Rest API

Representational state transfer

Reading JSON File

```
import json

with open('apod.json', 'r') as f:
        json_text = f.read()

# Decode the JSON string into a Python dictionary.
apod_dict = json.loads(json_text)
print(apod_dict['explanation'])

# Encode the Python dictionary into a JSON string.
new_json_string = json.dumps(apod_dict, indent=4)
print(new_json_string)
```

Reading JSON File

```
import simplejson as json

with open('apod.json', 'r') as f:
        json_text = f.read()

# Decode the JSON string into a Python dictionary.
apod_dict = json.loads(json_text)
print(apod_dict['explanation'])

# Encode the Python dictionary into a JSON string.
new_json_string = json.dumps(apod_dict, indent=4)
print(new_json_string)
```

Reading JSON File

```
import ujson as json

with open('apod.json', 'r') as f:
        json_text = f.read()

# Decode the JSON string into a Python dictionary.
apod_dict = json.loads(json_text)
print(apod_dict['explanation'])

# Encode the Python dictionary into a JSON string.
new_json_string = json.dumps(apod_dict, indent=4)
print(new_json_string)
```

Introduction

- To get data for various resources.
- have you ever thought, where does this data come from? Well, it’s the servers from where we get the data

Example

- Consider a scenario where you are using the Amazon app. Now, obviously, this application needs a lot of Input data, as the data present in the application is never static. Either it is product offers getting released on a daily or monthly basis, or various Festivals showing different interest and category products like time deal, day deal etc.
- It’s never static which implies to the fact that data is always changing in these applications.

Where to get the Data?

- Well, this data is received from the Server or most commonly known as a Web-server. So, the client requests the server for the required information, via an API, and then, the server sends a response to the client.
- Over here, the response sent to the client is in the form of an HTML Web Page. But, do you think this is an apt response that you would expect when you send a request?
- Well, I am assuming the fact that you would say NO. Since, you would prefer the data to be returned in the form of structured format, rather than the complete Web page
- the data returned by the server, in response to the request of the client is either in the format of JSON or XML. Both JSON and XML format have a proper hierarchical structure of data.
- The REST API creates an object, and thereafter send the values of an object in response to the client.

What is REST API?

- REST suggests to create an object of the data requested by the client and send the values of the object in response to the user. For example, if the user is requesting for a item in Bangalore at a certain place and time, then you can create an object on the server side.
- you have an object and you are sending the state of an object. This is why REST is known as Representational State Transfer.
- *Representational State Transfer a.k.a REST* is an architectural style as well as an approach for communications purpose that is often used in various web services development.

REST

- The architectural style of REST helps in leveraging the lesser use of bandwidth to make an application more suitable for the internet. It is often regarded as the "*language of the internet*" and is completely based on the resources.
- To understand better, let's dive a little deeper and see how exactly does a REST API work. Basically, the REST API breaks down a transaction in order to create small modules. Now, each of these modules is used to address a specific part of the transaction. This approach provides more flexibility but requires a lot of effort to be built from the very scratch.

Principles of REST API

- Well, there are six ground principles laid down by Dr. Fielding who was the one to define the REST API design in 2000. Below are the six guiding principles of REST:
    1. Stateless
    2. Client-Server
    3. Uniform Interface
    4. Cacheable
    5. Layered system
    6. Code on demand

Stateless

- The requests sent from a client to a server will contain all the required information to make the server understand the requests sent from the client.
- This can be either a part of URL, query-string parameters, body, or even headers. The URL is used to uniquely identify the resource and the body holds the state of the requesting resource. Once the server processes the request, a response is sent to the client through body, status or headers

client-server

- The client-server architecture enables a uniform interface and separates clients from the servers. This enhances the portability across multiple platforms as well as the scalability of the server components.

Uniform Interface

- To obtain the uniformity throughout the application, REST has the following four interface constraints:
  - Resource identification
  - Resource Manipulation using representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state

Cacheable

- In order to provide a better performance, the applications are often made cacheable.
- This is done by labeling the response from the server as cacheable or non-cacheable either implicitly or explicitly.
- If the response is defined as cacheable, then the client cache can reuse the response data for equivalent responses in the future.

Layered system

- The layered system architecture allows an application to be more stable by limiting component behavior.
- This type of architecture helps in enhancing the application’s security as components in each layer cannot interact beyond the next immediate layer they are in.
- Also, it enables load balancing and provides shared caches for promoting scalability.

Code on demand

- This is an optional constraint and is used the least. It permits a clients code or applets to be downloaded and to be used within the application. In essence, it simplifies the clients by creating a smart application which doesn’t rely on its own code structure.

Method of REST API

- When I say CRUD operations, I mean that we create a resource, read a resource, update a resource and delete a resource. Now, to do these actions, you can actually use the HTTP methods, which are nothing but the REST API Methods.

C → Create → POST

R → Read → GET

U → Update → PUT

D → Delete → DELETE

HTTP Request

Building Hello world in Flask

```
C:\Users\USER1>pip install Flask
Requirement already satisfied: Flask in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (1.1.1)
Requirement already satisfied: Werkzeug>=0.15 in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (from Flask) (0.16.0)
Requirement already satisfied: Jinja2>=2.10.1 in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (from Flask) (2.10.1)
Requirement already satisfied: itsdangerous>=0.24 in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (from Flask) (1.1.0)
Requirement already satisfied: click>=5.1 in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (from Flask) (7.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\user1\appdata\local\programs\python\python37\lib\site-packages (from Jinja2>=2.10.1->Flask) (1.1.1)
```

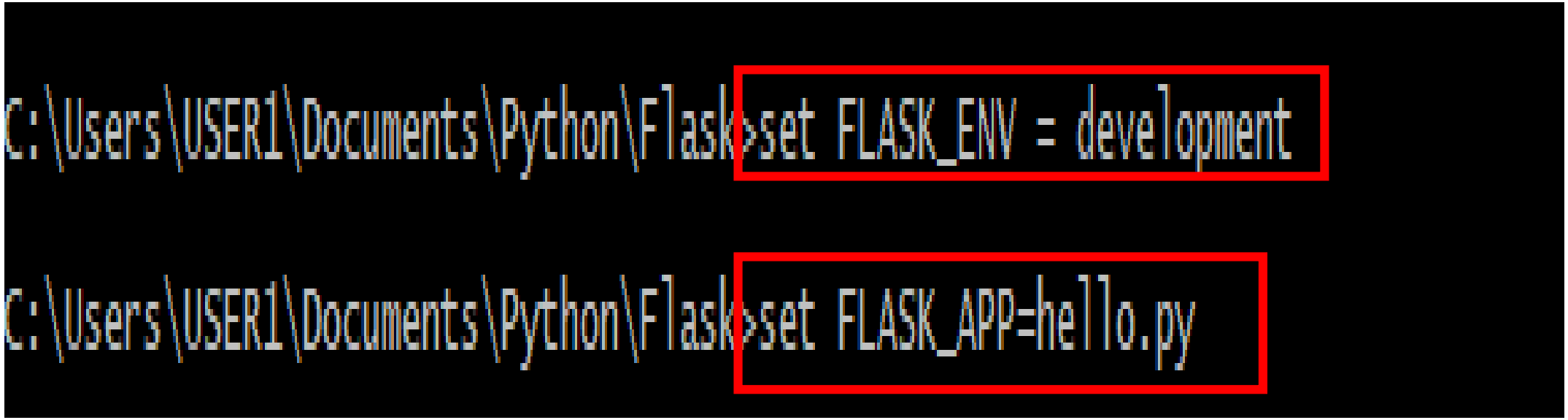
Creating the First Flask App

hello.py

hello.py

```
1   from flask import Flask
2   app = Flask(__name__)
3
4
5   @app.route("/")
6   def hello_world():
7     return "Hello, World!"
```

Setting up flask Environment



Running Flask

```
C:\Users\USER1\Documents\Python\Flask>flask run
 * Serving Flask app "hello.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [18/Jan/2020 20:01:53] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [18/Jan/2020 20:01:54] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Jan/2020 20:01:58] "GET /hello HTTP/1.1" 200 -
127.0.0.1 - - [18/Jan/2020 20:02:07] "GET /hello HTTP/1.1" 200 -
```

Building a REST API


```
{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    }
  ]
}
```


Output

hello.py

hello.py > ...

```python
from flask import Flask, jsonify
app = Flask(__name__)
tasks = [
    {
        'id': 1,
        'title': u'Buy groceries',
        'description': u'Milk, Cheese, Pizza, Fruit, Tylenol',
        'done': False
    },
    {
        'id': 2,
        'title': u'Learn Python',
        'description': u'Need to find a good Python tutorial on the web',
        'done': False
    }
]
@app.route('/todo/api/v1.0/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})
app.run(debug=True)

```

Building a REST API

http://localhost:5000/todo/api/v1.0/tasks/2

```
{
  "task": {
    "description": "Need to find a good Python tutorial on the web",
    "done": false,
    "id": 2,
    "title": "Learn Python"
  }
}
```

hello.py

hello.py > ...

```
29
30  from flask import abort
31
32  @app.route('/todo/api/v1.0/tasks/<int:task_id>', methods=['GET'])
33  def get_task(task_id):
34      task = [task for task in tasks if task['id'] == task_id]
35      if len(task) == 0:
36          abort(404)
37      return jsonify({'task': task[0]})
38  if __name__ == '__main__':
39    app.run(debug=True)
```

Building a REST API

Error Handing

hello.py

hello.py > not_found

```
40  from flask import make_response
41
42  @app.errorhandler(404)
43  def not_found(error):
44      return make_response(jsonify({'error': 'Not found'}), 404)
```

Run POST Command

```
curl -i -H "Content-Type: application/json" -X POST -d "{""""title"""":""""Read a book""""}"
http://localhost:5000/todo/api/v1.0/tasks
```

hello.py

hello.py > create_task

```
44
45  from flask import request
46
47  @app.route('/todo/api/v1.0/tasks', methods=['POST'])
48  def create_task():
49      if not request.json or not 'title' in request.json:
50          abort(400)
51      task = {
52          'id': tasks[-1]['id'] + 1,
53          'title': request.json['title'],
54          'description': request.json.get('description', ""),
55          'done': False
56      }
57      tasks.append(task)
58      return jsonify({'task': task}), 201
59  if __name__ == '__main__':
60      app.run(debug=True)
```

PUT

```
curl -i -H "Content-Type: application/json" -X PUT -d '{"done":true}' http://localhost:5000/todo/api/v1.0/tasks/2
```

```python
@app.route('/todo/api/v1.0/tasks/<int:task_id>', methods=['PUT'])
def update_task(task_id):
    task = [task for task in tasks if task['id'] == task_id]
    if len(task) == 0:
        abort(404)
    if not request.json:
        abort(400)
    if 'title' in request.json and type(request.json['title']) != unicode:
        abort(400)
    if 'description' in request.json and type(request.json['description']) is not unicode:
        abort(400)
    if 'done' in request.json and type(request.json['done']) is not bool:
        abort(400)
    task[0]['title'] = request.json.get('title', task[0]['title'])
    task[0]['description'] = request.json.get('description', task[0]['description'])
    task[0]['done'] = request.json.get('done', task[0]['done'])
    return jsonify({'task': task[0]})
```

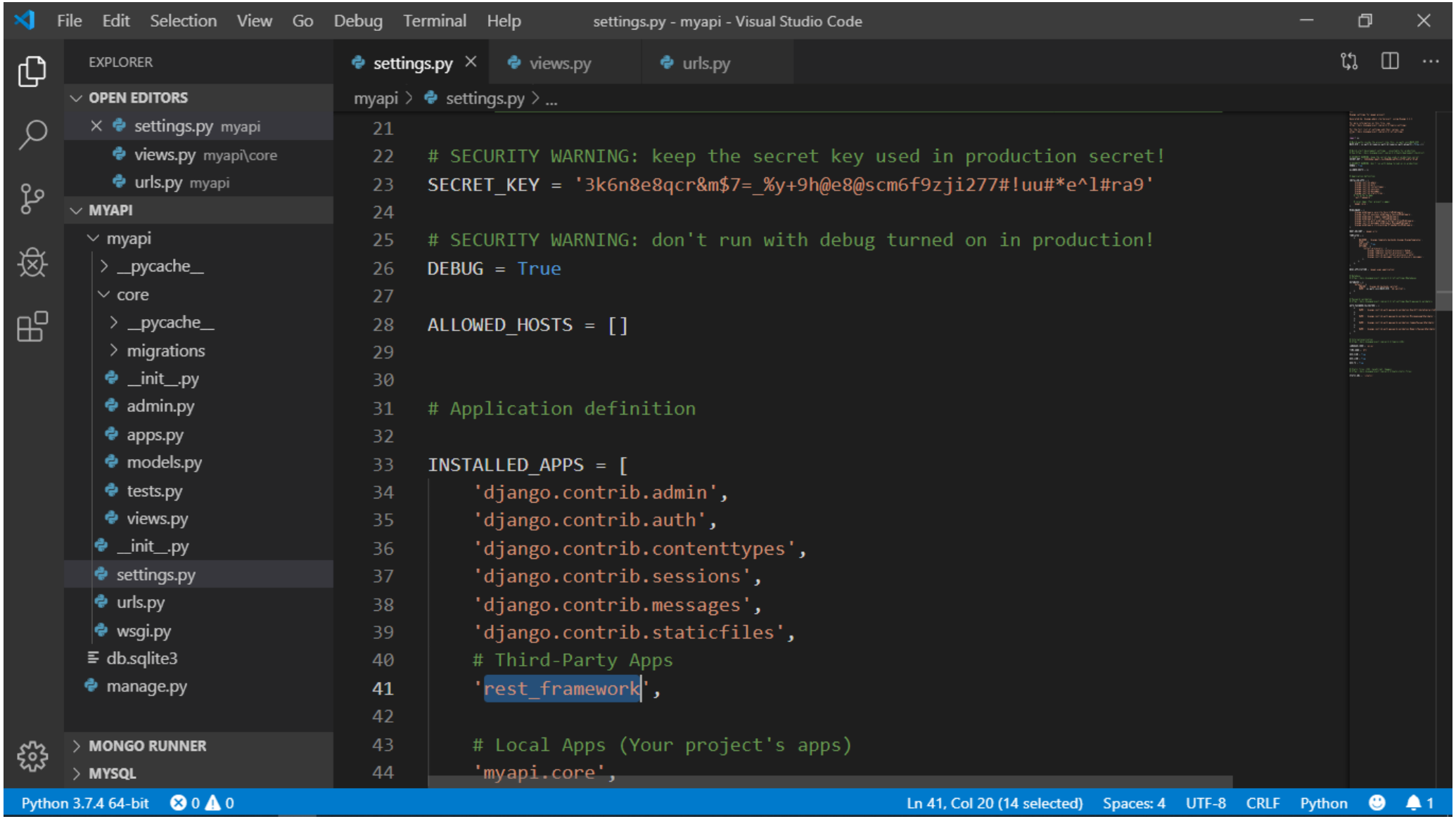
DELETE

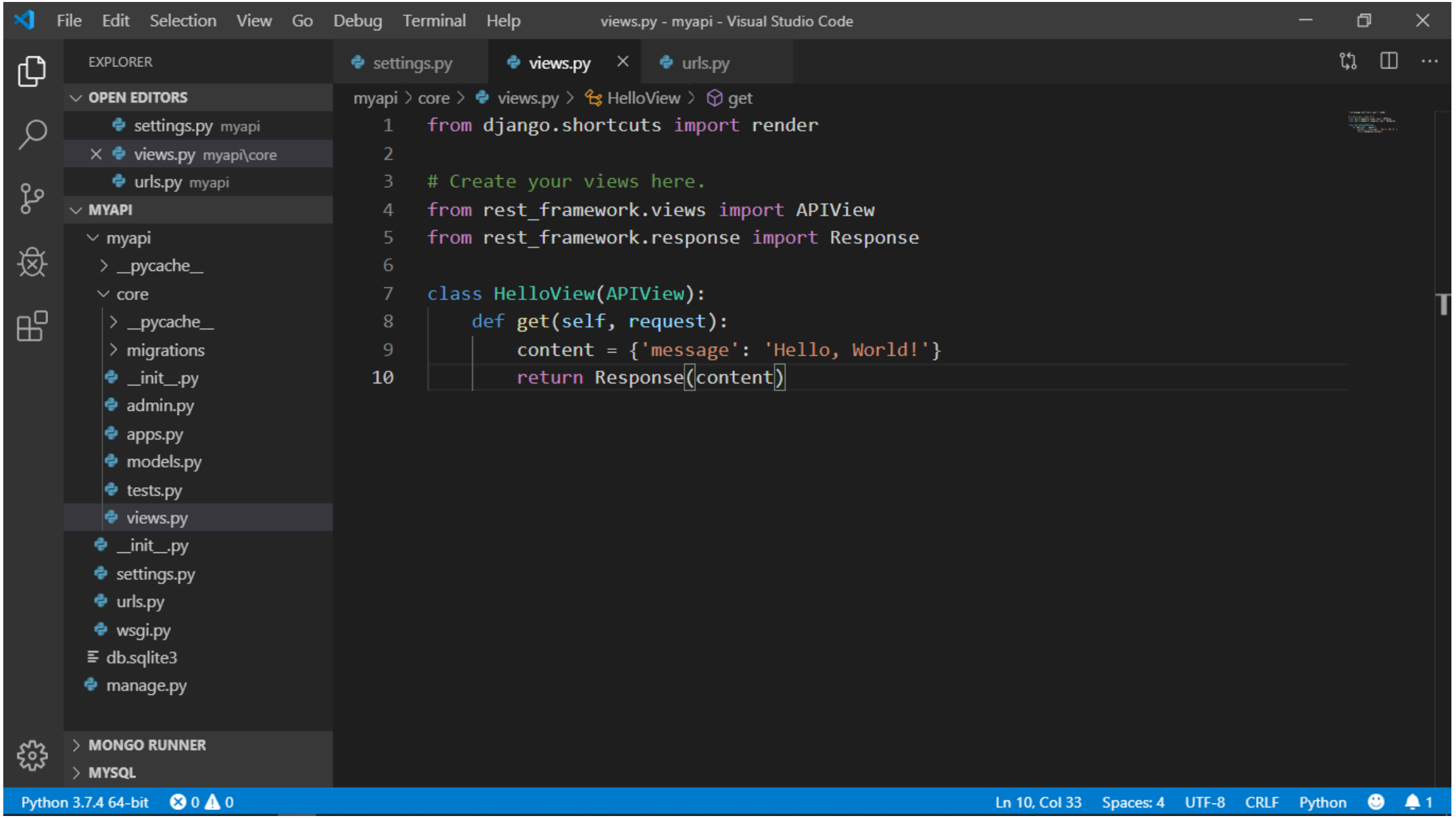
hello.py × sample.py

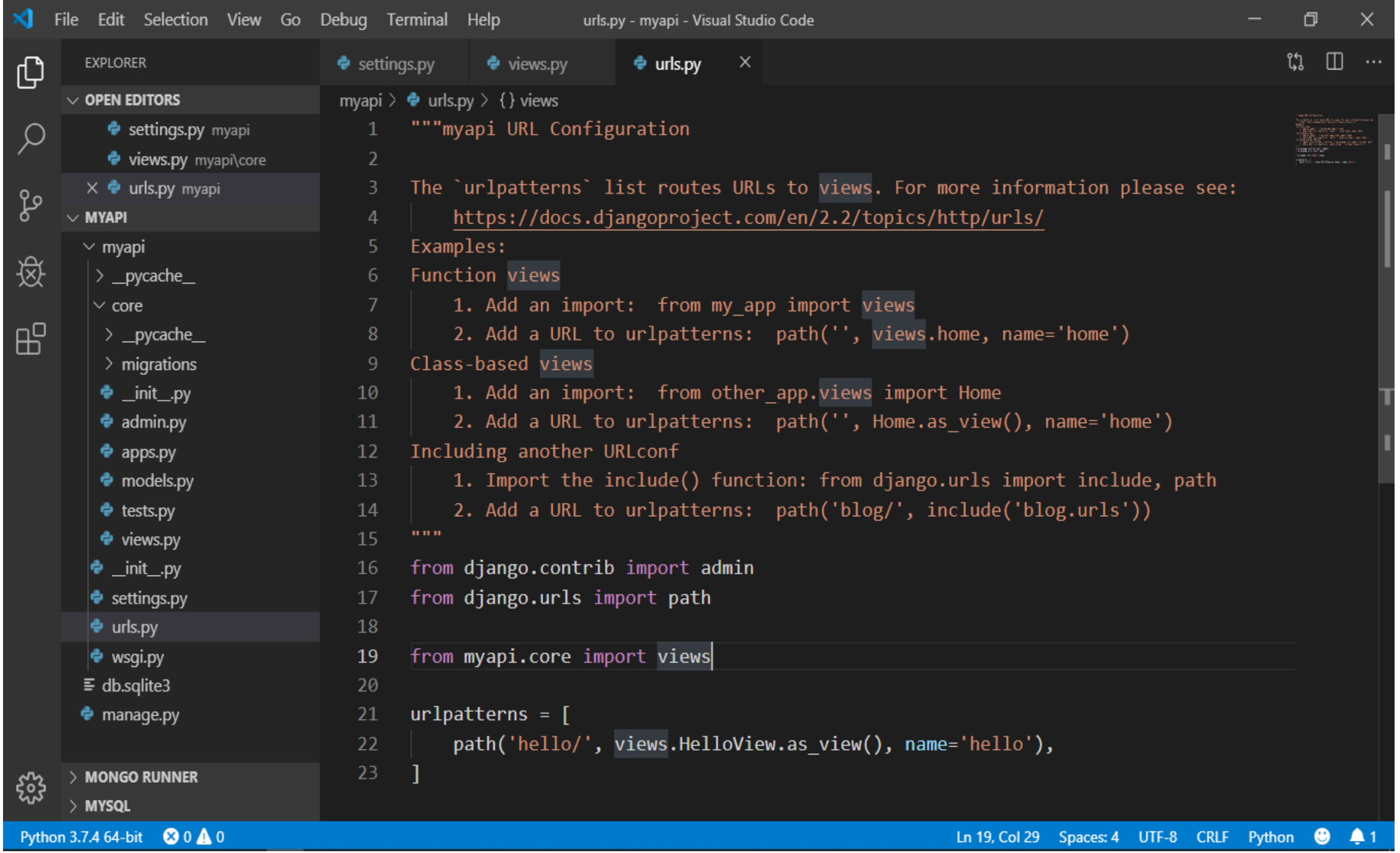
hello.py > update_task

```
63      if 'description' in request.json and type(request.json['description']) is not unicode:
64          abort(400)
65      if 'done' in request.json and type(request.json['done']) is not bool:
66          abort(400)
67      task[0]['title'] = request.json.get('title', task[0]['title'])
68      task[0]['description'] = request.json.get('description', task[0]['description'])
69      task[0]['done'] = request.json.get('done', task[0]['done'])
70      return jsonify({'task': task[0]})
71
72  @app.route('/todo/api/v1.0/tasks/<int:task_id>', methods=['DELETE'])
73  def delete_task(task_id):
74      task = [task for task in tasks if task['id'] == task_id]
75      if len(task) == 0:
76          abort(404)
77      tasks.remove(task[0])
78      return jsonify({'result': True})
79  if __name__ == '__main__':
80
81    app.run(debug=True)
```

Implementation of REST API With Django

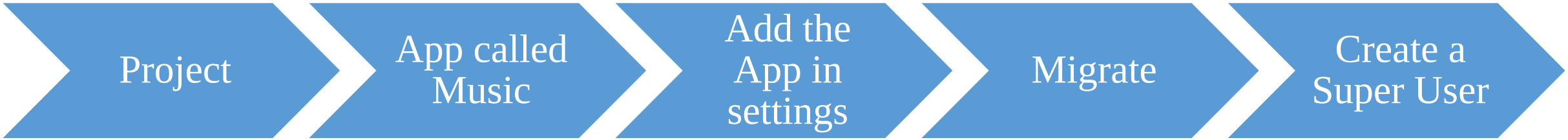

File Edit Selection View Go Debug Terminal Help
settings.py - myapi - Visual Studio Code
EXPLORER
OPEN EDITORS
settings.py myapi
views.py myapi\core
urls.py myapi
MYAPI
myapi
__pycache__
core
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
__init__.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
MONGO RUNNER
MYSQL
settings.py
views.py
urls.py
myapi > settings.py > ...
21
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = '3k6n8e8qcr&m$7=_%y+9h@e8@scm6f9zji277#!uu#*e^l#ra9'
24
25 # SECURITY WARNING: don't run with debug turned on in production!
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34 'django.contrib.admin',
35 'django.contrib.auth',
36 'django.contrib.contenttypes',
37 'django.contrib.sessions',
38 'django.contrib.messages',
39 'django.contrib.staticfiles',
40 # Third-Party Apps
41 'rest_framework',
42
43 # Local Apps (Your project's apps)
44 'myapi.core',
Python 3.7.4 64-bit
0 0
Ln 41, Col 20 (14 selected)
Spaces: 4
UTF-8
CRLF
Python
1


File Edit Selection View Go Debug Terminal Help
views.py - myapi - Visual Studio Code
EXPLORER
settings.py
views.py
urls.py
myapi > core > views.py > HelloView > get
OPEN EDITORS
settings.py myapi
views.py myapi\core
urls.py myapi
MYAPI
myapi
__pycache__
core
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
__init__.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
MONGO RUNNER
MYSQL
1 from django.shortcuts import render
2
3 # Create your views here.
4 from rest_framework.views import APIView
5 from rest_framework.response import Response
6
7 class HelloView(APIView):
8 def get(self, request):
9 content = {'message': 'Hello, World!'}
10 return Response(content)
Python 3.7.4 64-bit
0 0
Ln 10, Col 33
Spaces: 4
UTF-8
CRLF
Python
1

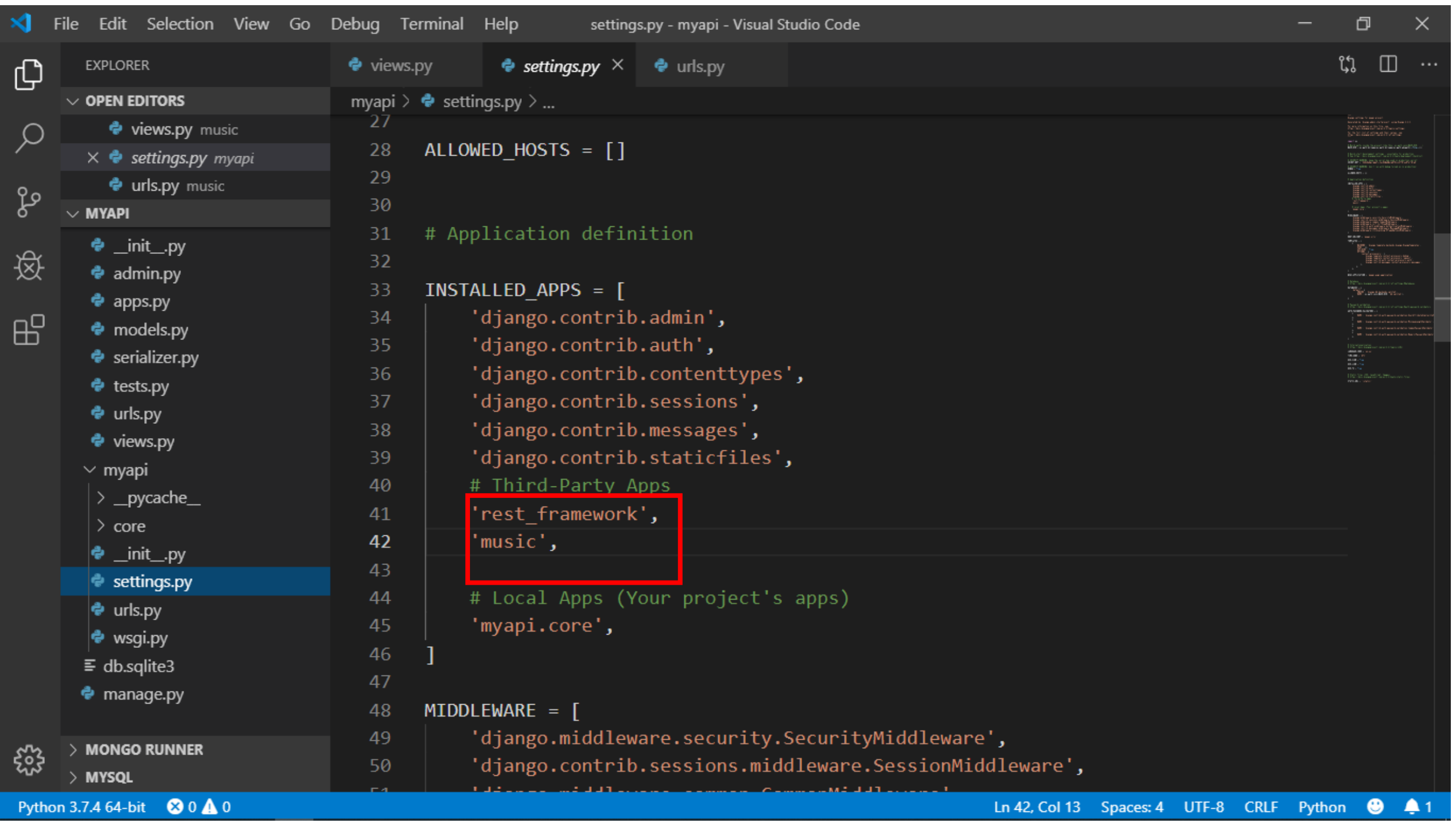

File Edit Selection View Go Debug Terminal Help
urls.py - myapi - Visual Studio Code
EXPLORER
settings.py
views.py
urls.py
myapi > urls.py > { } views
OPEN EDITORS
settings.py myapi
views.py myapi\core
urls.py myapi
MYAPI
myapi
__pycache__
core
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
__init__.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
MONGO RUNNER
MYSQL
"""myapi URL Configuration
The `urlpatterns` list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/2.2/topics/http/urls/
Examples:
Function views
1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from myapi.core import views
urlpatterns = [
path('hello/', views.HelloView.as_view(), name='hello'),
]
Python 3.7.4 64-bit
0 0
Ln 19, Col 29
Spaces: 4
UTF-8
CRLF
Python
1

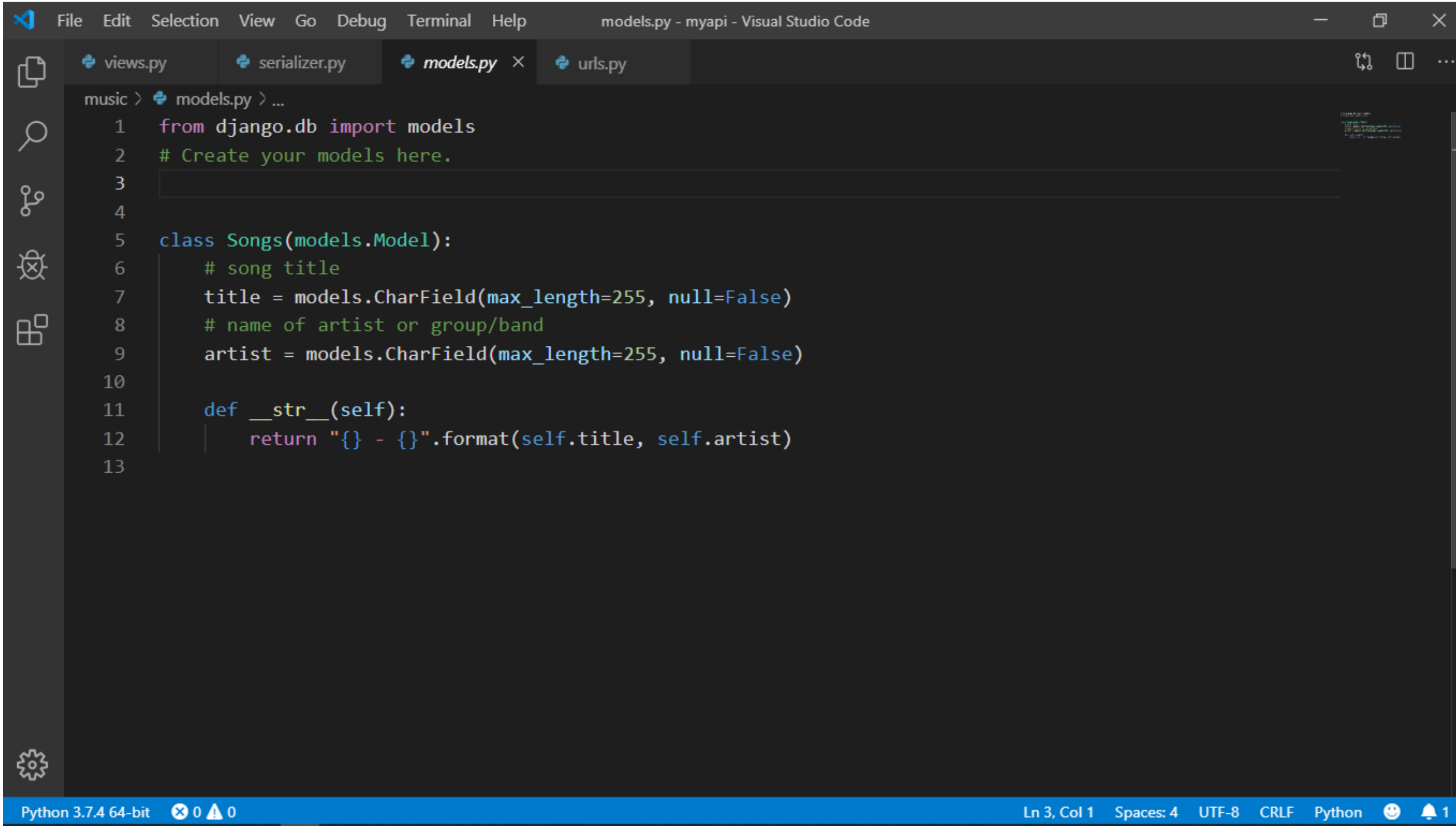
Implementation of Django

First Step


Project
App called Music
Add the App in settings
Migrate
Create a Super User

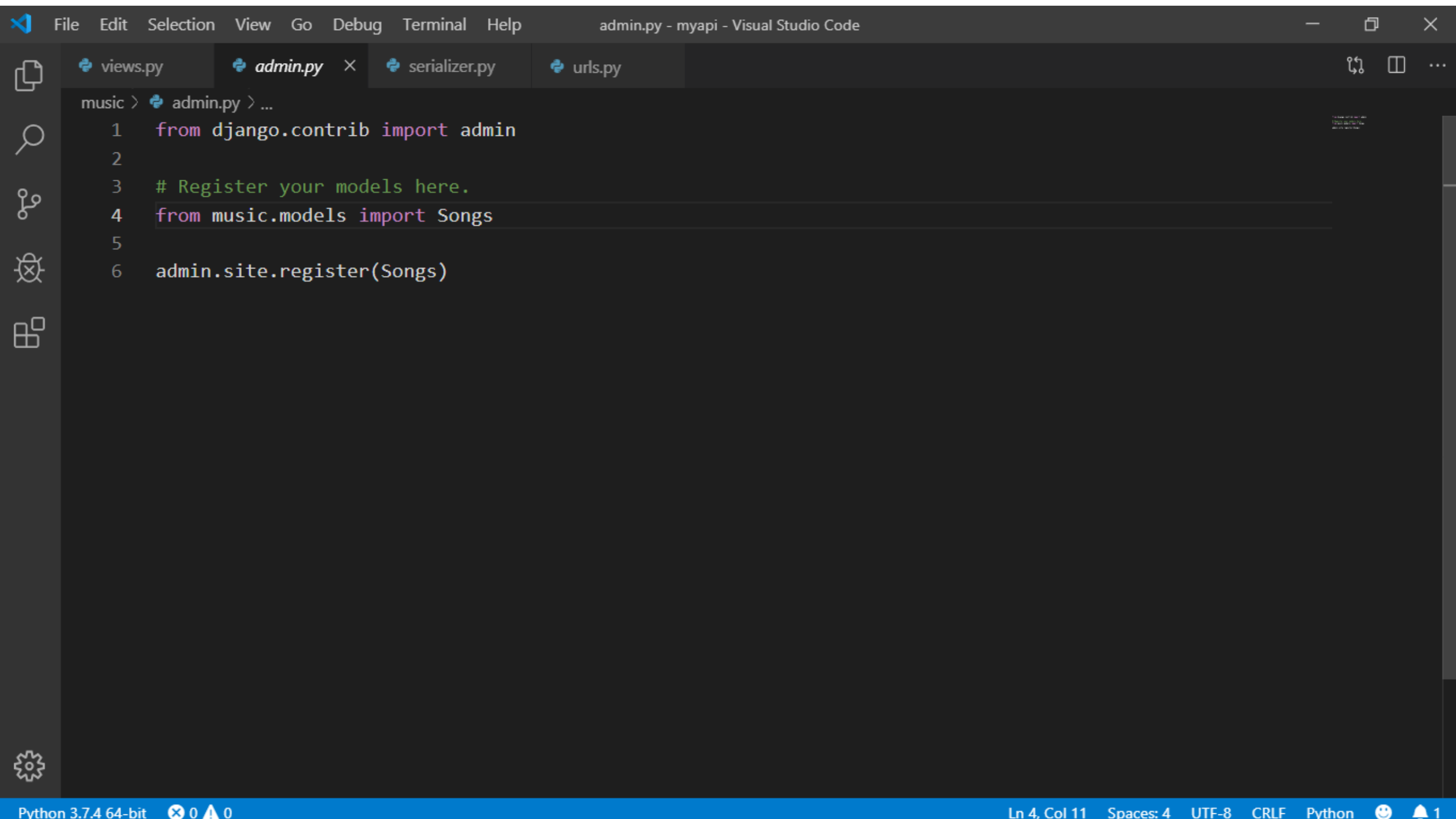
Steps to follow in your App


File Edit Selection View Go Debug Terminal Help
settings.py - myapi - Visual Studio Code
EXPLORER
views.py
settings.py
urls.py
myapi > settings.py > ...
OPEN EDITORS
views.py music
settings.py myapi
urls.py music
MYAPI
__init__.py
admin.py
apps.py
models.py
serializer.py
tests.py
urls.py
views.py
myapi
__pycache__
core
__init__.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
MONGO RUNNER
MYSQL
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34 'django.contrib.admin',
35 'django.contrib.auth',
36 'django.contrib.contenttypes',
37 'django.contrib.sessions',
38 'django.contrib.messages',
39 'django.contrib.staticfiles',
40 # Third-Party Apps
41 'rest_framework',
42 'music',
43
44 # Local Apps (Your project's apps)
45 'myapi.core',
46]
47
48 MIDDLEWARE = [
49 'django.middleware.security.SecurityMiddleware',
50 'django.contrib.sessions.middleware.SessionMiddleware',
Python 3.7.4 64-bit
0 0
Ln 42, Col 13
Spaces: 4
UTF-8
CRLF
Python
1


File Edit Selection View Go Debug Terminal Help
models.py - myapi - Visual Studio Code
views.py
serializer.py
models.py
urls.py
music > models.py > ...
1 from django.db import models
2 # Create your models here.
3
4
5 class Songs(models.Model):
6 # song title
7 title = models.CharField(max_length=255, null=False)
8 # name of artist or group/band
9 artist = models.CharField(max_length=255, null=False)
10
11 def __str__(self):
12 return "{} - {}".format(self.title, self.artist)
13
Python 3.7.4 64-bit
0 0
Ln 3, Col 1
Spaces: 4
UTF-8
CRLF
Python
1

```python
from rest_framework import serializers
from music.models import Songs


class SongsSerializer(serializers.ModelSerializer):
    class Meta:
        model = Songs
        fields = ("title", "artist")
```


File Edit Selection View Go Debug Terminal Help
admin.py - myapi - Visual Studio Code
views.py
admin.py
serializer.py
urls.py
music > admin.py > ...
1 from django.contrib import admin
2
3 # Register your models here.
4 from music.models import Songs
5
6 admin.site.register(Songs)
Python 3.7.4 64-bit
0 0
Ln 4, Col 11
Spaces: 4
UTF-8
CRLF
Python
1

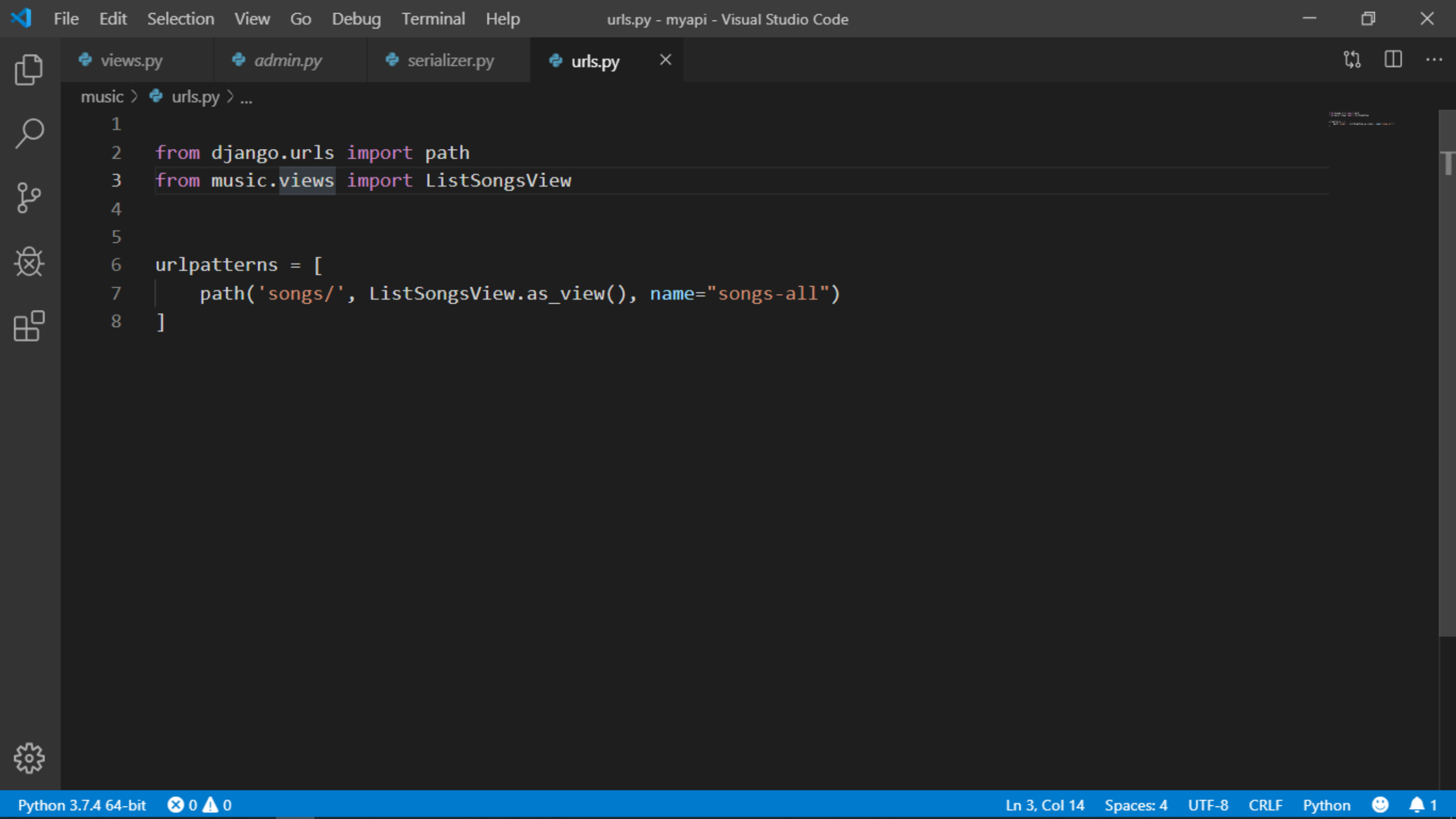
views.py | admin.py | serializer.py | urls.py

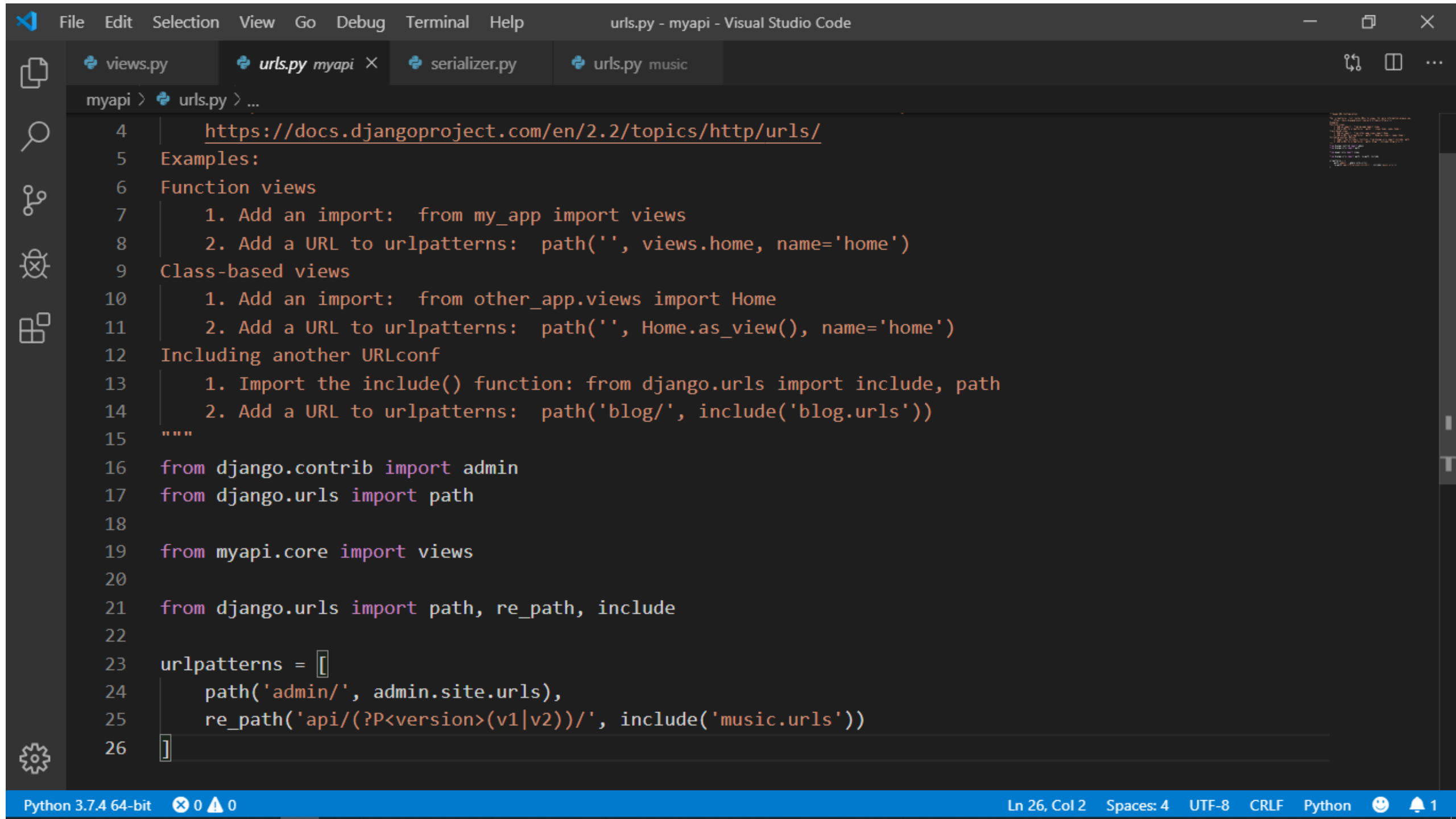
music > views.py > ListSongsView

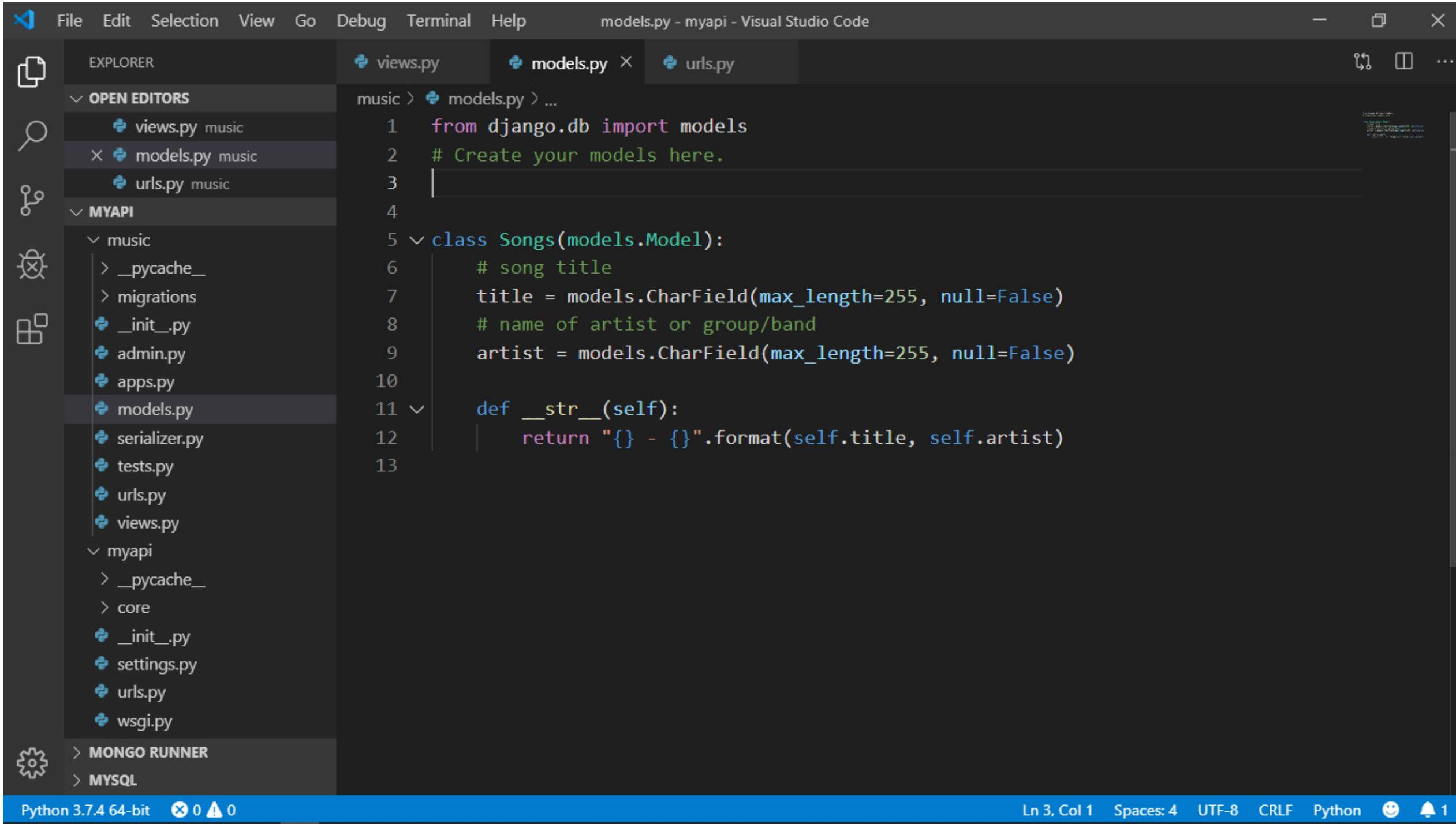
```python
from django.shortcuts import render

# Create your views here.
from rest_framework import generics
from music.models import Songs
from music.serializer import SongsSerializer


class ListSongsView(generics.ListAPIView):
    """
    Provides a get method handler.
    """
    queryset = Songs.objects.all()
    serializer_class = SongsSerializer
```


File Edit Selection View Go Debug Terminal Help
urls.py - myapi - Visual Studio Code
views.py
admin.py
serializer.py
urls.py
music > urls.py > ...
1
2 from django.urls import path
3 from music.views import ListSongsView
4
5
6 urlpatterns = [
7 path('songs/', ListSongsView.as_view(), name="songs-all")
8]
Python 3.7.4 64-bit
0 0
Ln 3, Col 14
Spaces: 4
UTF-8
CRLF
Python
1


File Edit Selection View Go Debug Terminal Help
urls.py - myapi - Visual Studio Code
views.py
urls.py myapi
serializer.py
urls.py music
myapi > urls.py > ...
https://docs.djangoproject.com/en/2.2/topics/http/urls/
Examples:
Function views
1. Add an import: from my_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
1. Add an import: from other_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from myapi.core import views
from django.urls import path, re_path, include
urlpatterns = [
path('admin/', admin.site.urls),
re_path('api/(?P<version>(v1|v2))/', include('music.urls'))
]
Python 3.7.4 64-bit
0 0
Ln 26, Col 2
Spaces: 4
UTF-8
CRLF
Python
1


File Edit Selection View Go Debug Terminal Help
models.py - myapi - Visual Studio Code
EXPLORER
views.py
models.py
urls.py
music > models.py > ...
OPEN EDITORS
views.py music
models.py music
urls.py music
MYAPI
music
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
serializer.py
tests.py
urls.py
views.py
myapi
__pycache__
core
__init__.py
settings.py
urls.py
wsgi.py
MONGO RUNNER
MYSQL
1 from django.db import models
2 # Create your models here.
3
4
5 class Songs(models.Model):
6 # song title
7 title = models.CharField(max_length=255, null=False)
8 # name of artist or group/band
9 artist = models.CharField(max_length=255, null=False)
10
11 def __str__(self):
12 return "{} - {}".format(self.title, self.artist)
13
Python 3.7.4 64-bit
0 0
Ln 3, Col 1
Spaces: 4
UTF-8
CRLF
Python
1

127.0.0.1:8000/api/v2/songs/

Django REST framework mahira

List Songs

List Songs

OPTIONS GET

Provides a get method handler.

```
GET /api/v2/songs/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "title": "On the way",
        "artist": "John"
    },
    {
        "title": "some",
        "artist": "some"
    }
]
```