Creating a To-Do List with Django

Youtube Video: To Do App | Django 3.0 by Dennis Ivy

Url is: https://www.youtube.com/watch?v=4RWFvXDUmjo

NOTE: x = python manage.py | Feel free to print these notes out!

Step by Step Instructions:

1) Create new django project, open cmd and cd to folder

2) Type `django-admin startproject my_todolist`

3) To start project as a local host site, `x runserver` | to make migrations, `x migrate`

4) Create an admin user with `x createsuperuser`

5) Make application called tasks `x startapp tasks`

6) In my_todolist/settings.py, add `'tasks',` under INSTALLED_APPS

Creating Url Route:

---

7) Make test function in tasks/views.py:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('Hello world')
```

8) Create urls.py file under tasks directory, add:

```
from django.urls import path

from . import views
```

9) Have first path in urls.py call the views.index (which is the function index from views.py) when homepage is called (denoted with ' '):

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

10) Let base url file under my_todolist/urls.py know about this path, add:

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

11) Under urlpatterns list, add:

```
path(' ', include('tasks.urls'))
```

Creating Templates:

---

12) Create 'templates' folder under tasks directory

13) Create 'tasks' folder under the newly created 'templates' folder

14) Create 'list.html' under 'tasks' folder and code in:

```
<h3>To Do</h3>
```

15) Under tasks/views.py, change views.index function to:

```
return render(request, 'task/list.html') #This will return the template just created
```

Creating Model (Task):

---

16) Under tasks/models.py, code in:

```
class Task(models.Model):
        title = models.CharField(max_length=200)
        complete = models.BooleanField(default=False)
        created = models.DateTimeField(auto_now_add=True)

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

17) Prepare the database for migration with: `x makemigrations`

18) Migrate the model Task into database as a database table with: `x migrate`

19) Register to admin interface with model in tasks/admin.py:

```
from .models import *

admin.site.register(Task)
```

Render Data (Tasks) in Template

---

20) Import the models into tasks/views.py: `from .models import *`

21) Make a query for the imported models under views.index function:

```
tasks = Task.objects.all()
```

22) Make context dictionary to throw in all models directly into render function (still under function): `context = {'tasks': tasks}`

23) Add context as a third parameter for render():

```
return render(request, 'tasks/list.html', context)
```

24) Now render the models out to the template, under list.html:

```
{% for task in tasks %} #creates for loop

{% endfor %}
```

25) Pass in the task models as a paragraph in between for loop:

```
<div>

    <p>{{task}}</p>

</div>
```

Creating Model Form:

---

26) Create forms.py under 'tasks' directory, this is where all forms classes are stored

27) Under forms.py, code in:

```
from django import forms
from django.forms import ModelForm
from .models import *


class TaskForm(forms.ModelForm):
    class Meta:
        model = Task #which model for form
        fields = '__all__' #which fields allowed in form
```

28) Import forms.py into tasks/views.py:

```
from .forms import *
```

Under views.index function:

```
form = TaskForm()
```

Pass form model into template by adding a key: `'form': form` under ‘context’ dictionary

29) Pass in template tag under list.html file and set up input field:

(Before for loop but after header, no indents)

```
<form>
    {{ form }}
    <input type="submit" name="Create Task">
</form>
```

30) Delete checkbox field by specifying which field to access:

Under list.html, change `{{ form }}` to `{{ form.title }}`

Creating Item:

---

31) Set form method, whenever form is submitted, to be sent back to views.index function under views.py. Under list.html, change `<form>` to `<form method="POST" action="/">` #action is base url path

32) Under views.py, add:

```
from django.shortcuts import redirect
```

33) Under views.index function, add (between TaskForm() and context init):

```
    if request.method == 'POST':
        form = TaskForm(request.POST) #pass post method
        if form.is_valid():
            form.save()   #saves to the database
        return redirect('/') #send back to same url path
```

34) To add CSRF Token, under list.html before {{form.title}} add:

`{% csrf_token %}` #for site security

35) Runserver and try it out! user typed string is submitted to same view as POST method, then that method puts it into the form and saves it

---

Updating Item:

---

36) Create new html file under templates/tasks called update_task.html:

```
<h3>Update Task</h3>

<form>


    <input type="submit" name="Update Task">

</form>
```

37) Create new view with dynamic url path for updating item (right under views.index):

```
def updateTask(request, pk): #pk is a url pattern
    task = Task.objects.get(id=pk)
    return render(request, 'tasks/update_task.html')
```

38) Input a path under urlpatterns in tasks/urls.py:

```
path('update_task/<str:pk>/', views.updateTask, name="update_task")
```

39) Change previous path with views.index, by adding in a parameter: `name="list"`

40) Update list.html to get path name 'update_task' and id to urlpattern by adding in between and {{ task }} :

```
<a href="{% url 'update_task' task.id %}">Update</a>
```
#percent signs are template tags

41) Create a form of an instance of the object selected (Under task instance initialization in views.updateTask function):

```
form = TaskForm(instance=task)
```

42) Create a context under form init line:

```
context = {'form':form}
```

43) Update render with context argument added at the end: `,context`

44) In update_task.html, add between <form> and <input type=...:

```
{% csrf_token %}
{{ form }}
```
#now form should be shown to user when update link is pressed

45) Update <form> to <form method="POST" action=""> for response when user enters into update form and clicks submit

46) Just like for views.index function, under views.updateTask function after task initialization, add:

```
  if request.method == 'POST':
    form = TaskForm(request.POST, instance=task)
        #need 'instance=' because if not then it will make a new instance
    if form.is_valid():
        form.save()
```

```
    return redirect('/') #redirect back to homepage
```

Deleting Item:

---

47) Option to confirm delete action, create new template under templates/tasks called delete.html, code in:

```
<p>Are you sure you want to delete "{{ item }}"?</p>
<a href="{% url 'list' %}">Cancel</a> #redirects back to homepage
```

48) Just like for updating a task, add a function under tasks/views.py:

```
def deleteTask(request, pk):
        item = Task.objects.get(id=pk)
        context = {'item':item}
        return render(request, 'tasks/delete.html', context)
```

49) Add in a url path under urlpatterns in tasks/urls.py:

```
path('delete/<str:pk>/', views.deleteTask, name="delete")
```

50) Add in link under first created href link in list.html:

```
<a href="{% url 'delete' task.id %}">Delete</a>
```

51) Create a form under delete.html after href:

```
<form method="POST">
  {% csrf_token %}
  <input type="submit" name="Confirm">
#when confirm is clicked, sends POST method back to the view
</form>
```

52) Under item initialization in views.deleteTask, add:

```
   if request.method == 'POST':
      item.delete()
```

```
    return redirect('/')
```

Crossing Out Complete Items Feature

---

53) In list.html, replace {{ task }} with:

```
  {% if task.complete == True %}
  <strike>{{ task }}</strike>
  {% else %}
  <span>{{ task }}</span>
  {% endif %}
```

Style Template

---

54) Replace <h3>To Do</h3> in list.html with a css bootstrap copy and paste:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<style>
    body{
        background-color: #638CB8;
    }
    input{
        width: 100%;
        padding: 12px 20px;
```

```
        margin: 8px 0;
        box-sizing: border-box;
}
input::placeholder {
  color: #d3d3d3;
}
.submit{
        background-color: #6BA3E8;
}
.center-column{
        width:600px;
        margin: 20px auto;
        padding:20px;
        background-color: #fff;
        border-radius: 3px;
        box-shadow: 6px 2px 30px 0px rgba(0,0,0,0.75);
}
.item-row{
        background-color: #906abd;
        margin: 10px;
        padding: 20px;
        border-radius: 3px;
        color: #fff;
        font-size: 16px;
        box-shadow: 0px -1px 10px -4px rgba(0,0,0,0.75);
}
```

```
        .btn-danger{
                background-color: #ffae19;
                border-color: #e59400;
        }
</style>
```

55) Create center item for todo list by sandwiching <form method="POST...> to {% endif %} with:

```
<div class="center-column">
...
</div>
```

56) Styling the button, add in parameter for <input> with:

Replace: `<input type="submit" name="Create Task">`

To: `<input class="btn btn-info" type="submit" name="Create Task">`

57) Do the same with the href= by replacing:

```
   <div>
     <a href="{% url 'update_task' task.id %}">Update</a>
     <a href="{% url 'delete' task.id %}">Delete</a>
```

Into:

```
   <div class="item-row">
     <a class="btn btn-sm btn-info" href="{% url 'update_task' task.id %}">Update</a>
     <a class="btn btn-sm btn-danger" href="{% url 'delete' task.id %}">Delete</a>
```

58) Add in a widget that grabs the form field and sets text input to 'Add new task..' as a placeholder (In tasks/forms.py under class TaskForm(forms.ModelForm)):

```
title = forms.CharField(widget= forms.TextInput(attrs={'placeholder':'Add new
task...'}))
```

59) Done!

60) Cry in the dark corner knowing you didn't understand half of what you were coding