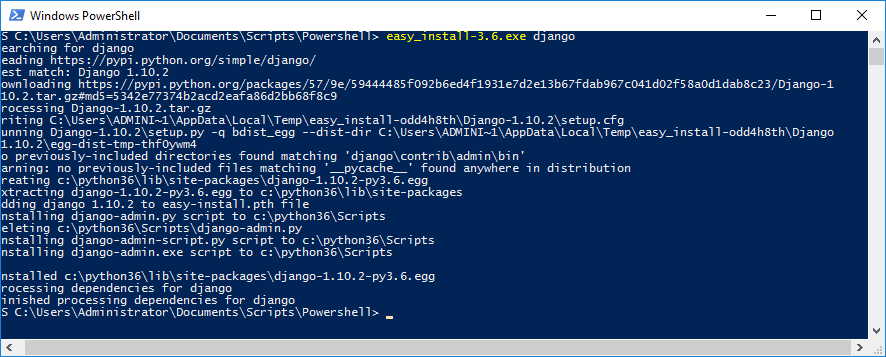
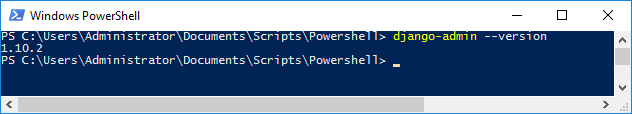
How to install Django

Best Practices – http://goo.gl/PRrEMe

1. Install python 3.5 or latest
2. Add “C:\Python36\Scripts” into your system variable path
3. Make sure you have “easy\_install” and “pip” installed under “C:\Python36\Scripts”
4. Open powershell and type “easy\_install django”

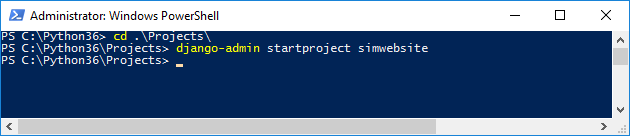


1. Check if django is installed correctly by typing “django-admin --version" in the powershell console.

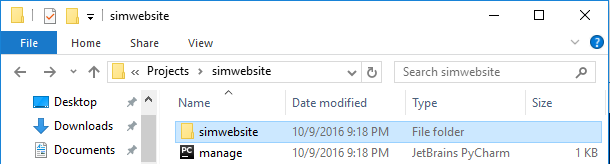


**Create django project**

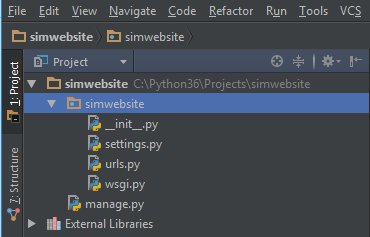
1. In powershell, navigate to the directory where you want your project folder to be created.
2. Type “django-admin startproject simwebsite”



1. Go to the current folder and look for the new project folder you created ex: simwebsite



**Open your project in pyCharm**



\_\_init\_\_.ph = File that tells python to view the “simwebsite” directory as a python package. No code inside this file

settings.py = Main settings and configuration file of your website.

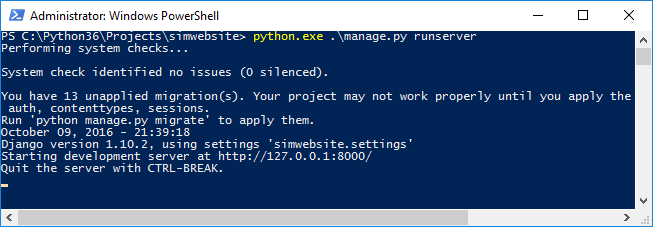
urls.py = URL declaration or table of contents of your website. Handles page directory direction.

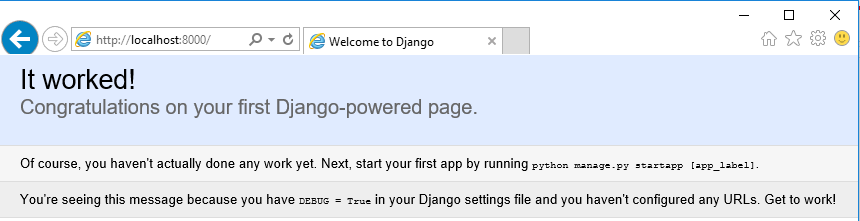
wsgi.py = Web server gateway interface.

manage.py =

**How to run django webserver and site**

1. Open powershell or cmd
2. Navigate to the website directory you have created.
3. In powershell, type “python.exe .\manage.py runserver” to start your local web server

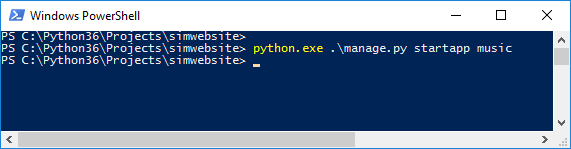


1. Open your browser and go to <http://localhost:8000> or <http://127.0.0.1:8000> and you should see it worked.
2. To stop your web server, press Ctrl+C

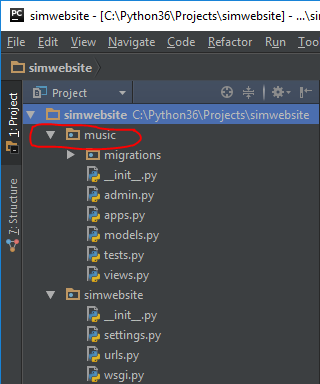
**Creating a New App**

An app in django is just a subfolders on your site ex: /forum, /photosvideos, /documents

1. In powershell, navigate to the directory where you created your project folder.
2. Type “python manage.py startapp appname” ex: “python manage.py startapp music”



1. View your newly created app on pyCharm or the directory where your project is created.



**Definitions**:

**Migrations** - Use to hookup or connect your website and source code into a database

**\_\_init\_\_.py** = Tells python to treat this directory as a package.

**admin.py** = Manage users and access database

**apps.py =** Configuration Settings of the app

**models.py** = A template for your database.

**tests.py =** Test file to make sure you don’t have bugs on your code

**views.py =** Are python functions that take a user request and respond them the result**.**

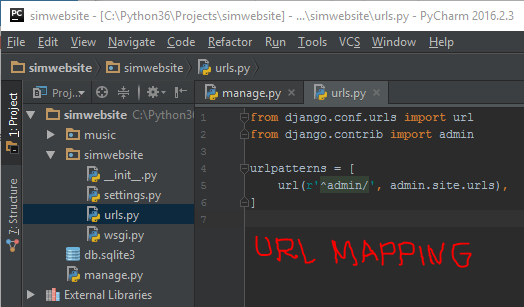
**Django admin page** <http://localhost:8000/admin>

Request

User

Django Web Page

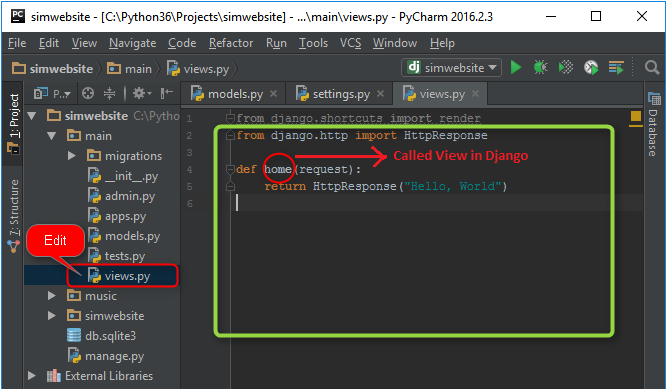
Django will look



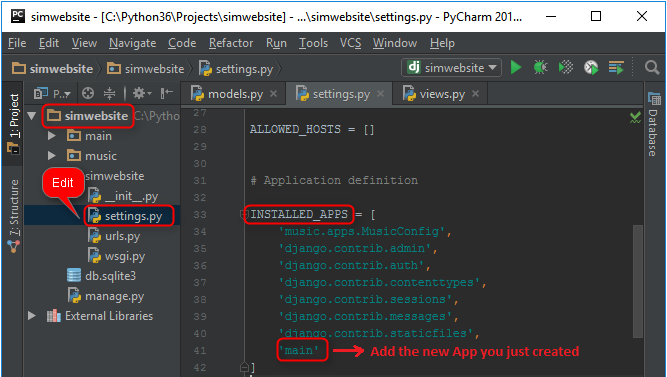
Website/urls.py

**Adding a New Page**

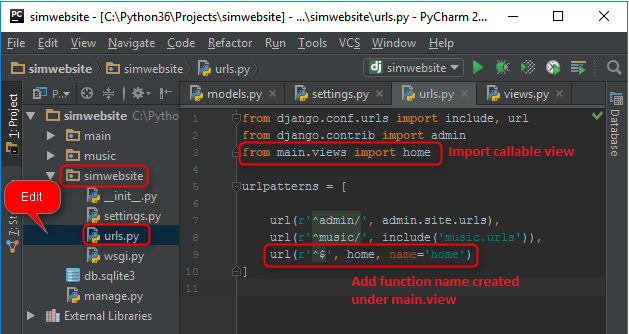
1. Create a new project ex: “python manage.py startapp main”
2. Edit “views.py” and add code to it.



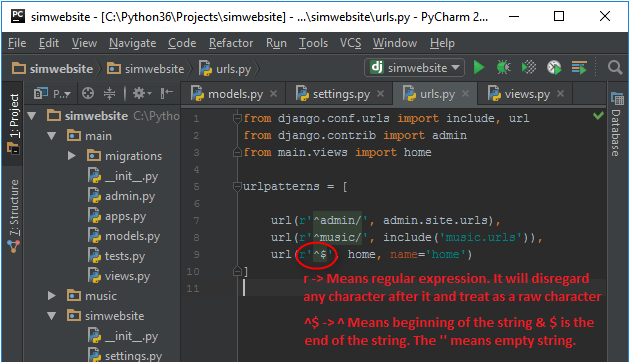
1. Navigate to the main site app and edit “views.py”
2. Add the new app created under “INSTALLED\_APPS”. See sample below



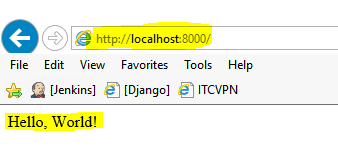
1. Navigate to the main web app and edit “urls.py” and import the callable view created in other app (ex: main) and add the function name under “urlpatterns” ex: url(r'^$', home, name='home')



**Note**: r 🡪 Means regular expression. More on regular expressions here <http://goo.gl/5uJsfy>



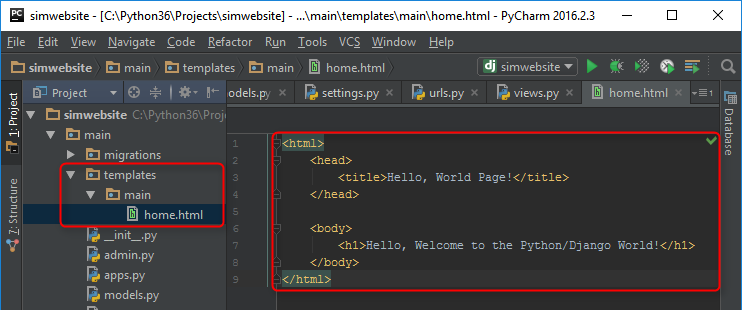
1. Save the project and refreshed <http://localhost:8000/>



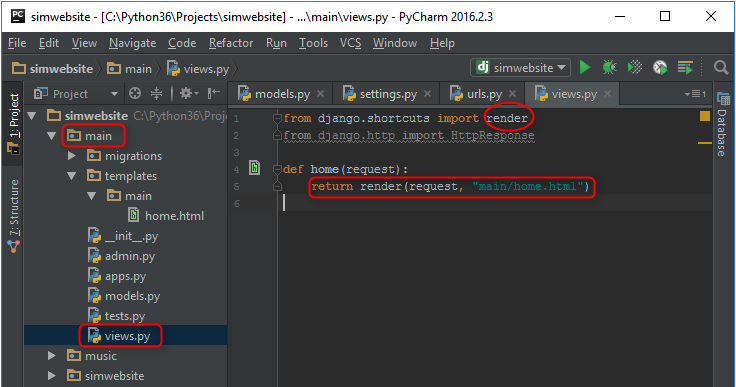
**Templates**

Another kind of Django component which helps html generation. We use template to render html page.

1. Create a new folder name “template” under the “main” app.
2. Create a new folder name “main” under the newly created template folder named “template”.
3. Create a new file name “home.html” under “main -> templates -> main” folder.



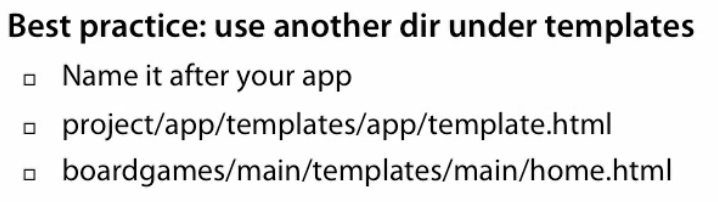
1. Edit “views.py” under the “main” app and edit the home function to return render result on http requests.



1. Save the project and refreshed <http://localhost:8000/>



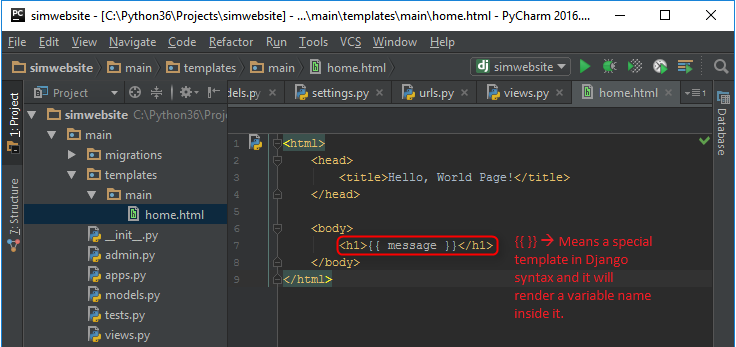
**Note**: We render a template from a view either via JSON or CSV using: django.shortcuts.render



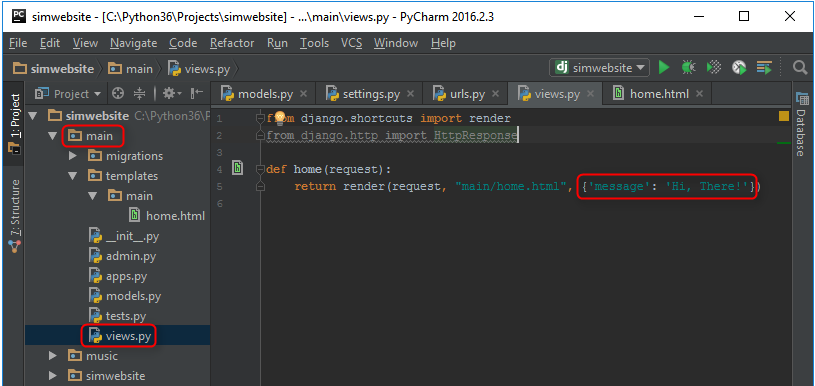
**Adding Variable to an HTML Page**

1. Edit the file name “home.html” under “main -> templates -> main” folder.
2. Add {{ message }} variable.

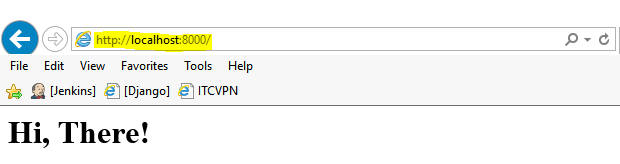
Note: {{ }} 🡪 Means a special template in Django syntax and it will render a variable name inside it.



1. To add a value to that variable, Edit “views.py” under the “main” app and edit the home function and add a dictionary value for the message variable. Ex: {'message': 'Hi, There!'}



1. Save the project and refreshed <http://localhost:8000/>

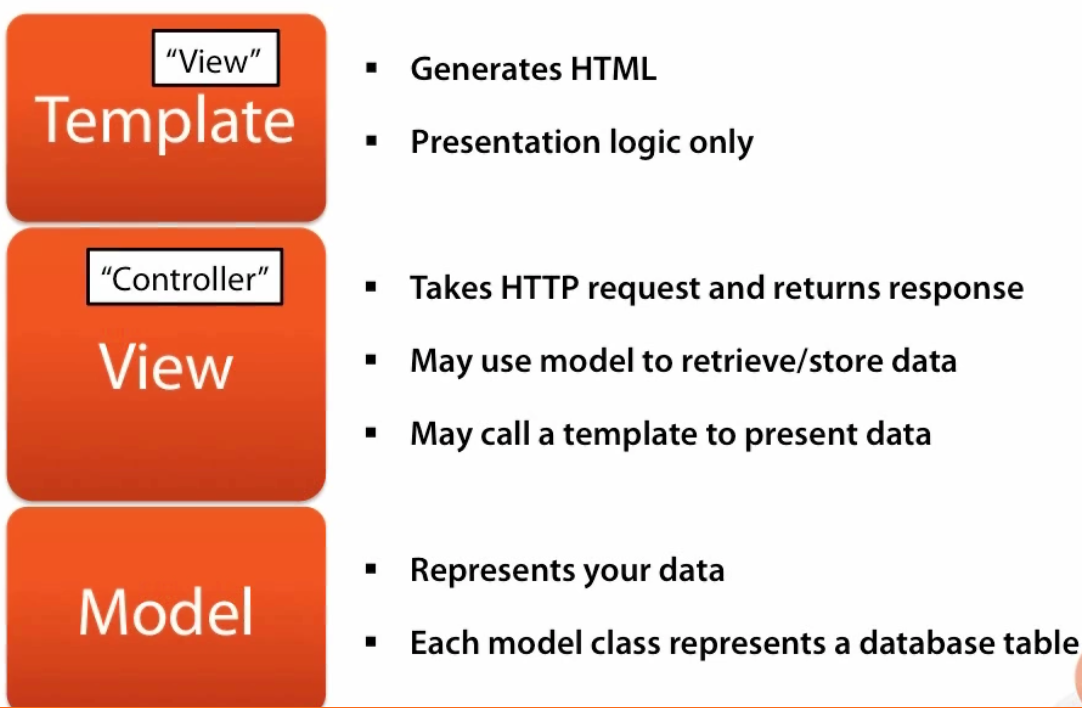


**Adding CSS Styling into an HTML Page**

You can download CSS styling bootstrap at <https://bootswatch.com/>

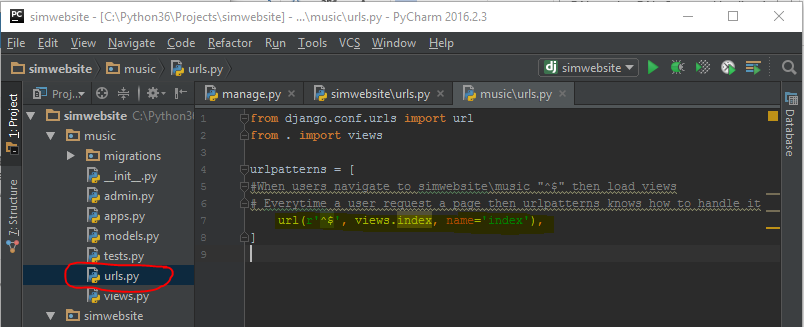
1. Create a new folder name “static” under the root project directory “simwebsite”.
2. Create a new folder name “bootstrap” under the new folder “static”.
3. Create a new folder name “css”,”fonts” & “js” under the new folder “bootstrap”.

**Model Template View**

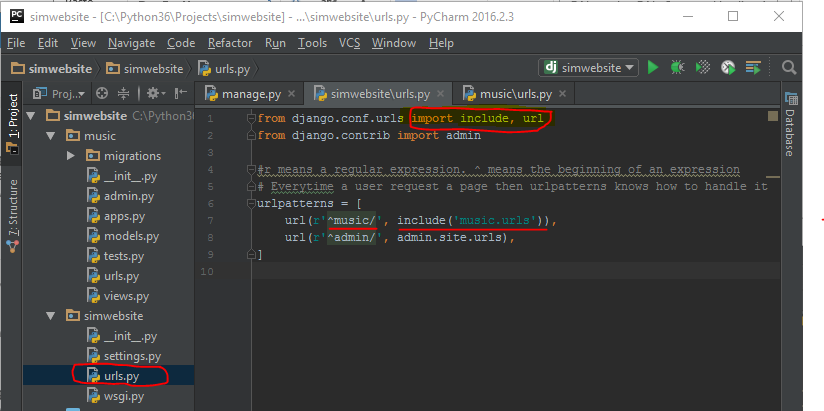


**Configure URL Mappings and Views**

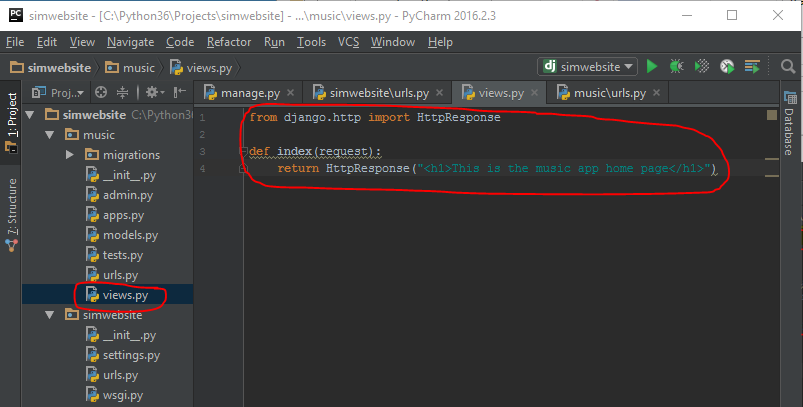
1. Add url.py under the new app created ex: music
2. Configure App subfolder urlpatterns to look for views.index



1. Configure the main website folder urls.py urlpatterns to look for the new page you created ex: music
2. Import a module “from django.conf.urls import url”



1. Configure views.py and add a HTTP response to the users every time they access the app folder.



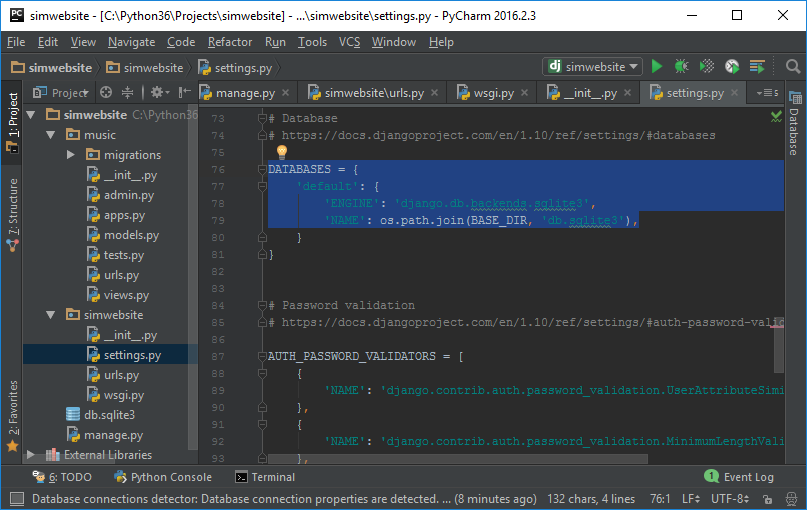
**How it works?**

A user to the website and requested music, django will always look on the main site “urls.py” file for url mapping pattern. When it found a match, then it will go “include(‘music/urls’)“ or go to this file “music/urls.py”. Once in app folder, it will look again under “urls.py” file and look for url mappings. When it sees “r’^$’” (r = regular expression or raw string, ^ just tells python to start here, $ just tells python that it’s the end). Under url patterns it will look next for a function called views.index. It will now in “views.py” file and return HTTPResponse to the user.

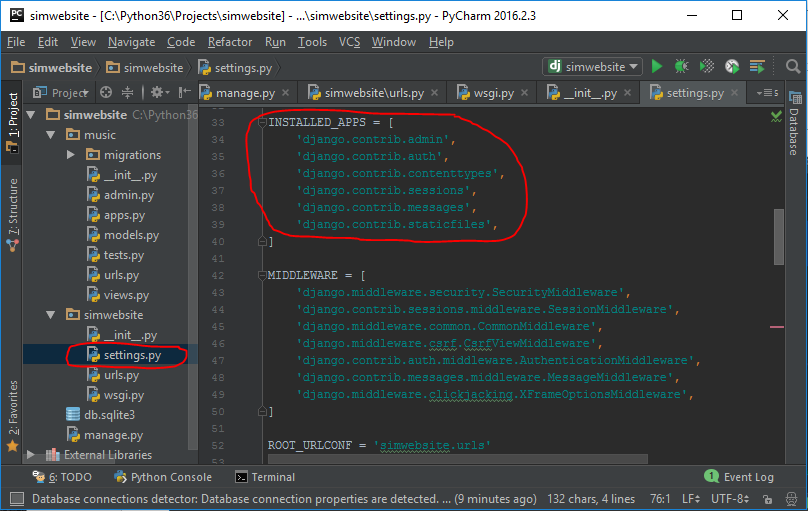
**Configuring the Database**

By default Django is installed with a lightweight database called SQLite3

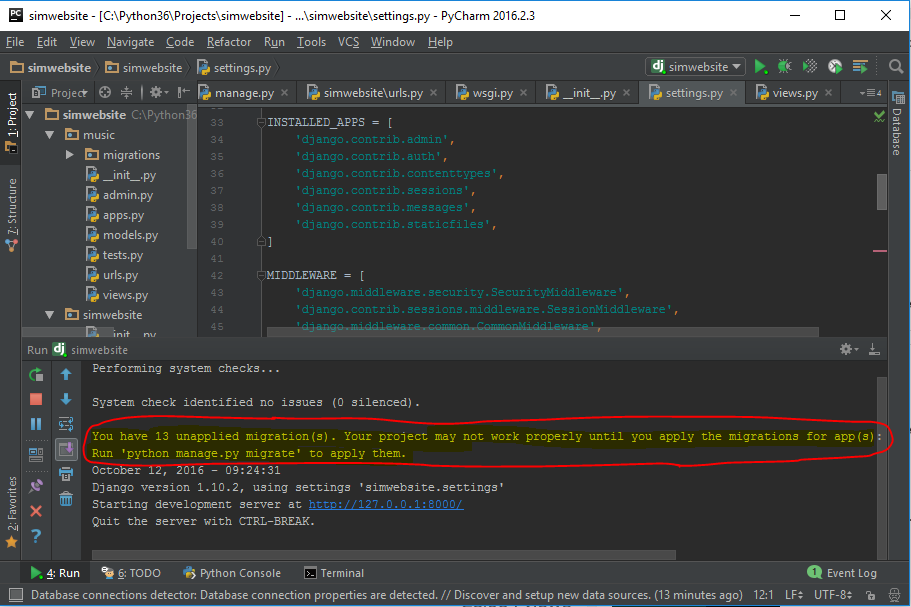
To check the database connection strings, open the main website settings.py



By default, Django also installs a default apps

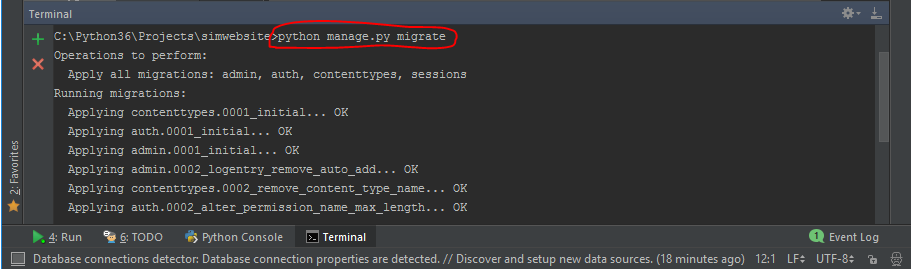


Errors when running the server for unapplied migration means that your apps are not in sync with your database.

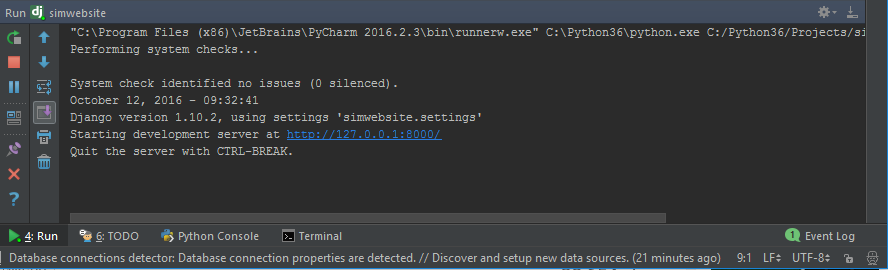


To sync to apps with the database, then please follow the steps below.

1. Open Command Prompt, type “python manage.py migrate”. It’s going to go on website settings and look for installed apps to sync up all the codes. It will also go to each apps directory and look for what tables are needed to work with the list of apps.



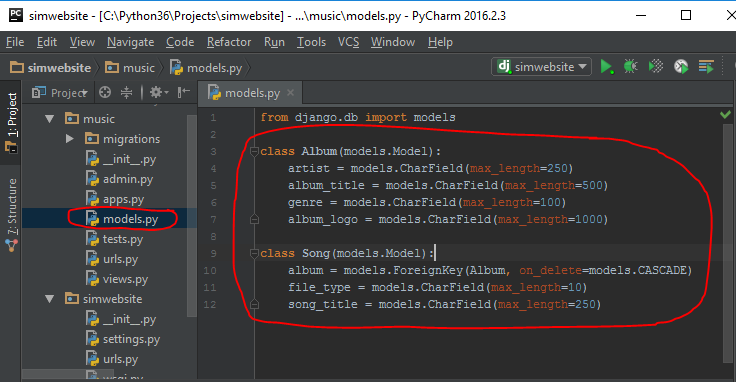
1. Open Command Prompt again and type “python manage.py migrate”. It will now show that system check identifies no issues on “website.settings”



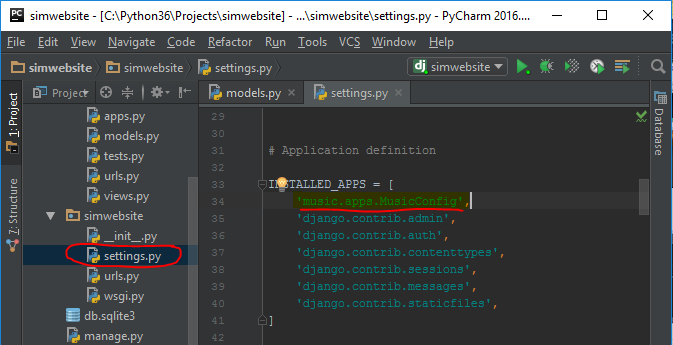
**Creating Database Model**

A model is a blueprint on how we are going to store our data and what are the types of data that we’re storing are.

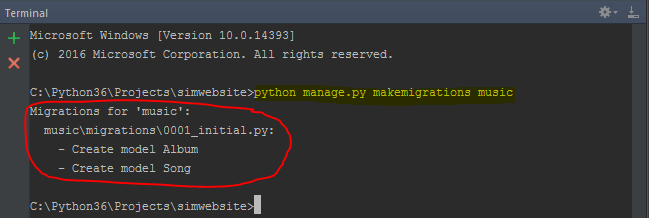
1. Navigate to the new apps folder then select “models.py” file. On this file, we will create a class and automatically Django will get each variable on each class and convert it into a column/table in the database.
2. Create a class code block under “models.py”. This class will turn into table columns later on the database.



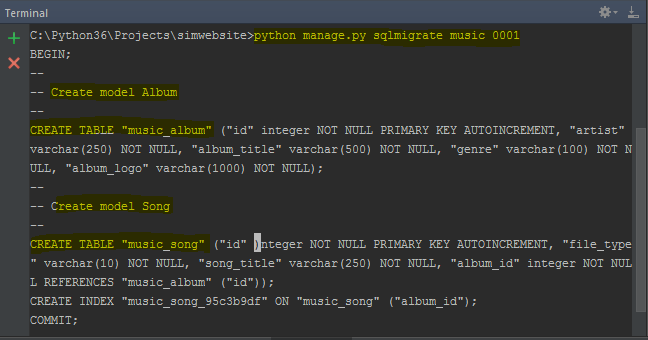
1. Go to the main website folder and open “settings.py”. Add the app you just created (ex: music) under INSTALLED\_APPS code block.



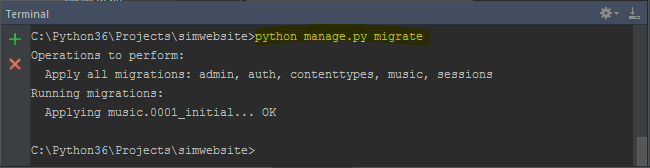
1. To create the actual table on “db.sqlite3” from models.py in step 2 then you need to run “python manage.py makemigrations appname” on the command line.



1. To view the details of the migration, you can type this command on the command line “>python manage.py sqlmigrate music 0001”. A migration is just a change on the Django database.



1. To commit the database migration changes you need to type “python manage.py migrate”. This will create the table on the database.



**Note**: To make any changes on the database tables, just update the models.py code and then follow step 4 & 6.

**Database API**

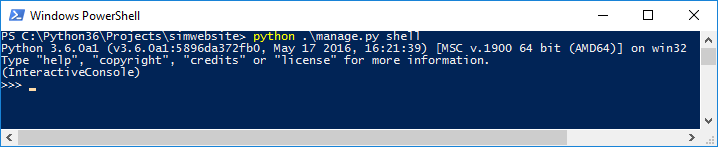
1. Open Powershell
2. PS C:\Python36\Projects\simwebsite> python .\manage.py shell

Python 3.6.0a1 (v3.6.0a1:5896da372fb0, May 17 2016, 16:21:39) [MSC v.1900 64 bit (AMD64)] on win32

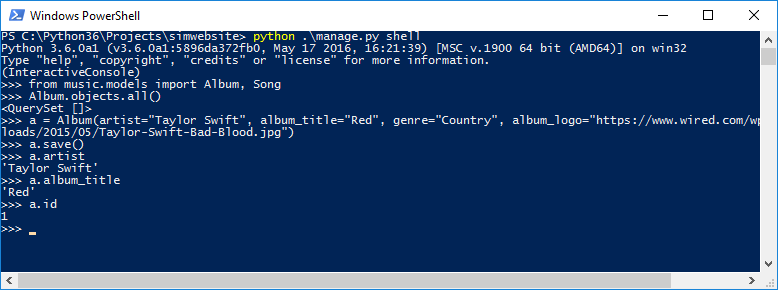
Type "help", "copyright", "credits" or "license" for more information.

(InteractiveConsole)

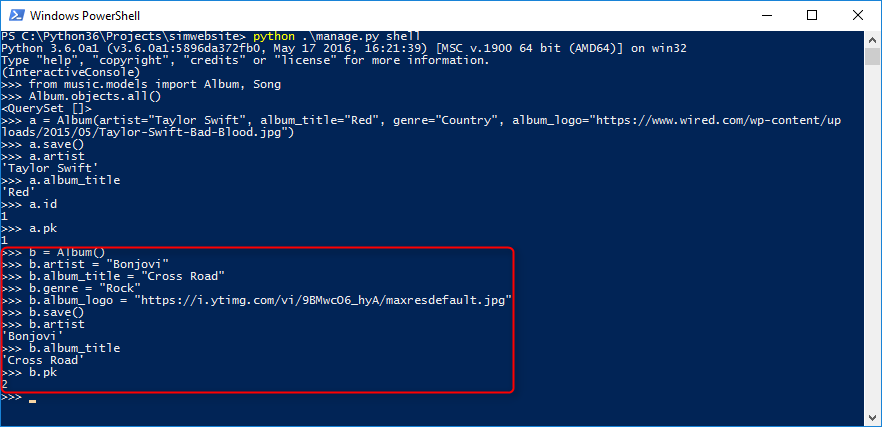
>>>



1. Create a variable and assign data to be save to the database. Ex: a = Album(artist="Taylor Swift", album\_title="Red", genre="Country", album\_logo=<https://www.wired.com/wp-content/picture.jpg>)

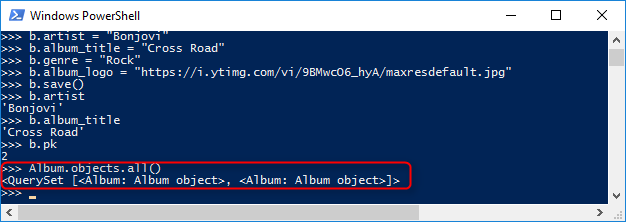


1. Using the database api, we can create another variable that we can use later and save to the database.

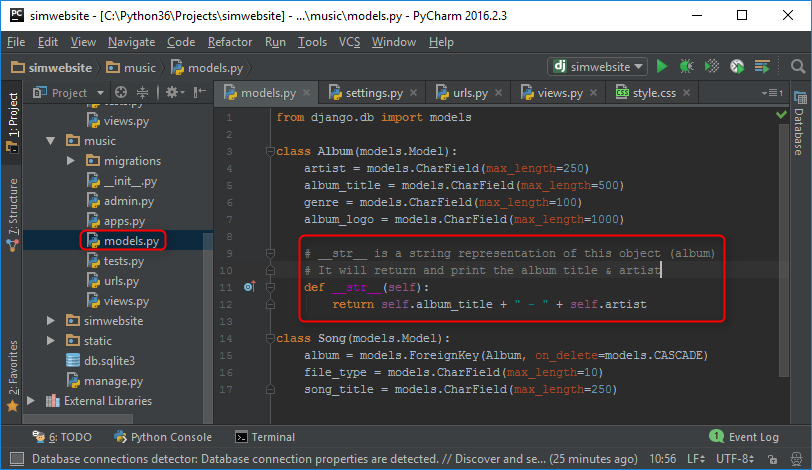


**Filtering Database Results**

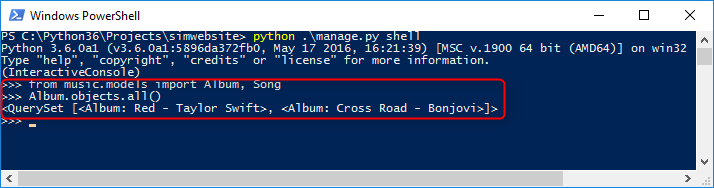
To filter or change album objects results then we need to add a function/method on the app models.py to return the title and the artist.



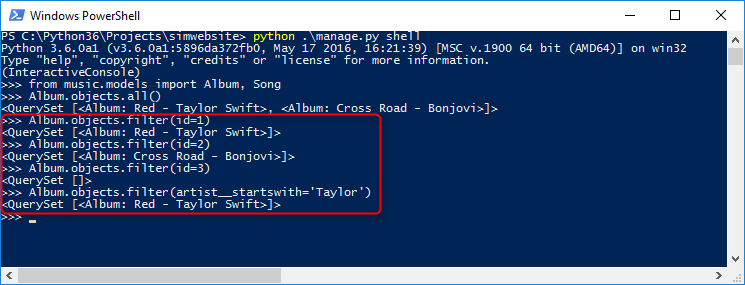
Edit the app “models.py”



* Exit the Database API shell and open it again.
* Open Powershell and type “python .\manage.py shell” then import “from music.models import Album, Song”
* Type “Album.objects.all()”. We should now see the name of the album & artist saved on the DB.

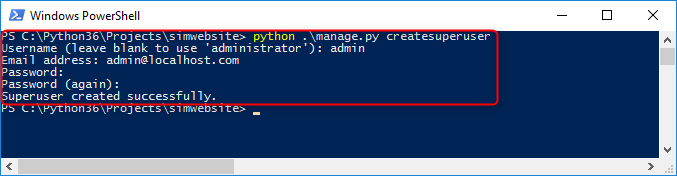


To filter the results, type “Album.objects.filter(id=1)” or “Album.objects.filter(artist\_\_startswith='Taylor')”

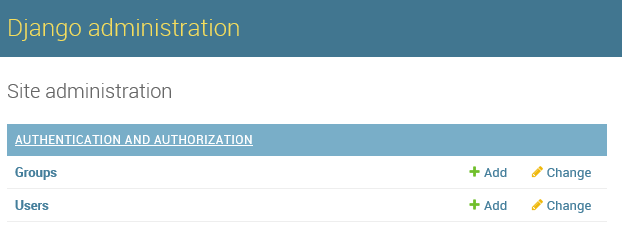


**Admin Interface**

1. To create a new user, type “python .\manage.py createsuperuser”



1. You should be able to login now to <http://localhost:8000/admin>

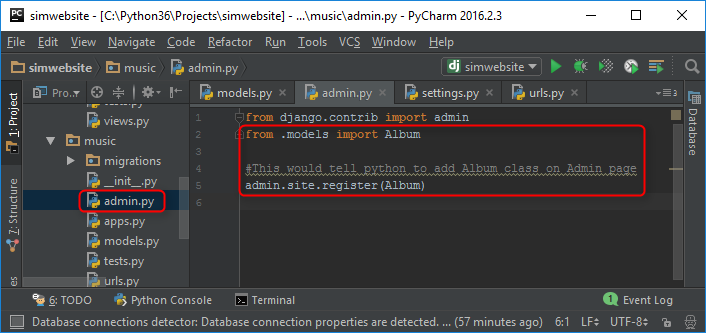


**Manage Database Tables using Admin Interface**

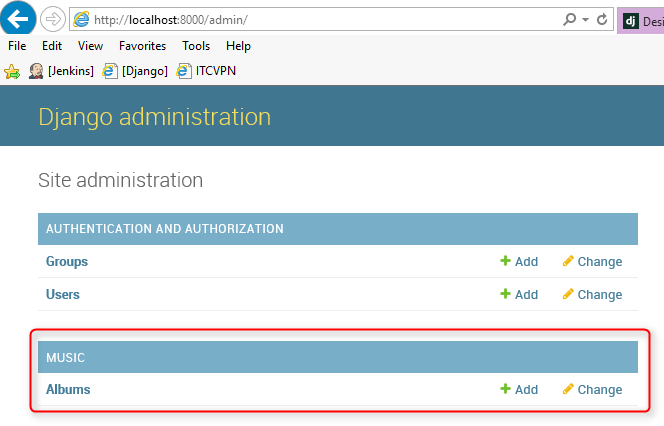
1. Update the “admin.py” file under the current app (music) and add the following:

from .models import Album

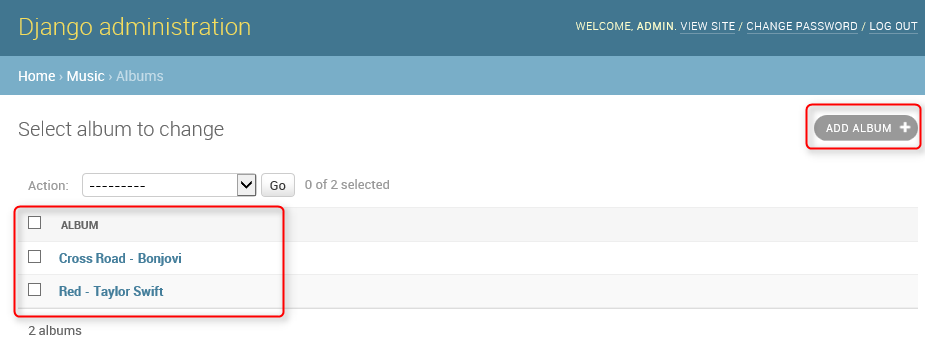
admin.site.register(Album)



1. Save the project and refresh <http://localhost:8000/admin> page.



1. You can now view, add or edit albums



**Adding View**

What is a view? A view is just a function that returns html page. Each url patterns are connected into a view

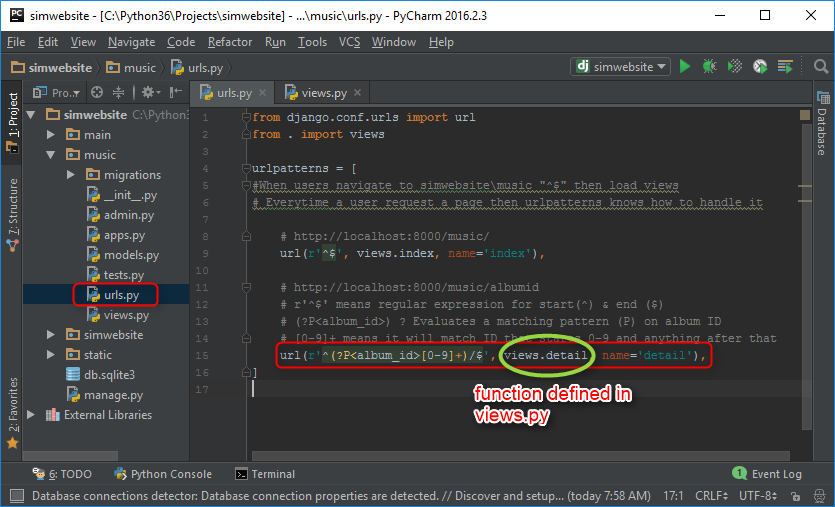
1. Navigate through the current app and select then edit “urls.py” file
2. Add a regular expression “r'^(?P<album\_id>[0-9]+)/$'” that will match the url pattern pass by the user

Ex: <http://localhost:8000/music/1/> or <http://localhost:8000/music/233/>

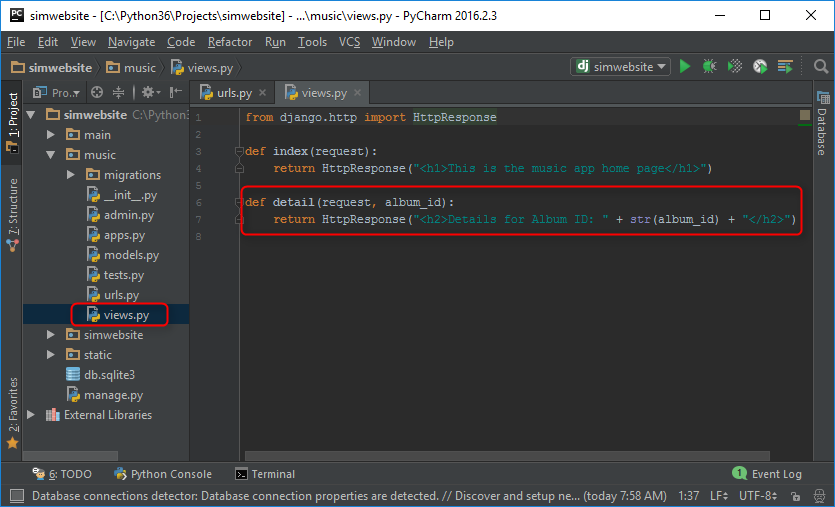
r'^$' means regular expression for start(^) & end ($)

(?P<album\_id>) Evaluates a matching pattern (P) on album ID

[0-9]+/ means it will match ID that starts 0-9 and anything after that and URL ends in “/”



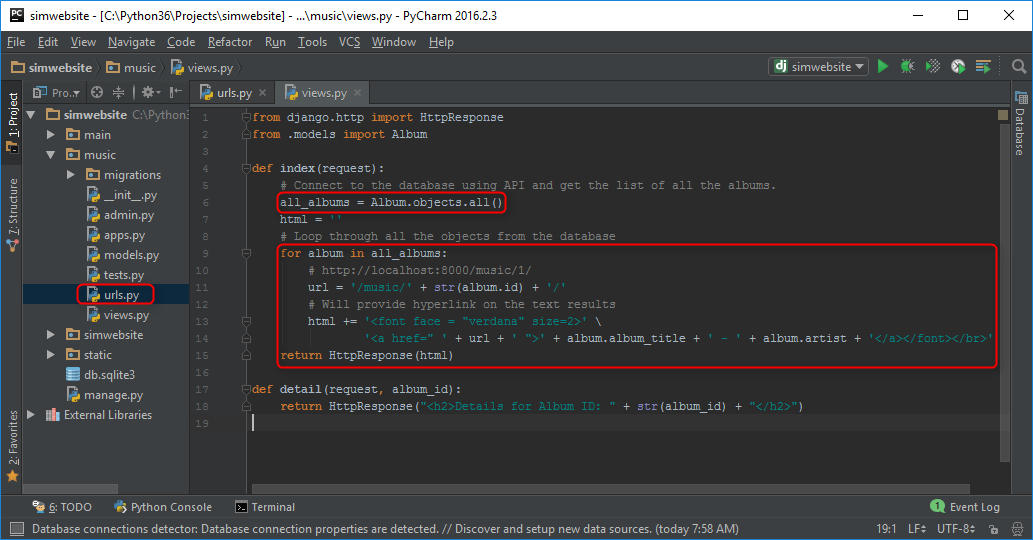
1. Edit views.py to load an html page whenever url pattern match an album id. Add a function that returns httpresponse html page when user request a url that match on the urls.py patterns.



**Connecting to the Database**

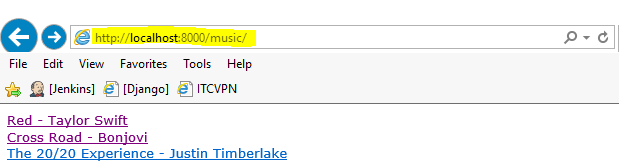
Once you have the URLs and Views setup, then we are now ready to connect to the database.

1. Edit “views.py” and add “all\_albums” variable where you can pull all objects stored on the database.
2. Add “Album.objects.all” to connect to Object Relational Mapper (ORM). ORM then query the database.

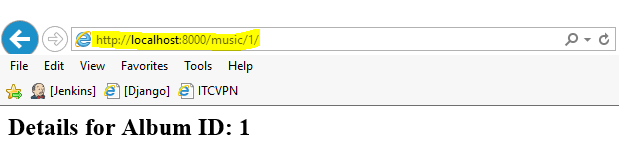


**Important**: If you’re developing a site for production, it is good practice to separate all your HTML code from python functions. This will be easy to understand and easy to manage.

1. Add a loop to go through all the objects stored in Album table and display the title & artist on the page.
2. Provide a hyper link that link to the ID of the record displayed ex: <http://localhost:8000/music/3/>



1. If you click the record then it will open the details of that Album

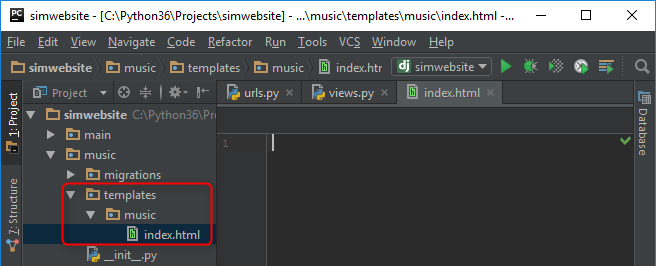


**Creating Templates**

Process Flow : Take a template 🡪 Load it 🡪 Rendered it plain html 🡪 Return it to the user.

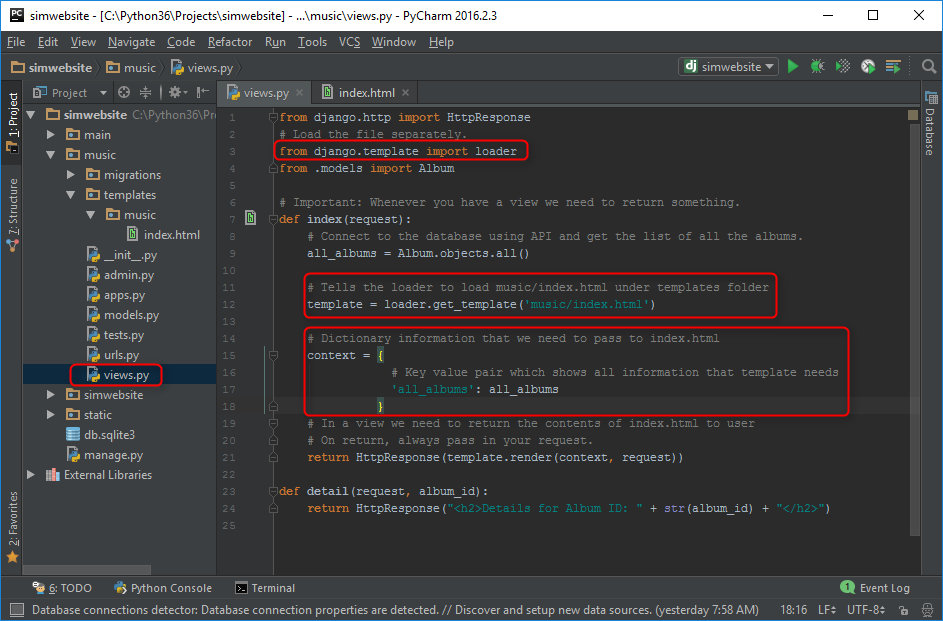
Generates html page and it allows us to separate html tags than python code.

1. First, we need to create a folder called “template” under the current app.
2. Second, create the same folder name as your apps under “template” ex: music
3. Under template 🡪 music, create all necessary html files here.

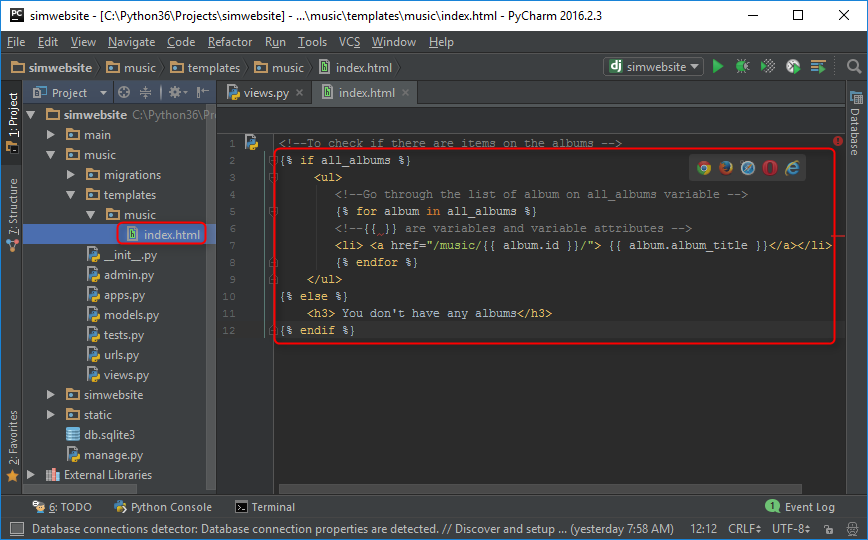


1. Next, is we need to import Django template loader under current app -> views.py

import django.template import loader



1. Add a variable and tell it to load /music/index.html template.
2. Add a dictionary variable named context that will display all albums.
3. Plugin variable created on “views.py” to “index.html”. Edit “index.html” and loop & show all albums



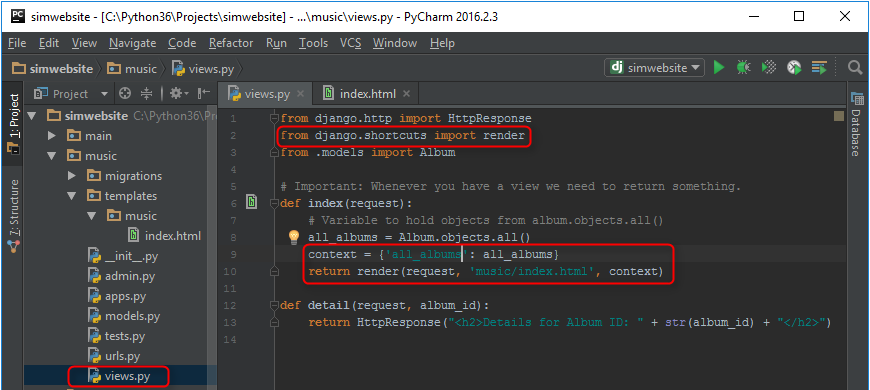
**Render Templates Shortcut**

This will combine render and load function to clean up your code.

1. Edit “views.py” and replace the import statement for loader with render.

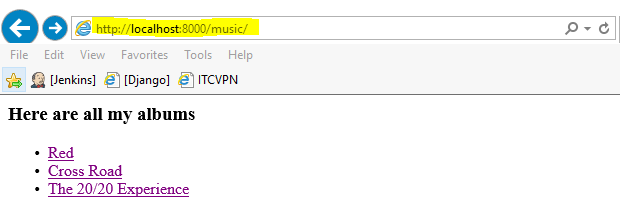
"from django.template import loader" replace with "from django.shortcuts import render"

1. Make context dictionary variable into a single line ex: context = {‘all\_albums’: all\_albums}
2. Change the return statement of the function “return render” and add a parameter for request, context & index.html location.



**Note**: The return HttpResponse is now built-in to return render

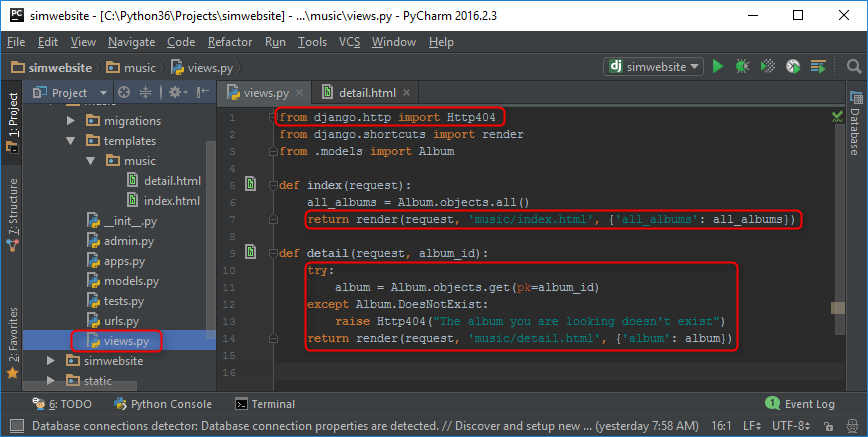
1. Add a header text on index.html to see the changes. Reload <http://localhost:8000/music/>



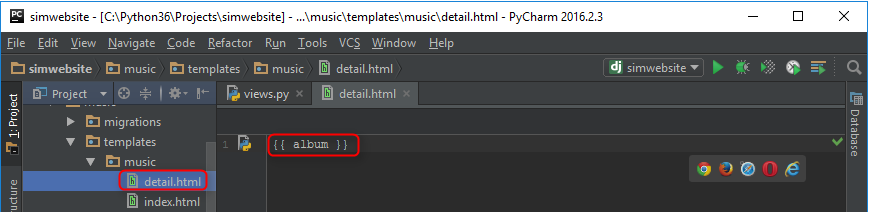
**Raising HTTP 404**

This is how to raise a page cannot be found if the user go to a link that is not existing.

1. Edit “views.py” and import the module “from django.http import Http404”.
2. Add a new html file under templates 🡪 appname 🡪 detail.html that will display the HTTP 404 message.
3. Edit the “details” function under “views.py” and add code to query the database and see if the id is existing under album. If yes, then we need to display the records under “detail.html” page. If not, then we need to display http 404.



1. Add {{ album }} on detail.html to show the album details when you click the link.



Next lesson - <https://www.youtube.com/watch?v=TnU_I8DKKYQ>