SQLITE3 DATABASE TUTORIAL

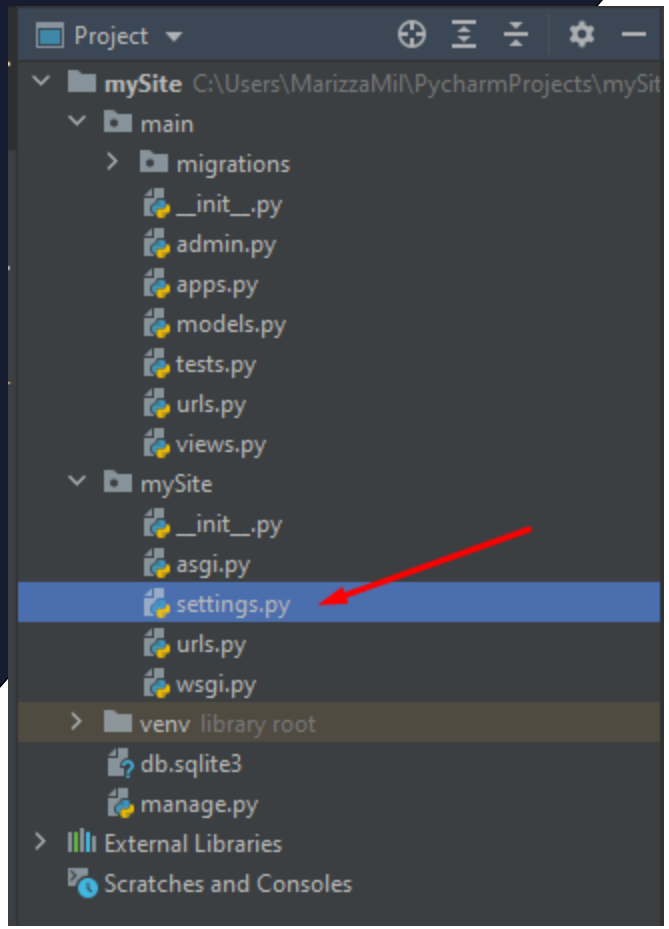
Database Setup

Almost all web applications use something called a database.

In this django tutorial you will learn how to use a simple database called SQLIte3.

MODIFYING SETTINGS.PY

The first step to setting up our database is to tell django that we have added an application to our project and that it requires some setup. To do this we need to navigate to the settings.py file from within our interior site directory and add the following line to the installed apps section: 'main.apps.MainConfig', If you've named your application something other than main you will need to replace "main" and "Main" with the name of your application.



 # settings.py file

```
# Application definition

INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',

'main.apps.MainConfig',

]
```

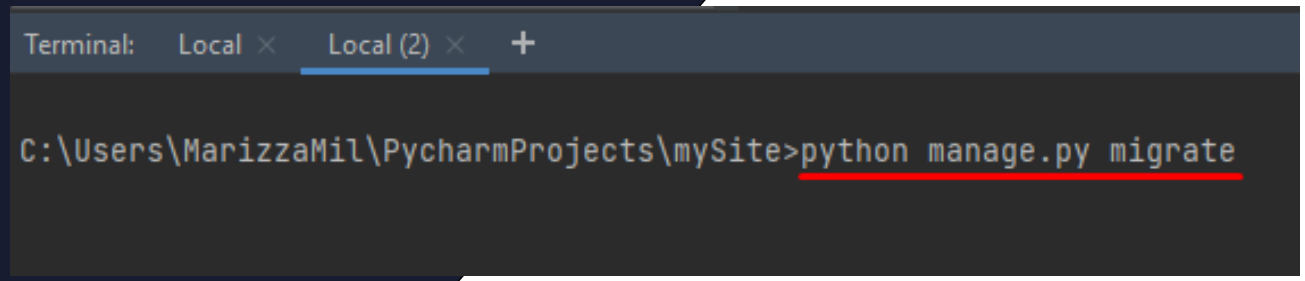
Add code

MIGRATIONS

Django has it's own version control system that is called migrations. Similar to GIT when you make a change that requires any new dependencies to be installed you need to tell django from the command line. Each change you make will be logged as a migration and can be viewed afterwards to allow you to revert to previous versions.

To tell django to start setting up our database use the following command (make sure you are in the directory containing manage.py).

$ python manage.py migrate



Now we should see an empty SQLite database in our directory.

DEFINING MODELS

Now that we have setup our database we need to define some models for storing information. To do this navigate to the models.py from inside our application folder. For this tutorial we will be creating a basic to-do list. This means we will need a model for a to-do list and for each of the items on our to-do list.

When we define a model we simply create a class that is the name of our model that inherits from models.Model. Then we define all of the fields or attributes of our model as class variables. We can also add methods to use on our models. We will create the two models as seen below:

For a full list of the different fields you may use see here.

 # models.py file

```
from django.db import models


class ToDoList(models.Model):
    name = models.CharField(max_length=200)

    def __str__(self):
    return self.name


class Item(models.Model):
    todolist = models.ForeignKey(ToDoList, on_delete=models.CASCADE)
    text = models.CharField(max_length=300)
    complete = models.BooleanField()

    def __str__(self):
    return self.text
```

MAKING MIGRATIONS

Now that we've updated our models file we need to tell django to make changes to our database. To do this we use the following command:
python manage.py makemigrations main.

If your app is named something else replace "main" with its name.

```
C:\Users\MarizzaMil\PycharmProjects\mySite>python manage.py makemigrations main
Migrations for 'main':
  main\migrations\0001_initial.py
    - Create model ToDoList
    - Create model Item
```

Finally to apply the migrations we use:
python manage.py migrate

```
C:\Users\MarizzaMil\PycharmProjects\mySite>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, main, sessions
Running migrations:
  Applying main.0001_initial... OK
```

WORKING WITH OUR DATABASE

Now that we've defined some models we can start adding information to our database. The commands shown below were executed in a python shell, however they can be used from your python files in the same/similar ways. Open python shell: python manage.py shell

```
(venv) C:\Users\MarizzaMil\PycharmProjects\mySite>python manage.py shell
Python 3.9.5 (tags/v3.9.5:0a7dcbd, May  3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

WORKING WITH OUR DATABASE

```
# import will be different depending on script location
>>> from main.models import Item, ToDoList

# create a ToDoList
>>> list1 = ToDoList(name="Itay's List")

# saves the ToDoList in the database
>>> list1.save()

# prints all of the ToDoLists in the database
>>> print(ToDoList.objects.all())
<QuerySet [<ToDoList: Itay's List>]>

# gets the ToDoList object(s) with name "Itay's List"
>>> find_list = ToDoList.objects.get(name="Itay's List")
```

WORKING WITH OUR DATABASE

Since we defined a relationship between Item and ToDoList each ToDoList has an "item_set"

```
# get all of the items on a ToDoList
>>> list1.item_set.all()
<QuerySet []>
```

```
# add an item to the ToDoList
>>> list1.item_set.create(text="Go to the mall", complete=False)
<Item: Go to the mall>
```

Danger zone. Commands that I didn't execute

```
# change the name of the list
    list1.name = "new name"
# save changes
    list1.save()
# delete the list
    list1.delete()
```

We used these commands from the command line to demonstrate how they work.

Typically you will just use these from your python scripts.

MODIFY FILE URLS.PY

 # main/urls.py file

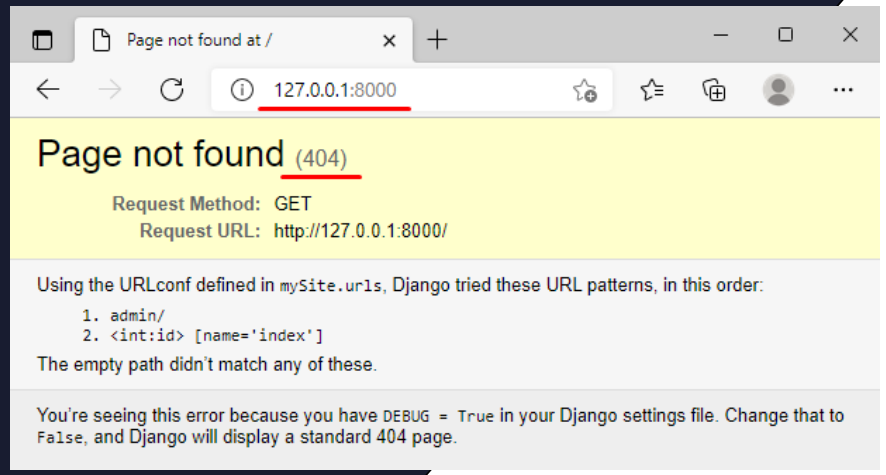
Add code

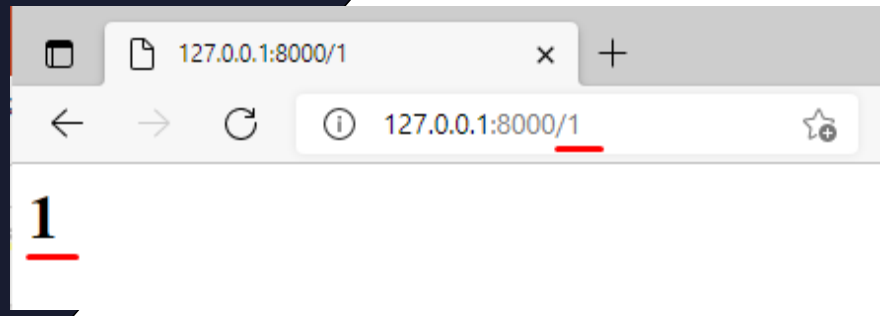
```
from django.urls import path
from . import views

urlpatterns = [

path('<int:id>', views.index, name='index'),

]
```

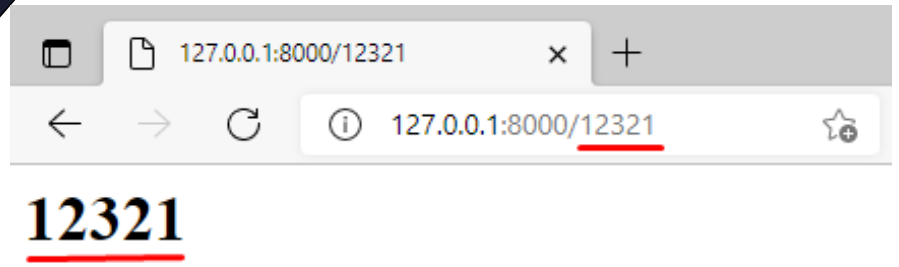
MODIFY FILE VIEWS.PY

Modify code

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request, id):
return HttpResponse("<h1>%s</h1>" % id)
```





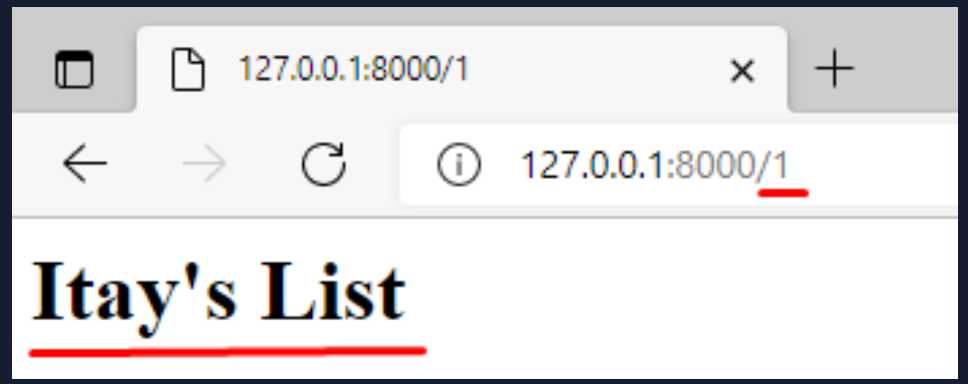


MODIFY FILE VIEWS.PY

```
from django.shortcuts import render
from django.http import HttpResponse

from .models import ToDoList, Item

def index(request, id):

    ls = ToDoList.objects.get(id=id)

    return HttpResponse("<h1>%s</h1>" % ls.name)
```

Add code

Modify code



MODIFY FILE URLS.PY

 # main/urls.py file

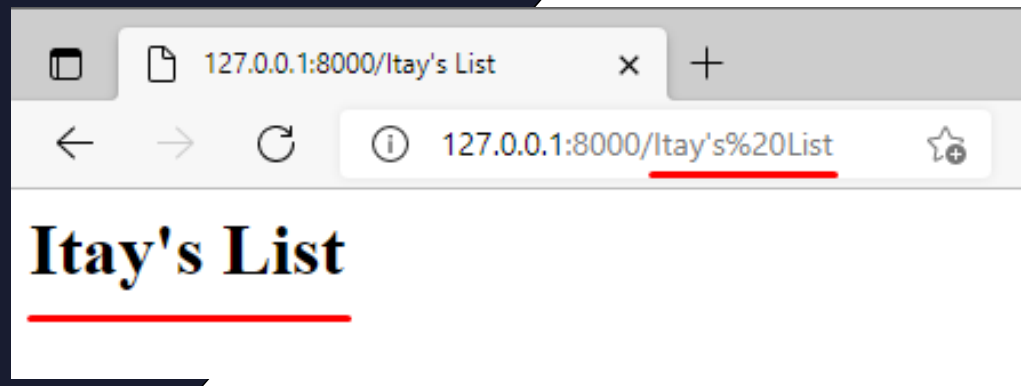
```
from django.urls import path
from . import views

urlpatterns = [

path('<str:name>', views.index, name='index'),
]
```

Modify code

MODIFY FILE VIEWS.PY

```
from django.shortcuts import render
from django.http import HttpResponse
from .models import ToDoList, Item

def index(request, name):
ls = ToDoList.objects.get(name=name)
return HttpResponse("<h1>%s</h1>" % ls.name)
```

Modify code



MODIFY FILE VIEWS.PY

```
from django.shortcuts import render
from django.http import HttpResponse
from .models import ToDoList, Item

def index(request, name):
ls = ToDoList.objects.get(name=name)

item = ls.item_set.get(id=1)

return HttpResponse("<h1>%s</h1><br></br><p>%s</p>" % (ls.name, item.text))
```

Add code

Modify code