How To Create A To-Do List App Using Django

A To-Do list is a list that you can keep to put all of the Athat you need to complete on a given day. It can be very useful for managing time, by planning your day ahead of time, and prioritizing activities.

Today, we will be creating a basic To-Do application using the Django framework. I'll help you create an app where you can add new items to the list by clicking a button, and you can delete items from the list once the work gets completed.

If you know the fundamentals of Python, Django, and a little bit of HTML, you're good to go with this project. So, let's invest some time to make a beautiful web application using Django.

Step 1: Install Django

Let's start the project. Before we proceed, make sure you have Django installed in your system. If you don't have it, open your command line (or cmd) and install Django using the following command.

```
pip install django
```

If you're on a Mac or Linux, use *pip3 install django*.

Make sure you have an Internet connection. Django will get downloaded automatically and installed in your system. Now, let's move on and start our project.

Step 2: Start a New Project

Open your command prompt or command line. If you want to create your project in a specific folder, go to that folder using the cd command. For example, *cd Desktop* will take you to the Desktop folder.

Start the project by typing in the following command.

```
django-admin startproject todoproject
```

Then you will see that a new folder called *todoproject* is created which contains some files. Drag and drop that folder to your IDE (or you can open that folder in your IDE).

Inside the folder, we have the following files. In my case, I opened my project folder with the VSCode IDE.

Now, go inside the *todoproject* folder by typing the following in your command line/prompt.

```
cd todoproject
```

Now, let's run the project using the following command.

```
python manage.py runserver
```

A URL address will be generated on the command line, something like ' http://127.0.0.1:8000/'.

Copy that URL and paste it on a new tab in your browser. Now you will see your Django project running on your web browser.

Step 3: Create an App

Now, let's create an app. We call each individual functionality of a Django project as an app.

Open your command line and go to your project folder using the *'cd'* command.

Now, you are in the directory called '*todoproject*'. Let's make an app and name it as 'todoapp'.

```
python manage.py startapp todoapp
```

Now open your IDE. Go to *'settings.py'* file and mention your app name in the *INSTALLED_APPS* list.

Step 4: Create a Template

Let's create a template for our todo list app. For that, type in the following command:

```
mkdir templates
```

This command will create a new folder called 'templates' inside the project directory.

Open that in your IDE and manually create a new HTML file called 'todolist.html'.

Now, let's add a heading for our todo list app using an h1 tag.

```
<h1>My To Do List</h1>
```

Now, go to views.py file and add a view for the template.

```
from django.shortcuts import render

def todoappView(request):

    return render(request, 'todolist.html')
```

We need to link this view to the URL. So, go to urls.py and make the following changes at the bottom.

```
from django.contrib import admin

from django.urls import path

from todoapp.views import todoappView



urlpatterns = [

    path('admin/', admin.site.urls),

    path('todoapp/', todoappView),

]
```

Now, let’s go to settings.py file and make changes to the *TEMPLATES* list. Find the section called *DIRS* and replace that line with the following code. Django will look at this directory for templates.

```
'DIRS': [os.path.join(BASE_DIR, 'templates')]
```

Now, run the server and go to the URL/todoapp.

You’ll see a page with the heading “My To Do App”.

Now, let's go to the next steps.

Step 5: Create a Model

Let's create a model representing each todo item that we'll create and store in a database.

So, go to models.py and add the following code.

```
from django.db import models


class TodoListItem(models.Model):


    content = models.TextField()
```

This model represents each item on the todo list. We'll use this model to interact with the database.

Now, we need to do migrations. We do migrations in Django when we need to change the configuration of the database.

Open your command prompt and type in the following:

```
python manage.py makemigrations
```

After that command is executed, type in the following command as well.

```
python manage.py migrate
```

Then, you'll see the changes that we've done on your command prompt.

Step 6: Create the To-Do List

Now, let's try to retrieve todo items to the view. Open views.py file, and import the model.

```
from .models import TodoListItem
```

We'll create a new variable to store all the todo items. Then, we'll return this list to the template file. So make the following changes in the view:

```
def todoappView(request):


    all_todo_items = TodoListItem.objects.all()
    return render(request, 'todolist.html',

    {'all_items':all_todo_items})
```

Here we're returning the list 'all_todo_items' as 'all_items' via a dictionary.

Now, let's go to the templates file and create a list to show all the todo items.

Open todolist.html and add the following code.

```
<ul>

    {% for i in all_items %}

    <li>{{i.content}}

    </li>

    {% endfor %}

</ul>
```

Here we're using a for loop to iterate through the list of items we have in the *all_items* variable.

Now, just below the heading, let's create a forum so that users can add the todo items to the list.

```
<form action="/addTodoItem/" method = "post">{% csrf_token %}

    <input type="text" name="content">

    <input type="submit" value="Add Todo Item">

</form>
```

We have two input tags in the forum. A text field for the user to type in the item, and a button to submit the item.

The action="/addTodoItem/" will send an HTTP request to that URL. We'll be using the POST request here. The tag *{% csrf_token %}* is added for security purposes.

Now, when we refresh the URL, we'll see a page like this:

Now, let’s go to views.py to create a view to handle that post request.

Add the following code to create a view to handle the request:

```
def addTodoView(request):

    x = request.POST['content']

    new_item = TodoListItem(content = x)

    new_item.save()

    return HttpResponseRedirect('/todoapp/')
```

We used a method called HttpResponseRedirect. So, we need to import that at the top of this file before we actually use it.

```
from django.http import HttpResponseRedirect
```

Now, go to urls.py and create a new path for the view that we’ve just now created.

```
 path('addTodoItem/',addTodoView),
```

Also, don’t forget to import the addTodoView to the urls.py file. So, make the following change:

```
from todoapp.views import todoappView, addTodoView
```

Now, refresh the page on your browser. You’ll be able to add new items to the list.

Step 7: Add Delete Buttons

Now, let’s go to the todolist.html file and create delete buttons for each todo list item.

So, add the following button in between the <li> tags just after the content.

```
<form action="/deleteTodoItem/{{i.id}}/" method = "post">{% csrf_token %}

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

</form>
```

We use the id for identifying each specific item from the list.

Now, let’s go to urls.py and add a new path for the delete requests.

```
path('deleteTodoItem/<int:i>/', deleteTodoView),
```

We used the integer i as a primary key to identify the specific item in the list that needs to be deleted.

Also, don’t forget to import the deleteTodoView in this file.

```
from todoapp.views import todoappView, addTodoView, deleteTodoView
```

Now, let’s go to views.py and create a new view function for the delete requests.

```
def deleteTodoView(request, i):

    y = TodoListItem.objects.get(id= i)
    y.delete()


    return HttpResponseRedirect('/todoapp/')
```

Now, our basic todo list app is ready.