Hyperiondev

TASK

Django - Poll App

Visit our website

Introduction

WELCOME TO THE DJANGO - POLL APP TASK!

This task will serve as an introduction to creating a basic poll application. This will enable you to become more comfortable with writing Django views, models, and URLs. You may start this application in a separate project or continue in your mySite project.

Disclaimer: We've curated the most relevant bits of the official documentation for you and added some additional explanation to make Django as concise and accessible as possible.

CREATING THE POLLS APP

To create your app, make sure you're in the same directory as manage.py and type the following command:

```
> python manage.py startapp polls
```

This will create a directory called **polls/**, which is laid out like this:

- **polls/**
  - **migrations/**
    - **__init__.py**
  - **__init__.py**
  - **admin.py**
  - **apps.py**
  - **models.py**
  - **tests.py**
  - **views.py**

This directory structure will house the poll application. Don't forget to install your application and set up a URL for it as we have previously done. Moving along, open the file **polls/views.py** and put the following Python code in it:

```python
from django.http import HttpResponse

# Create your views here.
def index(request):
    return HttpResponse("Hello world. You're at the polls index.")
```

To call the view, we need to map it to a URL. Remember the procedure for mapping a URL to a view? Create a new file called **urls.py** in your polls directory and place the following code in it:

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

urlpatterns = [
    path('', views.index, name='index')
]
```

CREATING OUR MODELS

In our simple 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.

Recall the content covered in the SQL tasks. Let's review the important aspects:

- A relational database consists of a set of tables.
- Each table represents an object, contains a set of records, and requires a primary key for unique identification of each record.
- To model relations, we use foreign keys, which are references to primary keys in related tables.
- According to the rules of normalisation:
    - One-to-many relationships require foreign keys on the many side
    - Association tables need to be constructed for many-to-many relationships

One feature that has been built into Django is object-relational mapping (ORM). Because each entry in a SQL table represents a single object, this can be converted to a class instance in Python. Normally, the syntax for this is complex and requires some difficult conversions. ORM makes this conversion seamless, meaning that you interact minimally with SQL. You only need to define the relevant classes, and Django will handle the rest for you. Edit the **polls/models.py** file so that it looks like this:

```
from django.db import models

# Create your models here.
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

The code should be pretty straightforward since you have become somewhat familiar with creating models in the previous task. Each model is represented by a class that inherits from `django.db.models.Model`. Each model has a number of class variables, each of which represents a database field. Each field is represented by an instance of a Field class that tells Django what type of data each field holds – e.g., `CharField` for character fields and `DateTimeField` for datetimes.

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 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 also 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 that we want to store a relationship between our `Question` and `Choice` entities in our database. A `Question` can have more than one `Choice` associated with it. In databases we use foreign keys to indicate relationships between entities. In Django, we can use a `ForeignKey` field to show this many-to-one relationship.

Notice that in the `Choice` model, we have added a `ForeignKey` field (called `Question`). This tells Django that each `Choice` is related to a single `Question`. The `ForeignKey` field is always placed on the "many" side of a many-to-one relationship. The `ForeignKey` field always points to a primary key (which is a field used to uniquely identify a specific object that may be automatically generated by Django) of the object it is related to. In the example above, therefore, the `ForeignKey` in the `Choice` object points to the primary key of the `Question` that the `Choice` is related to. For more information about how many-to-one relationships are implemented with Django consult the **documentation about many-to-one relationships**, Django supports all the common database relationships: many-to-one, many-to-many, and one-to-one.

Instructions

Feel free at any point to refer back to the material if you get stuck. Remember that if you require more assistance, we are always here to help you!

Compulsory Task

Please complete the following to proceed to the next task:

- In our **polls/models.py** file, we haven't defined a `__str__` method as we have previously done. Define these methods for each class and return `question_text` and `choice_text` for the `Question` class and `Choice` class, respectively.
- Make the necessary migrations for our models in **polls/models.py**
- Register the `Question` class, so that it will be available when we login to the admin page of our site.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCE

Django documentation. (n.d.). Django Software Foundation. Retrieved October 18, 2022, from **https://docs.djangoproject.com/en/4.1/**