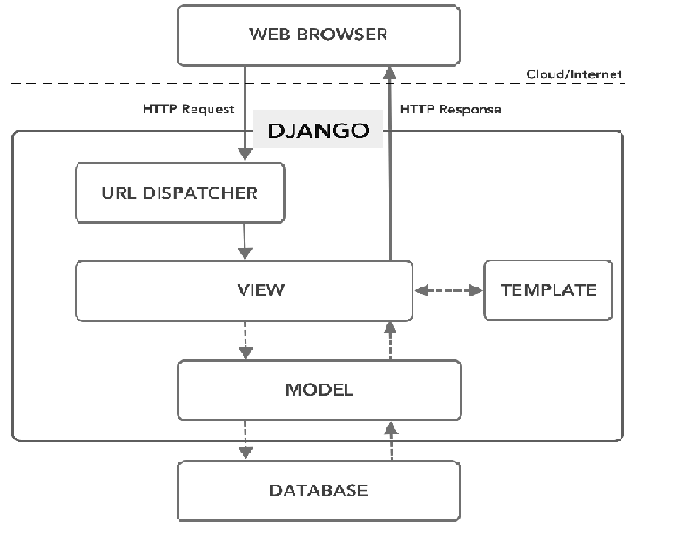
Myenv>> django-admin startproject musiclibrary

cd musiclibrary

Myenv>>....\Scripts>>



The Django MVT – Model Views Template



project

musiclibrary

--manage.py

--musiclibrary

----\_\_init\_\_.py

----manage.py

----settings.py

----urls.py

----wsgi.py

The explanation of the above files is as follows:

**\_\_init\_\_.py:** A file required by Python, This file tells python to view this directory (musiclibrary folder) as python packages. Also, the above file is an empty file, and usually, you will not add anything inside it.

**manage.py**: One of the advantages of the command line is that it allows you to interact with this project in different ways. It allows you to do things like creating users for your website and access to the database. You should never edit or add anything to this file.

**settings.py**: This is a file that contains all of the Settings/configurations for your entire website. Take a look at it to see the available settings, along with the default values.

**urls.py**: To put it simply url.py is like a table of content for the Django website. It will direct the user requests to an appropriate view function to handle them.

**wsgi.py:** This file is used in deploying the project and to help your Django application communicate with the webserver. Much more on this in future sections dedicated to the deployment of Django projects.

## create a view.py file

## Write our first Hello World!

The example above was a very simple example of how views work. Most of the time, views do more complex things.  Open the views.py file that you just created and copy-paste this little code snippet in it.

from django.shortcuts import render

from django.http import HttpResponse

​

def hello\_world(request):

return HttpResponse("Hello World!")

## URLs in Django

The default urls.py file inside the main app folder looks something like this.

from django.contrib import admin

from django.urls import path

​

urlpatterns = [

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

]

Add line in urls.py =🡺 path(‘say-hello/’,views.hello\_world),

from django.contrib import admin

from django.urls import path

from .import views

​urlpatterns = [

path('say-hello/', views.hello\_world),

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

​

]

​

## Run local development server

python manage.py runserver

velopment server and by default, it will be reachable using your localhost on port 8000. Open a (<http://127.0.0.1:8000/say-hello/>)

What is an App?

An app is a web application that has a specific meaning in your project, like a home page, a contact form, or a members database.

In this tutorial we will create an app that allows us to list and register members in a database.

But first, let's just create a simple Django app that displays "Hello World!".

Create App

D:\>my\_tennis\_club> py manage.py startapp members

Django creates a folder named members in my project, with this content:

my\_tennis\_club  
    manage.py  
    my\_tennis\_club/  
    members/  
        migrations/  
            \_\_init\_\_.py  
        \_\_init\_\_.py  
        admin.py  
        apps.py  
        models.py  
        tests.py  
        views.py

## Views

Django views are Python functions that takes http requests and returns http response, like HTML documents.

A web page that uses Django is full of views with different tasks and missions.

Views are usually put in a file called views.py located on your app's folder.

There is a views.py in your members folder that looks like this:

my\_tennis\_club/members/views.py:

from django.shortcuts import render

from django.http import HttpResponse

def members(request):

return HttpResponse("Hello world!")

## URLs

Create a file named urls.py in the same folder as the views.py file, and type this code in it:

my\_tennis\_club/members/urls.py:

from django.urls import path

from . import views

urlpatterns = [

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

]

The urls.py file you just created is specific for the members application. We have to do some routing in the root directory my\_tennis\_club as well. This may seem complicated, but for now, just follow the instructions below.

There is a file called urls.py on the my\_tennis\_club folder, open that file and add the include module in the import statement, and also add a path() function in the urlpatterns[] list, with arguments that will route users that comes in via 127.0.0.1:8000/.

Then your file will look like this:

my\_tennis\_club/my\_tennis\_club/urls.py:

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

path('', include('members.urls')),

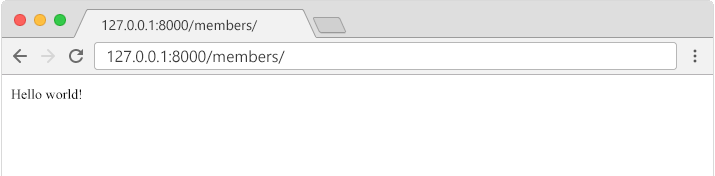
path('admin/', admin.site.urls),

]

If the server is not running, navigate to the /my\_tennis\_club folder and execute this command in the command prompt:

py manage.py runserver

In the browser window, type [127.0.0.1:8000/members/](http://127.0.0.1:8000/members/) in the address bar.



## Templates

In the [Django Intro](https://www.w3schools.com/django/django_intro.php) page, we learned that the result should be in HTML, and it should be created in a template, so let's do that.

Create a templates folder inside the members folder, and create a HTML file named myfirst.html.

The file structure should be like this:

my\_tennis\_club  
    manage.py  
    my\_tennis\_club/  
    members/  
        templates/  
            myfirst.html

Open the HTML file and insert the following:

my\_tennis\_club/members/templates/myfirst.html:

<!DOCTYPE html>

<html>

<body>

<h1>Hello World!</h1>

<p>Welcome to my first Django project!</p>

</body>

</html>

## Modify the View

Open the views.py file and replace the members view with this:

my\_tennis\_club/members/views.py:

from django.http import HttpResponse

from django.template import loader

def members(request):

template = loader.get\_template('myfirst.html')

return HttpResponse(template.render())

## Change Settings

To be able to work with more complicated stuff than "Hello World!", We have to tell Django that a new app is created.

This is done in the settings.py file in the my\_tennis\_club folder.

Look up the INSTALLED\_APPS[] list and add the members app like this:

my\_tennis\_club/my\_tennis\_club/settings.py:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'members'

]

Then run this command:

py manage.py migrate

Which will produce this output:

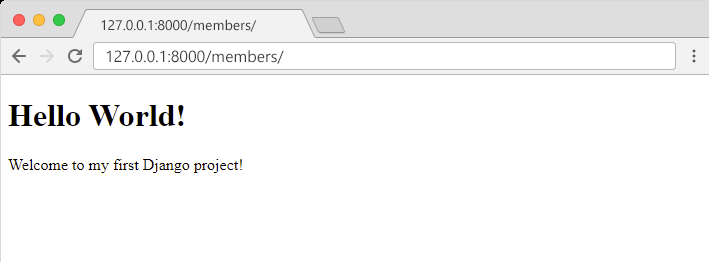
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, sessions  
Running migrations:  
  Applying contenttypes.0001\_initial... OK  
  Applying auth.0001\_initial... OK  
  Applying admin.0001\_initial... OK  
  Applying admin.0002\_logentry\_remove\_auto\_add... OK  
  Applying admin.0003\_logentry\_add\_action\_flag\_choices... OK  
  Applying contenttypes.0002\_remove\_content\_type\_name... OK  
  Applying auth.0002\_alter\_permission\_name\_max\_length... OK  
  Applying auth.0003\_alter\_user\_email\_max\_length... OK  
  Applying auth.0004\_alter\_user\_username\_opts... OK  
  Applying auth.0005\_alter\_user\_last\_login\_null... OK  
  Applying auth.0006\_require\_contenttypes\_0002... OK  
  Applying auth.0007\_alter\_validators\_add\_error\_messages... OK  
  Applying auth.0008\_alter\_user\_username\_max\_length... OK  
  Applying auth.0009\_alter\_user\_last\_name\_max\_length... OK  
  Applying auth.0010\_alter\_group\_name\_max\_length... OK  
  Applying auth.0011\_update\_proxy\_permissions... OK  
  Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK  
  Applying sessions.0001\_initial... OK  
  
(myworld) C:\Users\Your Name\myworld\my\_tennis\_club>

Start the server by navigating to the /my\_tennis\_club folder and execute this command:

py manage.py runserver

In the browser window, type [127.0.0.1:8000/members/](http://127.0.0.1:8000/members/) in the address bar.

The result should look like this:



Django Models

Up until now in this tutorial, output has been static data from Python or HTML templates.

Now we will see how Django allows us to work with data, without having to change or upload files in the prosess.

In Django, data is created in objects, called Models, and is actually tables in a database.

Create Table (Model)

To create a model, navigate to the models.py file in the /members/ folder.

Open it, and add a Member table by creating a Member class, and describe the table fields in it:

my\_tennis\_club/members/models.py:

from django.db import models

class Member(models.Model):

firstname = models.CharField(max\_length=255)

lastname = models.CharField(max\_length=255)

The first field, firstname, is a Text field, and will contain the first name of the members.

The second field, lastname, is also a Text field, with the member's last name.

Both firstname and lastname is set up to have a maximum of 255 characters.

SQLite Database

When we created the Django project, we got an empty SQLite database.

It was created in the my\_tennis\_club root folder, and has the filename db.sqlite3.

By default, all Models created in the Django project will be created as tables in this database.

Migrate

Now when we have described a Model in the models.py file, we must run a command to actually create the table in the database.

Navigate to the /my\_tennis\_club/ folder and run this command:

py manage.py makemigrations members

Which will result in this output:

Migrations for 'members':  
  members\migrations\0001\_initial.py  
    - Create model Member  
  
(myworld) C:\Users\*Your Name*\myworld\my\_tennis\_club>

Django creates a file describing the changes and stores the file in the /migrations/ folder:

**my\_tennis\_club/members/migrations/0001\_initial.py:**

# Generated by Django 4.1.2 on 2022-10-27 11:14

from django.db import migrations, models

class Migration(migrations.Migration):

initial = True

dependencies = [

]

operations = [

migrations.CreateModel(

name='Member',

fields=[

('id', models.BigAutoField(auto\_created=True, primary\_key=True, serialize=False, verbose\_name='ID')),

('firstname', models.CharField(max\_length=255)),

('lastname', models.CharField(max\_length=255)),

],

),

]

Note that Django inserts an id field for your tables, which is an auto increment number (first record gets the value 1, the second record 2 etc.), this is the default behavior of Django, you can override it by describing your own id field.

The table is not created yet, you will have to run one more command, then Django will create and execute an SQL statement, based on the content of the new file in the /migrations/ folder.

Run the migrate command:

py manage.py migrate

Which will result in this output:

Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, members, sessions  
Running migrations:  
  Applying members.0001\_initial... OK  
  
(myworld) C:\Users\*Your Name*\myworld\my\_tennis\_club>

Now you have a Member table in you database!

View SQL

As a side-note: you can view the SQL statement that were executed from the migration above. All you have to do is to run this command, with the migration number:

py manage.py sqlmigrate members 0001

Which will result in this output:

BEGIN;  
--  
-- Create model Member  
--  
CREATE TABLE "members\_member" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "firstname" varchar(255) NOT NULL, "lastname" varchar(255) NOT NULL); COMMIT;

## Add Records

To open a Python shell, type this command:

**py manage.py shell**

Now we are in the shell, the result should be something like this:

Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
(InteractiveConsole)  
>>>

At the bottom, after the three >>> write the following:

>>> from members.models import Member

Hit [enter] and write this to look at the empty Member table:

>>> Member.objects.all()

This should give you an empty QuerySet object, like this:

<QuerySet []>

A QuerySet is a collection of data from a database.

Read more about QuerySets in the [Django QuerySet](https://www.w3schools.com/django/django_queryset.php) chapter.

Add a record to the table, by executing these two lines:

>>> member = Member(firstname='Srikar', lastname='Thangella')  
>>> member.save()

Execute this command to see if the Member table got a member:

>>> Member.objects.all().values()

Hopefully, the result will look like this:

<QuerySet [{'id': 1, 'firstname': 'Srikar', 'lastname': 'Thangella'}]>

Adding multiple records:

>>> member1 = Member(firstname='Tobias', lastname='Refsnes')  
>>> member2 = Member(firstname='Linus', lastname='Refsnes')  
>>> member3 = Member(firstname='Lene', lastname='Refsnes')  
>>> member4 = Member(firstname='Stale', lastname='Refsnes')  
>>> member5 = Member(firstname='Jane', lastname='Doe')  
>>> members\_list = [member1, member2, member3, member4, member5]  
>>> for x in members\_list:  
>>>   x.save()

To view the multiple records added:

Member.objects.all().values()  
<QuerySet [{'id': 1, 'firstname': 'Emil', 'lastname': 'Refsnes'},  
{'id': 2, 'firstname': 'Tobias', 'lastname': 'Refsnes'},  
{'id': 3, 'firstname': 'Linus', 'lastname': 'Refsnes'},  
{'id': 4, 'firstname': 'Lene', 'lastname': 'Refsnes'},  
{'id': 5, 'firstname': 'Stale', 'lastname': 'Refsnes'},  
{'id': 6, 'firstname': 'Jane', 'lastname': 'Doe'}]>

## Create Template

After creating Models, with the fields and data we want in them, it is time to display the data in a web page.

Start by creating an HTML file named all\_members.html and place it in the /templates/ folder:

my\_tennis\_club/members/templates/all\_members.html:

<!DOCTYPE html>

<html>

<body>

<h1>Members</h1>

<ul>

{% for x in mymembers %}

<li>{{ x.firstname }} {{ x.lastname }}</li>

{% endfor %}

</ul>

<h1>Members</h1>

<ul>

{% for x in mymembers %}

<li><a href="details/{{ x.id }}">{{ x.firstname }} {{ x.lastname }}</a></li>

{% endfor %}

</ul>

</body>

</html>

Do you see the {% %} brackets inside the HTML document?

They are Django Tags, telling Django to perform some programming logic inside these brackets.

Start by creating a new template called details.html:

my\_tennis\_club/members/templates/details.html:

<!DOCTYPE html>

<html>

<body>

<h1>{{ mymember.firstname }} {{ mymember.lastname }}</h1>

<p>Back to <a href="/members">Members</a></p>

</body>

</html>

Create new View

Then create a new view in the views.py file, that will deal with incoming requests to the /details/ url:

my\_tennis\_club/members/views.py:

from django.http import HttpResponse

from django.template import loader

from .models import Member

def members(request):

mymembers = Member.objects.all().values()

template = loader.get\_template('all\_members.html')

context = {

'mymembers': mymembers,

}

return HttpResponse(template.render(context, request))

def details(request, id):

mymember = Member.objects.get(id=id)

template = loader.get\_template('details.html')

context = {

'mymember': mymember,

}

return HttpResponse(template.render(context, request))

The details view does the following:

* Gets the id as an argument.
* Uses the id to locate the correct record in the Member table.
* loads the details.html template.
* Creates an object containing the member.
* Sends the object to the template.
* Outputs the HTML that is rendered by the template.

## Add URLs

Now we need to make sure that the /details/ url points to the correct view, with id as a parameter.

Open the urls.py file and add the details view to the urlpatterns list:

my\_tennis\_club/members/urls.py:

from django.urls import path

from . import views

urlpatterns = [

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

path('members/details/<int:id>', views.details, name='details'),

]

[Run Example »](https://www.w3schools.com/django/showdjango.php?filename=demo_add_link_details)

If you have followed all the steps on your own computer, you can see the result in your own browser: [127.0.0.1:8000/members/](http://127.0.0.1:8000/members/).

If the server is down, you have to start it again with the runserver command:

py manage.py runserver

[127.0.0.1:8000/members/](http://127.0.0.1:8000/members/)details/1