By Sir RED

Python - Django Framework for Web Development Part 2

Django with MySql Database

Requirements:

1. **MySql Server (e.g. WAMP Server)**
   http://wampserver.aviatechno.net/

2. **MySql Connector Python**

   Download MySQL Connector for Python and install:

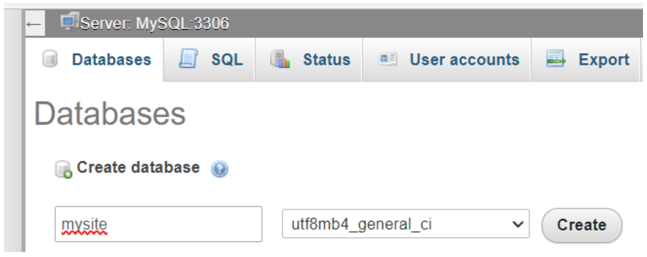
   https://dev.mysql.com/downloads/connector/python/

3. **MySqlClient:**

   Open command prompt or terminal and run this command to install MySqlClient:

   pip install mysqlclient

4. **Create Database**

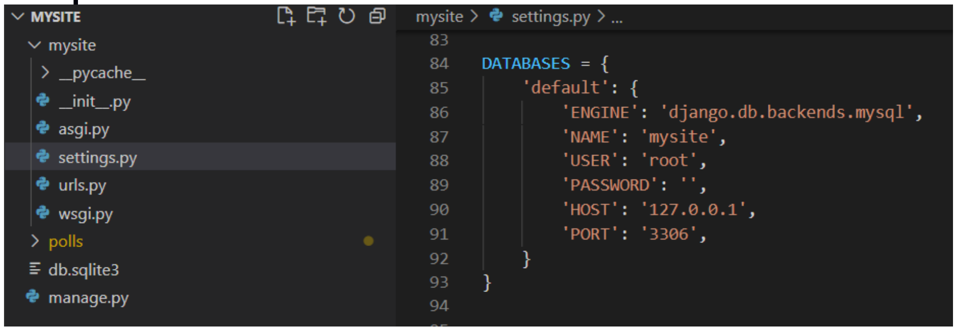


Django Project Database Configuration

1. Open **mysite/settings.py** file and find the **DATABASES = {**`'NAME': BASE_DIR / 'db.sqlite3',`**}**
2. Replace it with the following MySQL Drivers:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite', #database name
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

Sample:



```python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

If you wish to use another database, install the appropriate database bindings and change the following keys in the **DATABASES 'default'** item to match your database connection settings:

- **ENGINE** – Either **'django.db.backends.sqlite3'**, **'django.db.backends.postgresql'**, **'django.db.backends.mysql'**, or **'django.db.backends.oracle'**. Other backends are also available.
- **NAME** – The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, **NAME** should be the full absolute path, including filename, of that file. The default value, **BASE_DIR / 'db.sqlite3'**, will store the file in your project directory.

If you are not using SQLite as your database, additional settings such as **USER**, **PASSWORD**, and **HOST** must be added. For more details, see the reference documentation for **DATABASES**.

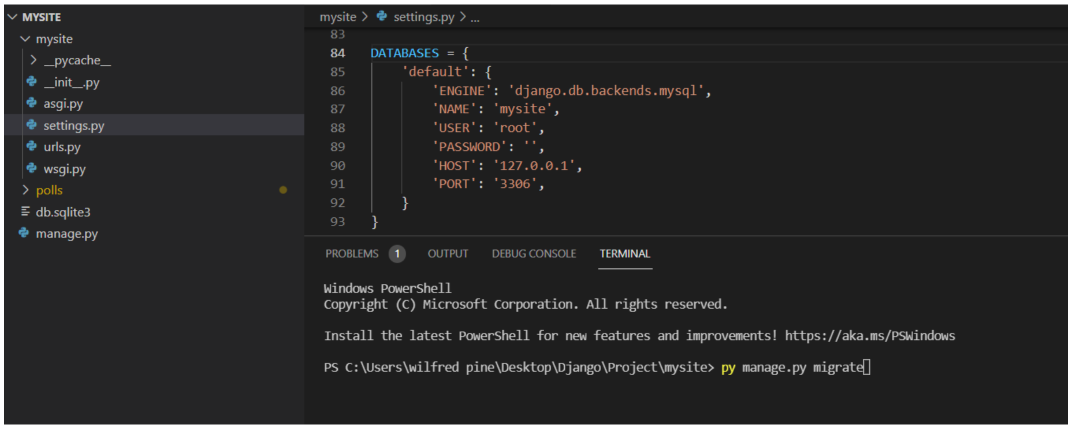
Also, note the **INSTALLED_APPS** setting at the top of the file. That holds the names of all Django applications that are activated in this Django instance. Apps can be used in multiple projects, and you can package and distribute them for use by others in their projects.
By default, **INSTALLED_APPS** contains the following apps, all of which come with Django:

- **django.contrib.admin** – The admin site. You'll use it shortly.
- **django.contrib.auth** – An authentication system.
- **django.contrib.contenttypes** – A framework for content types.
- **django.contrib.sessions** – A session framework.
- **django.contrib.messages** – A messaging framework.
- **django.contrib.staticfiles** – A framework for managing static files.

These applications are included by default as a convenience for the common case.
Some of these applications make use of at least one database table, though, so we need to create the tables in the database before we can use them. To do that, run the following command:

```
py manage.py migrate
```

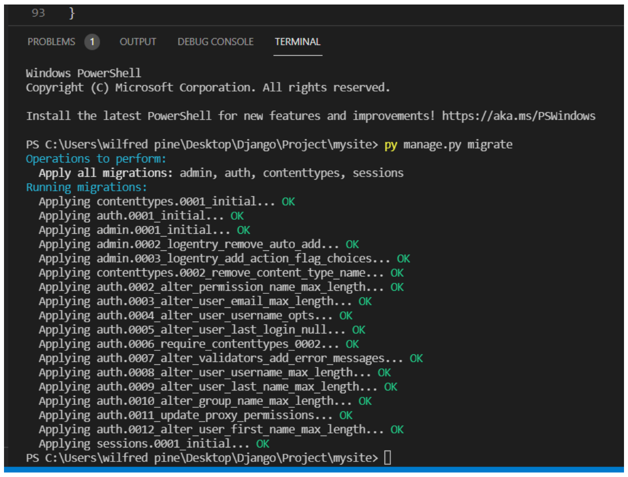
The **migrate** command looks at the **INSTALLED_APPS** setting and creates any necessary database tables according to the database settings in your **mysite/settings.py** file and the database migrations shipped with the app.



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mysite',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> py manage.py migrate
```

After **migrate** command:



```
93    }
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> py manage.py migrate
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
PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> []
```

Ten (10) tables form INSTALLED_APPS added to your database. (6 tables from Authentication apps/system)

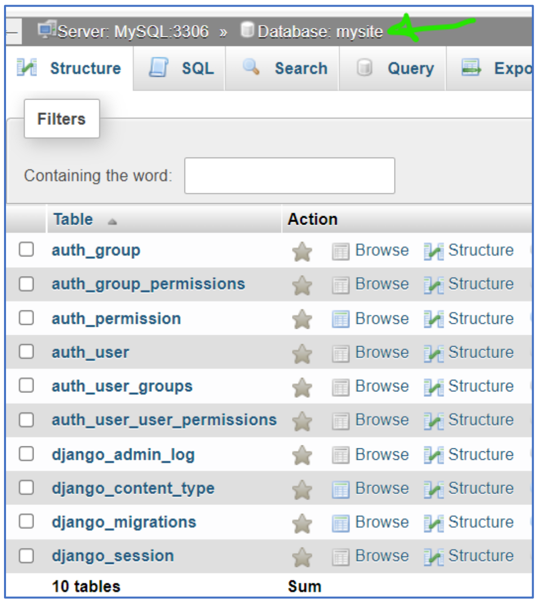


Table	Action		
auth_group	☆	Browse	Structure
auth_group_permissions	☆	Browse	Structure
auth_permission	☆	Browse	Structure
auth_user	☆	Browse	Structure
auth_user_groups	☆	Browse	Structure
auth_user_user_permissions	☆	Browse	Structure
django_admin_log	☆	Browse	Structure
django_content_type	☆	Browse	Structure
django_migrations	☆	Browse	Structure
django_session	☆	Browse	Structure
10 tables	Sum		

Models

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.

The basics:

- Each model is a Python class that subclasses **django.db.models.Model**.
- Each attribute of the model represents a database field.
- With all of this, Django gives you an automatically generated database-access API

Quick Example:

This example model defines a **Person**, which has a **first_name** and **last_name**:

```
from django.db import models

class Person(models.Model):

    first_name = models.CharField(max_length=30)

    last_name = models.CharField(max_length=30)
```

first_name and **last_name** are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column.

The above **Person** model would create a database table like this:

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

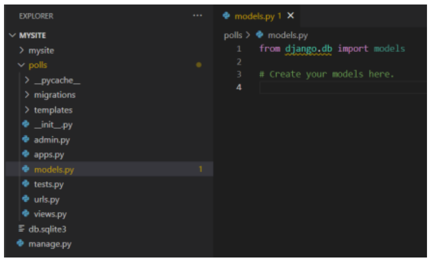
Creating model of your apps

Now we'll define your models – essentially, your database layout, with additional metadata.

In our poll app, we'll create two models: **Question** and **Choice**. A **Question** has a question and a publication date. A **Choice** has two fields: the text of the choice and a vote tally. Each **Choice** is associated with a **Question**.

These concepts are represented by Python classes. Edit the **polls/models.py** file so it looks like this:

1. Open the app **“polls/models.py”**



```
from django.db import models

# Create your models here.

```

2. Add the following code for creating table “question”:

```
class Question(models.Model):
    question_id = models.AutoField(primary_key=True)
    question_text = models.CharField(max_length=254)
    pub_date = models.DateTimeField('date published')
```

3. Creating table “choice”

```
class Choice(models.Model):
    choice_id = models.AutoField(primary_key=True)
    question_id = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Here, each model is represented by a class that subclasses **django.db.models.Model**. Each model has a number of class variables, each of which represents a database field in the model.

Each field is represented by an instance of a **Field** class – e.g., **CharField** for character fields and **DateTimeField** for datetimes. This tells Django what type of data each field holds.

The name of each **Field** instance (e.g. **question_text** or **pub_date**) is the field’s name, in machine-friendly format. You’ll use this value in your Python code, and your database will use it as the column name.

You can use an optional first positional argument to a **Field** to designate a human-readable name. That’s used in a couple of introspective parts of Django, and it doubles as documentation. If this field isn’t provided, Django will use the machine-readable name. In this example, we’ve only defined a human-readable name for **Question.pub_date**. For all other fields in this model, the field’s machine-readable name will suffice as its human-readable name.

Some **Field** classes have required arguments. **CharField**, for example, requires that you give it a **max_length**. That’s used not only in the database schema, but in validation, as we’ll soon see.

A **Field** can also have various optional arguments; in this case, we’ve set the **default** value of **votes** to 0.

Finally, note a relationship is defined, using **ForeignKey**. That tells Django each **Choice** is related to a single **Question**. Django supports all the common database relationships: many-to-one, many-to-many, and one-to-one.

Model Manager

"A Manager is the interface through which database query operations are provided to Django models."

Advantages of model managers:

- Readability - when we use model managers, we reduce complexity in our model.
- Ease of maintenance - model managers make our model simple and readable which also leads to ease of maintenance in the long run.

1. **Example 1 (**using **model.Manager):**

   Assigning "**models.Manager**" into "**objects"** variable.

```
from django.db import models

# Create your models here.

# For Question Table
class Question(models.Model):
    question_id = models.AutoField(primary_key=True)
    question_text = models.CharField(max_length=254)
    pub_date = models.DateTimeField('date published')
    objects = models.Manager

# For Choice Table
class Choice(models.Model):
    choice_id = models.AutoField(primary_key=True)
    question_id = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    objects = models.Manager
```

2. **Example 2** (creating class manager of class **Question)**:

We created a class **QuestionManager** and created a validation method before it inserted to database. You can also add methods for your query sets or any SQL statements.

```
class QuestionManager(models.Manager):
    def validator(self, postData):
        errors = {}
        if len(postData['question_text']) > 254:
            errors['question_text'] = "Question text cannot be longer than 254 characters"
        return errors
```

To use the **QuestionManager**, we call and assign it to a variable "**objects**" inside **Question** Class:

```
class Question(models.Model):
    question_id = models.AutoField(primary_key=True)
    question_text = models.CharField(max_length=254)
    pub_date = models.DateTimeField('date published')
    objects = QuestionManager() #added/call question's manager
```

Notes: if you have no queryset inside QuestionManager class,
the **"objects = QuestionManager()"** is the same as **"objects = model.Manager"**.

3. **Example 3 (**queryset in **Manager** class**):**

```
class QuestionManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(pub_date='2022-02-04')

class Question(models.Model):
    question_id = models.AutoField(primary_key=True)
    question_text = models.CharField(max_length=254)
    pub_date = models.DateTimeField('date published')

    objects = models.Manager() #display all questions
    filtered_objects = QuestionManager() #display questions published at 2022-02-04
```

Activating Models

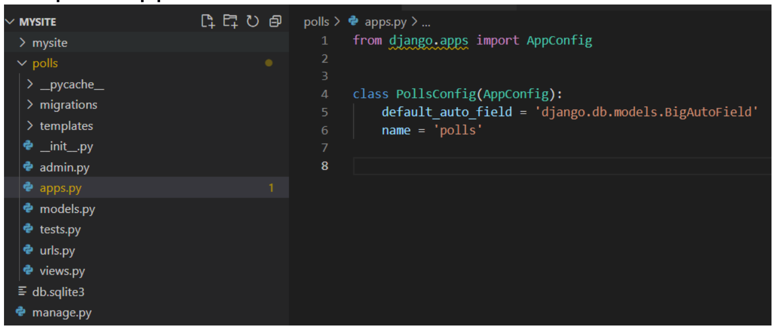
That small bit of model code gives Django a lot of information. With it, Django can:

- Create a database schema (**CREATE TABLE** statements) for this app.
- Create a Python database-access API for accessing **Question** and **Choice** objects.

But first we need to tell our project that the **polls** app is installed.

To include the app in our project, we need to add a reference to its configuration class in the **INSTALLED_APPS** setting.

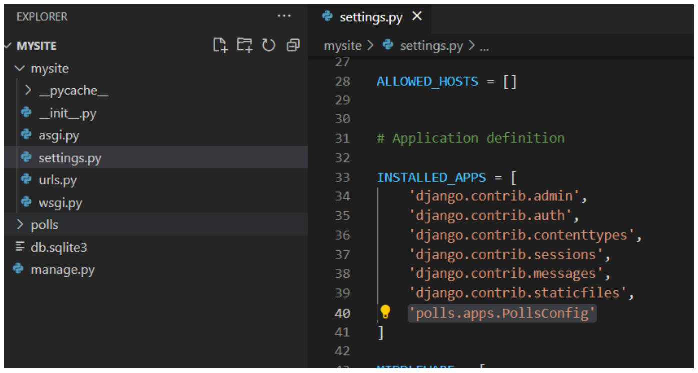
1. Open your "**polls/apps.py"** and you will see the config class (**PollsConfig**) of our polls apps.



```
from django.apps import AppConfig


class PollsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'polls'
```

The **PollsConfig** class is in the **polls/apps.py** file, so its dotted *path* is **'polls.apps.PollsConfig'**

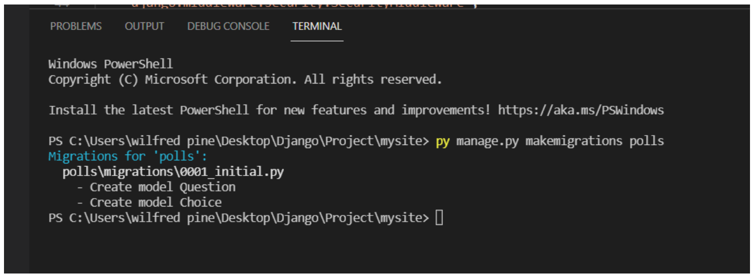
2. Open your “**mysite/settings.py”** inside **INSTALLED_APPS,** replace **'polls'** into **'polls.apps.PollsConfig'**



```
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'polls.apps.PollsConfig'
]
```

3. Now Django knows to include the **polls** app. Let’s run another command:

py manage.py makemigrations polls



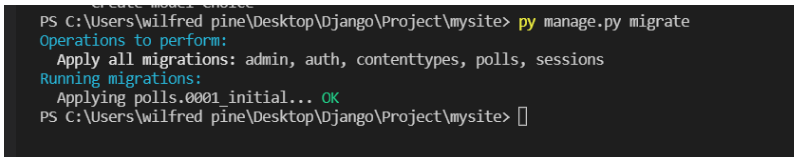
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> py manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    - Create model Question
    - Create model Choice
PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> 
```

By running **makemigrations**, you’re telling Django that you’ve made some changes to your models (in this case, you’ve made new ones) and that you’d like the changes to be stored as a *migration*.

Migrations are how Django stores changes to your models (and thus your database schema) - they’re files on disk.

4. **Migrate** Command

py manage.py migrate



```
PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK
PS C:\Users\wilfred pine\Desktop\Django\Project\mysite> 
```

The **Questions** & **Choices** in your polls' model will now added to your database.

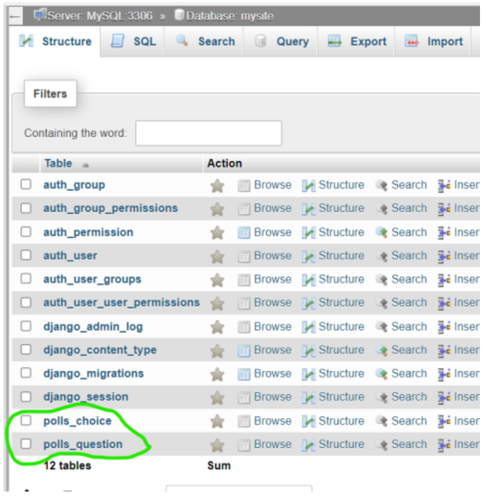
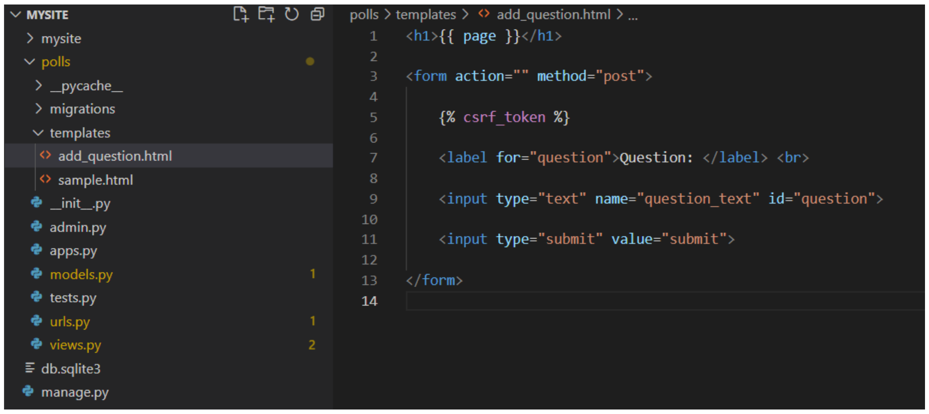


	Table	Action
☐	auth_group	Browse Structure Search Inser
☐	auth_group_permissions	Browse Structure Search Inser
☐	auth_permission	Browse Structure Search Inser
☐	auth_user	Browse Structure Search Inser
☐	auth_user_groups	Browse Structure Search Inser
☐	auth_user_user_permissions	Browse Structure Search Inser
☐	django_admin_log	Browse Structure Search Inser
☐	django_content_type	Browse Structure Search Inser
☐	django_migrations	Browse Structure Search Inser
☐	django_session	Browse Structure Search Inser
☐	polls_choice	Browse Structure Search Inser
☐	polls_question	Browse Structure Search Inser
	12 tables	Sum

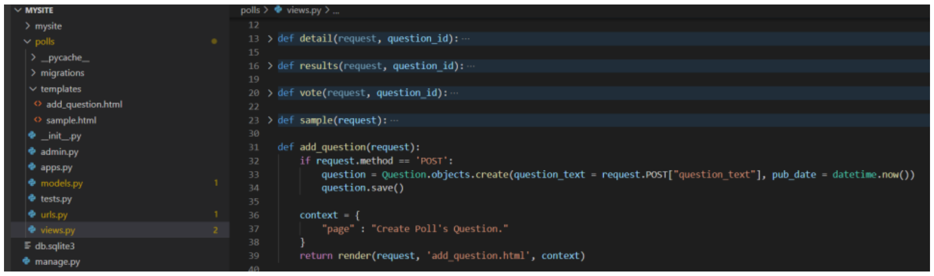
CRUD

1. Create a template for adding question.



```html
<h1>{{ page }}</h1>

<form action="" method="post">

    {% csrf_token %}

    <label for="question">Question: </label> <br>

    <input type="text" name="question_text" id="question">

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

</form>
```

2. Create a method at **views.py** and render the template.



```python
> def detail(request, question_id): ···

> def results(request, question_id): ···

> def vote(request, question_id): ···

> def sample(request): ···

def add_question(request):
    if request.method == 'POST':
        question = Question.objects.create(question_text = request.POST["question_text"], pub_date = datetime.now())
        question.save()

    context = {
        "page" : "Create Poll's Question."
    }
    return render(request, 'add_question.html', context)
```

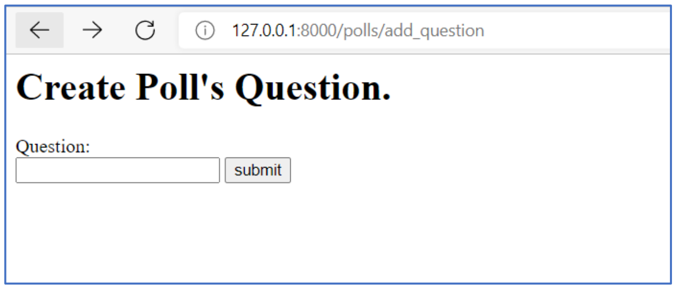
To use the model **question**, you need to import it in your views

```python
from polls.models import Question
```

3. Add path on your **polls/urls.py**

```
    path('add_question', views.add_question),
```

4. Run your project


127.0.0.1:8000/polls/add_question

Create Poll's Question.

Question:

Form Method

- **POST Method**

```
if request.method == 'POST':
    question_text = request.POST["question_text"]
    pub_date = datetime.now()
```

- **Save**

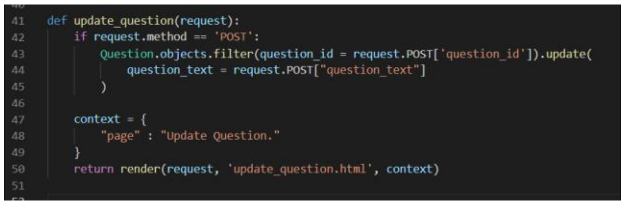
```
question = Question.objects.create(question_text = request.POST["question_text"], pub_date = datetime.now())
question.save()
```

To use the model **Question**, you need to import it in your views

- **Full Code**

```
from datetime import datetime
from django.http import HttpResponse
from django.shortcuts import render
from polls.models import Question

# Create your views here.
def add_question(request):
    if request.method == 'POST':
        question = Question.objects.create(question_text = request.POST["question_text"], pub_date = datetime.now())
        question.save()

    context = {
        "page" : "Create Poll's Question."
    }
    return render(request, 'add_question.html', context)
```

- **Update**



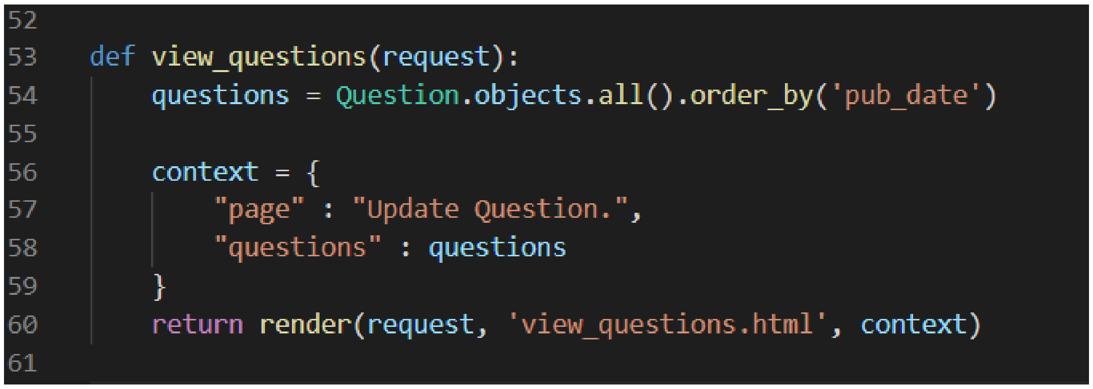
```
def update_question(request):
    if request.method == 'POST':
        Question.objects.filter(question_id = request.POST['question_id']).update(
            question_text = request.POST["question_text"]
        )

    context = {
        "page" : "Update Question."
    }
    return render(request, 'update_question.html', context)
```

- **Delete**

```
question = Question.objects.get(question_id = 25)
question.delete()
```

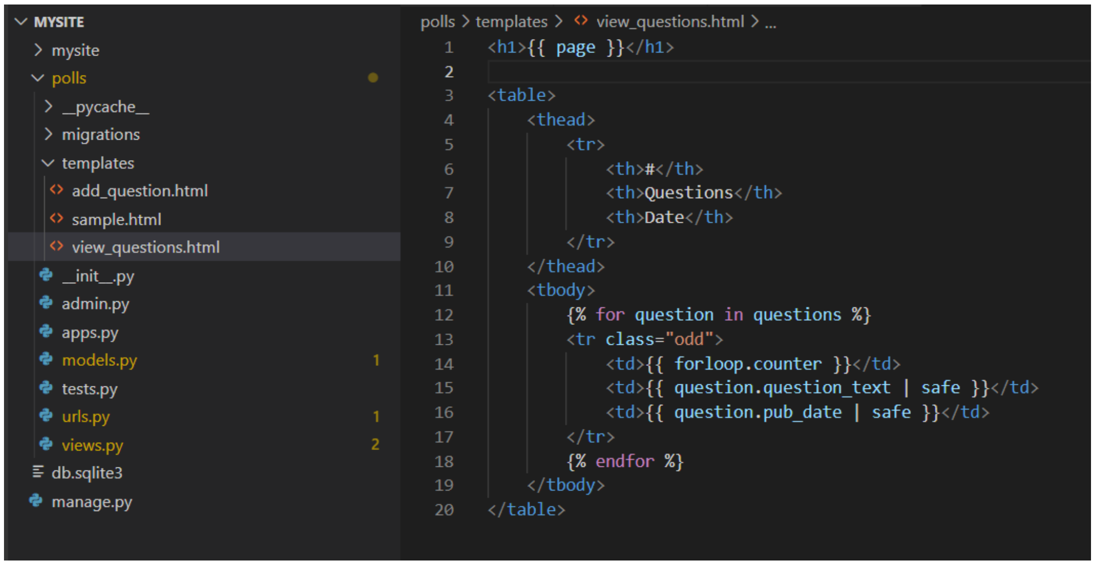
- **Select**

Views



```
def view_questions(request):
    questions = Question.objects.all().order_by('pub_date')

    context = {
        "page" : "Update Question.",
        "questions" : questions
    }
    return render(request, 'view_questions.html', context)
```

Template



```
<h1>{{ page }}</h1>

<table>
    <thead>
        <tr>
            <th>#</th>
            <th>Questions</th>
            <th>Date</th>
        </tr>
    </thead>
    <tbody>
        {% for question in questions %}
        <tr class="odd">
            <td>{{ forloop.counter }}</td>
            <td>{{ question.question_text | safe }}</td>
            <td>{{ question.pub_date | safe }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
```

REFERENCES:

https://docs.djangoproject.com/en/4.0/

https://learndjango.com/tutorials/django-best-practices-projects-vs-apps#:~:text=Apps%20A%20Django%20app%20is%20a%20small%20library,another%20app%20called%20payments%20to%20charge%20logged-in%20subscribers.