

# CRUD Operations and Forms

---

In the present document we will adapt our forms from the previous document in order to accept Create, Read, Update, and Delete (CRUD) operations that affect the database.

There are four verbs used in any web operation defined in the HTTP/HTTPS protocol:

Verb	Description	Database Equivalent	Operation
POST	Informs the site that there's data being sent to it.	INSERT	CREATE
GET	Informs the site that should send a response with the data indicated by the argument	SELECT	READ
UPDATE	Informs the site that the data sent must update already existing data.	UPDATE	UPDATE
DELETE	Informs the site that it should any data indicated by the parameter	DELETE	DELETE

We need to modify our form views in order to respond to each one of the HTTP/HTTPS verbs and interact with our database accordingly.

## Initial Modifications to our Forms

First, we are not interested in showing all the fields present in our /projects/models.py

The screenshot shows a web browser window with the URL `127.0.0.1:8000/create-project/`. The page title is "Project Form". It contains the following fields:

- Title:
- Description:
- Demo link:
- Source link:
- Tags: A dropdown menu with options "React", "Django", "Python", and "JavaScript".
- Vote total:  0 X
- Vote ratio:  X
- 

FOOTER

We want to leave Vote total and Vote ratio out of the form. With that in mind, let's modify our forms.py in /projects/forms.py to accept only the fields for:

- title
- description
- demo\_link
- source\_link
- tags

in our `createProject` class.

The screenshot shows the Visual Studio Code interface with the file `forms.py` open. The code defines a `ProjectForm` class that inherits from `ModelForm` and specifies the `Meta` class with the `model` set to `Project` and the `fields` list containing `'title'`, `'description'`, `'demo_link'`, `'source_link'`, and `'tags'`.

```

File Edit Selection View Go Run Terminal Help
forms.py - devsearch - Visual Studio Code
EXPLORER ... models.py forms.py views.py project_form.html urls.py ...
DEVSERCH projects > forms.py > ProjectForm > Meta
1 om django.forms import ModelForm
2 om .models import Project
3
4
5 ass ProjectForm(ModelForm):
6     class Meta:
7         model = Project
8         fields = ['title', 'description', 'demo_link', 'source_link', 'tags']
9

```

so, our form should look like this:

Logo

Add Project

## Project Form

Title:

Description:

Demo link:

Source link:

Tags:

FOOTER

If you submit something into the form:

File Edit Selection View Go Run Terminal Help

project\_form.html - devsearch - Visual Studio Code

EXPLORER    ...    views.py    project\_form.html    url:

DEVSERACH

- > env
- > projects
  - > \_\_pycache\_\_
  - > migrations
  - > templates\projects
    - > project\_form.html
    - > projects.html
    - > single-project.html
  - & \_\_init\_\_.py
  - & admin.py
  - & apps.py
  - & forms.py
  - & models.py
  - & tests.py
  - & urls.py
  - & views.py
- > templates
  - > main.html
  - > navbar.html
  - db.sqlite3
  - manage.py

project\_form.html

```

1  {% extends 'main.html' %} 
2  {% block content %} 
3 
4  <h1>Project Form</h1> 
5 
6  <form method="POST"> 
7      {% csrf_token %} 
8 
9      {{ form.as_p }} 
10     <input type="submit"> 
11 
12 </form> 
13 {% endblock %}

```

PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

System check identified no issue

Django version 3.2.4, using sett
Starting development server at h
Quit the server with CTRL-BREAK.
[ ] "GET /cre

Add Project

## Project Form

Title:

Description:

Demo link:

Source link:

Tags:

you should be able to see the POST method after the submission:

```
- 18:00:28
Django version 3.2.4, using settings 'devsearch.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[ 18:00:30] "GET /create-project/ HTTP/1.1" 200 1568
[ 18:00:59] "POST /create-project/ HTTP/1.1" 200 1568
```

The reply for the POST method is given by the `form` argument method in the `project_form.html`, but the form doesn't do anything with the input data. The reason is that there's another argument missing:

```
File Edit Selection View Go Run Terminal Help
project_form.html - devsearch - Visual Studio Code
EXPLORER ... views.py project_form.html urls.py
DEVSERACH projects > templates > projects > project_form.html > form
1  {% extends 'main.html' %} 
2  {% block content %} 
3      The URI of a program that processes the form information.
4  <h1>Pr This value can be overridden by a formaction attribute on a
5      <button> or <input> element.
6  <form action="" method="POST">
7      {% csrf_token %}
8
9      {{form.as_p}}
10     <input type="submit">
11  </form>
12
13  {% endblock %}
```

This missing argument is the `formaction` attribute. The `formaction` attribute is the direction to the function that will process the form data.

For the time being, we'll leave this argument empty (an empty string) since we have to define the processing function first. We'll return after we have defined such function.

Let's see how the data is passed from the html form to the function form `createProject` class. Our first order of business is to instanciate our `ProjectForm` class inside the `createProject` class in the `/projects/views.py`

The `request` parameter contains information on the request type. The request type is identified by an enumerate:

```
request.POST
request.GET
request.UPDATE
request.DELETE
```

that we can use to redirect the function accordingly. In our case, we'll use `request.POST` and retreive the form input data on a variable `form`

The screenshot shows the Visual Studio Code interface. The left sidebar contains project files like `views.py`, `urls.py`, and `models.py`. The main editor shows Python code for a Django application:

```

12
13 def project(request, pk):
14     projectObj = Project.objects.get(id=pk)
15     return render(request, 'projects/single-project.html', {'project': projectObj})
16
17
18 def createProject(request):
19     form = ProjectForm()
20
21     if request.method == 'POST':
22         form = ProjectForm(request.POST)
23
24         context = {'form': form}

```

The terminal below shows the server starting and log entries:

```

18:00:28
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[18:00:30] "GET /create-project/ HTTP/1.1" 200 1568
[18:00:59] "POST /create-project/ HTTP/1.1" 200 1568

```

But for demonstration purposes, let's comment the line which assigns the variable from inside the `if` statement and print the argument `request.POST` and introduce some data into the form:

The screenshot shows the Visual Studio Code interface with the same file structure and code as before, but with a comment added to the `if` statement:

```

12
13 def project(request, pk):
14     projectObj = Project.objects.get(id=pk)
15     return render(request, 'projects/single-project.html', {'project': projectObj})
16
17
18 def createProject(request):
19     form = ProjectForm()
20
21     if request.method == 'POST':
22         print(request.POST)
23         #form = ProjectForm(request.POST)
24

```

The terminal shows the system check and server start:

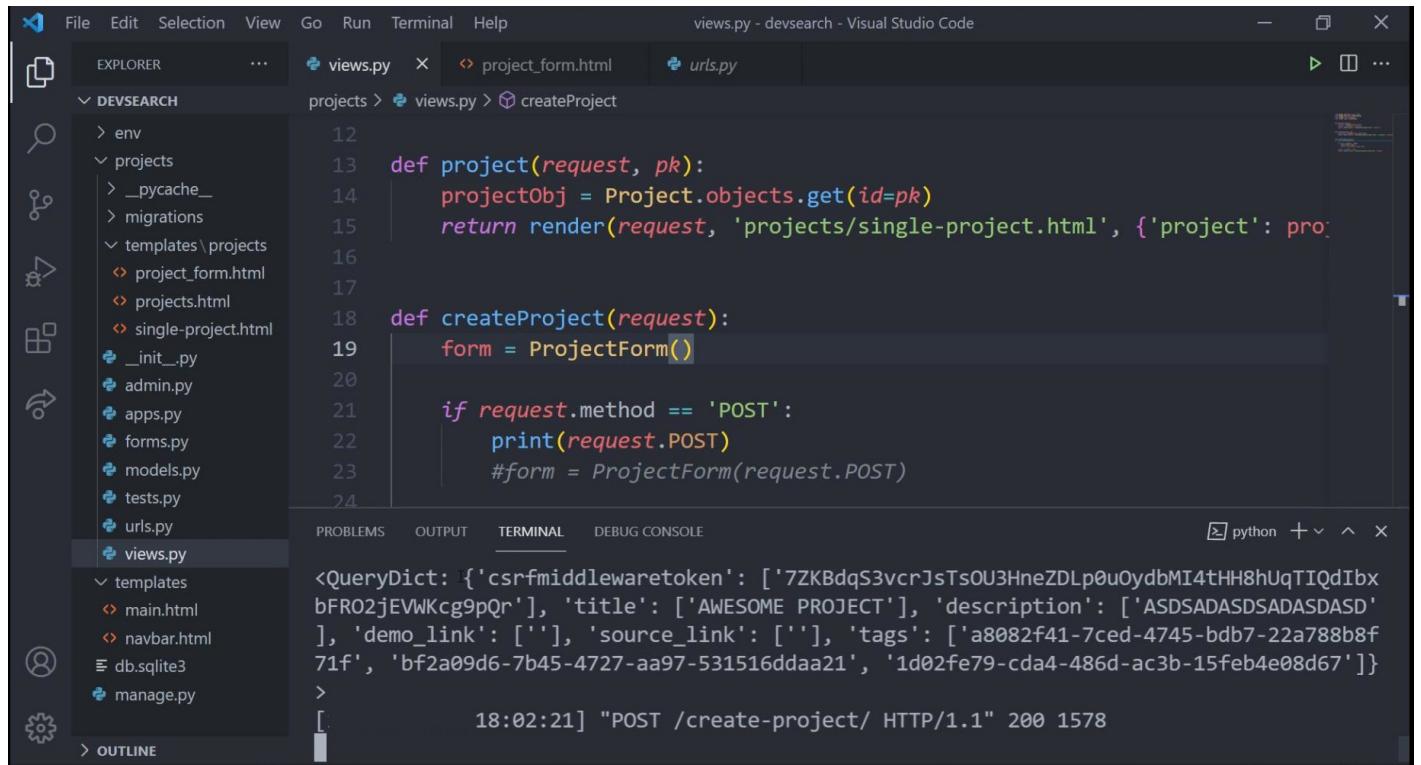
```

System check identified no issues
18:02:00
Django version 3.2.4, using settings 'devsearch.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

```

A browser window is open at `127.0.0.1:8000/create-project/`, showing a form titled "Project Form". The form has fields for Title (set to "AWESOME PROJECT"), Description (set to "ASDSADASDASDASDASD"), Demo link, Source link, and Tags (with "Python" selected). A "Submit" button is at the bottom.

The reply is an object of type `QueryDict` which contains all the data returned by the form



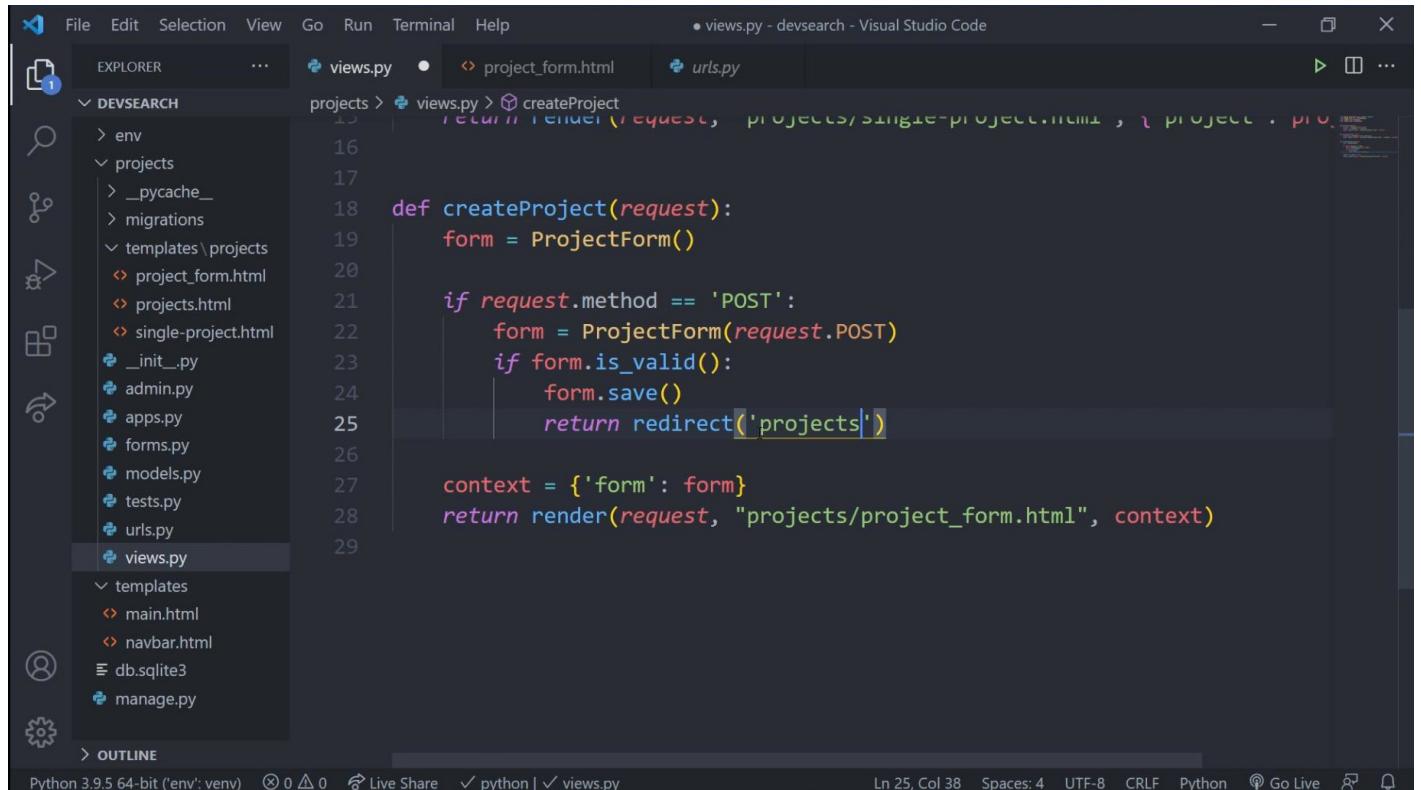
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "DEVSERACH".
- Editor:** The "views.py" file is open, containing Python code for a Django project. It includes functions for displaying a single project and creating a new project.
- Terminal:** Shows a successful POST request to "/create-project/" at 18:02:21.
- Status Bar:** Shows Python 3.9.5 64-bit ('env': venv) and other development tools.

```

12
13     def project(request, pk):
14         projectObj = Project.objects.get(id=pk)
15         return render(request, 'projects/single-project.html', {'project': pro:
16
17
18     def createProject(request):
19         form = ProjectForm()
20
21         if request.method == 'POST':
22             print(request.POST)
23             #form = ProjectForm(request.POST)
24
25
26
27
28
29
    
```

Going back to our /projects/views.py and add the following code:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "DEVSERACH".
- Editor:** The "views.py" file is open, containing Python code for a Django project. It includes a modified "createProject" function that saves the form if it's valid.
- Status Bar:** Shows Python 3.9.5 64-bit ('env': venv) and other development tools.

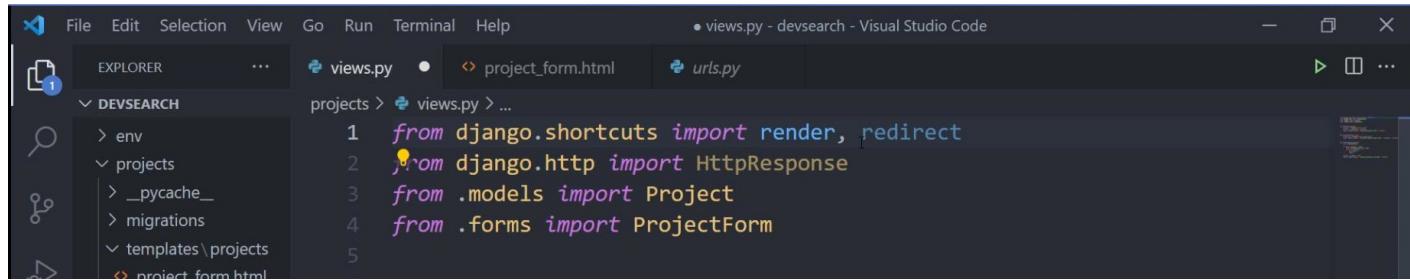
```

16
17
18     def createProject(request):
19         form = ProjectForm()
20
21         if request.method == 'POST':
22             form = ProjectForm(request.POST)
23             if form.is_valid():
24                 form.save()
25                 return redirect('projects')
26
27
28
29
    
```

A beauty of django is its automatic form validation facilities. Django forms auto-check that the contents of each field do not inject code to tables and their data types correspond to each field. Hence, the member function `form.is_valid()` indicate to us that the form complies with all the fields we introduced in our model for that specific form.

If the form is valid, i. e., all the fields pass the default validation tests, we save the form and return to the main page *projects* through a redirection.

The ***django.shortcuts.redirect*** function is a facility that allow us to go to specific parts of the project. We need to include it via an import:



```
File Edit Selection View Go Run Terminal Help
EXPLORER views.py project_form.html urls.py
DEVSERACH projects > views.py > ...
1 from django.shortcuts import render, redirect
2 from django.http import HttpResponseRedirect
3 from .models import Project
4 from .forms import ProjectForm
5
```

### How does *redirect* work?

*redirect* works on dynamic assignment mechanisms we've seen before in our templates with url:

```
{% extends 'main.html' %}
{% block content %}

<p> Hello, you've reach the {{ page }} page </p>

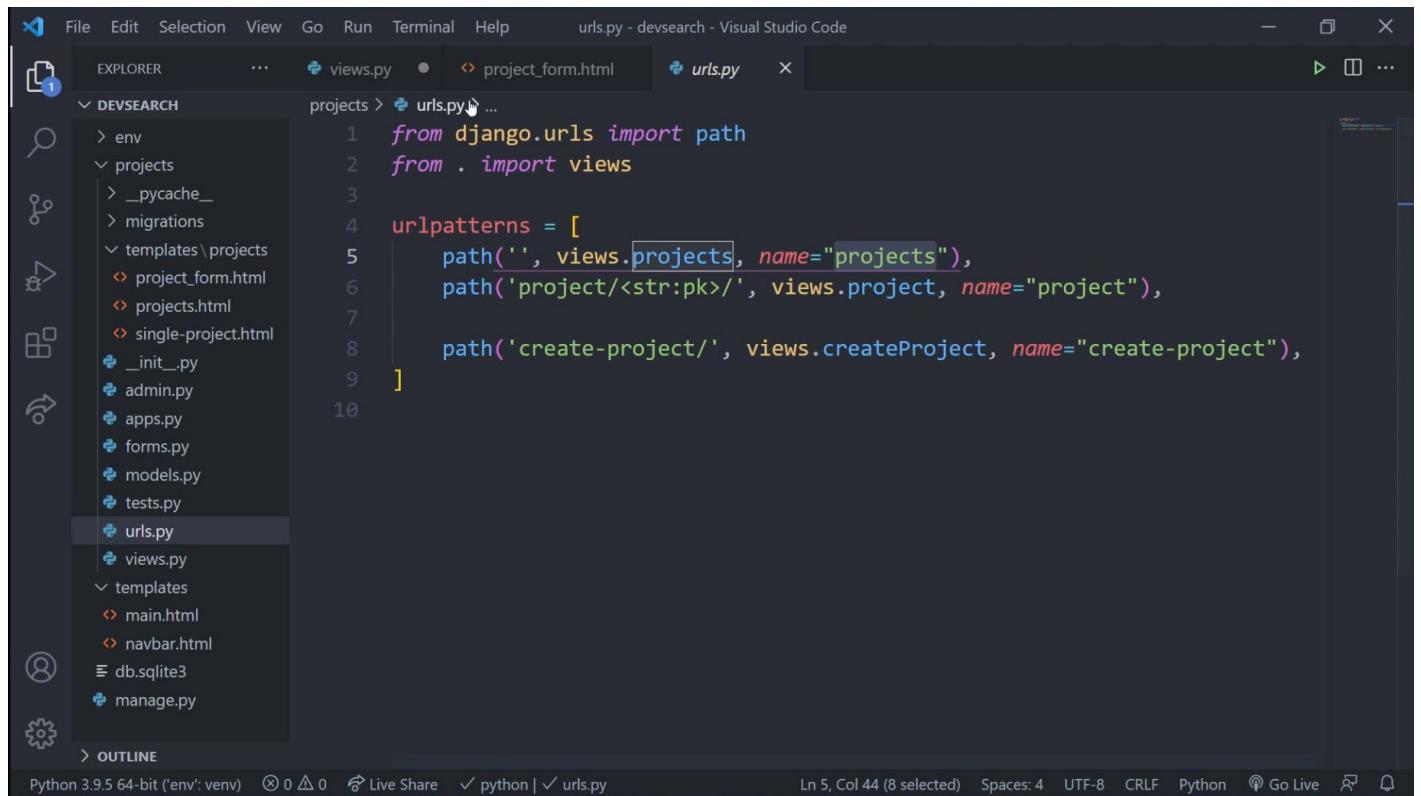
<!-- Let's render the list of projects -->
<h1> Our Project List: </h1>
<ol>
    {% for project in projects %}
        <li> Project Title: <a href="{% url 'project' project.id %}">{{ project.title }}</a> --- {{ project.description }} </li>
    {% endfor %}
</ol>

{% endblock content %}
```

Notice that

```
<a href="{% url 'project' project.id %}">{{ project.title }} </a>
```

'project' is the *name = 'project'* in our /projects/urls.py file:



```
File Edit Selection View Go Run Terminal Help urls.py - devsearch - Visual Studio Code
EXPLORER ... views.py ... urls.py ...
PROJECTS > urls.py ...
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.projects, name="projects"),
6     path('project/<str:pk>/', views.project, name="project"),
7
8     path('create-project/', views.createProject, name="create-project"),
9 ]
10
```

so, for returning to the main page ('projects') page, we use the redirection function

```
if form.is_valid:
    form.save()
    return redirect('projects')
```

uses the path() with the *name = 'projects'*. The assignment is dynamic.

## Insert Data to our Database

This is our testing phase for the form. Let's introduce some data into our form:

**Logo**

[Add Project](#)

## Project Form

Title:

Description:

Demo link:

Source link:

Tags:

FOOTER

and after submitting the form, you should return to your projects page which should also include the new project:

**Logo**

[Add Project](#)

## Projects

ID	Project	Positive Votes	Votes	
b7074c3e-c42b-4527-92e3-c3e0eb40d750	Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m. <a href="#">View</a>
a7907870-580f-4db8-a110-8e10a88b15a5	Portfolio Website	65%	6	June 15, 2021, 1:25 p.m. <a href="#">View</a>
4751d9bc-3709-46eb-828b-fab89d848ca8	Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m. <a href="#">View</a>
4403d6b4-7f86-4c4b-912d-49540b17a7d6	Code Sniper	78%	80	June 15, 2021, 3:53 p.m. <a href="#">View</a>
befd0ddd-829a-490d-81df-2eb25698e104	Yogo Vive	98%	51	June 15, 2021, 3:53 p.m. <a href="#">View</a>
f4f79b7e-41be-4276-9281-88479c6419b6	Cool tuts	0%	0	June 15, 2021, 10:04 p.m. <a href="#">View</a>

FOOTER

Let's clean the view a little bit by removing the ID part of the table in the /projects/templates/projects/projects.html:

The screenshot shows the Visual Studio Code interface. The left sidebar displays the project structure under 'EXPLORER'. The main editor area shows the content of 'projects.html'. The code is a table definition with columns for ID, Project, Positive Votes, Votes, and a 'View' link.

```

5   <table>
6     <tr>
7       <th>ID</th>-----
8       <th>Project</th>
9       <th>Positive Votes</th>
10      <th>Votes</th>
11      <th></th>
12    </tr>
13    {% for project in projects %}
14    <tr>
15      <td>{{project.id}}</td>-
16      <td>{{project.title}}</td>
17      <td>{{project.vote_ratio}}%</td>
18      <td>{{project.vote_total}}</td>
19      <td>{{project.created}}</td>
20      <td><a href="{% url 'project' project.id %}">View</a></td>
21    </tr>
22    {% endfor %}
23  </table>
24
25

```

Our template should looks cleaner:

The screenshot shows a web browser window at '127.0.0.1:8000'. The page has a header with 'Logo' and a 'Add Project' button. Below it is a section titled 'Projects' containing a table with columns 'Project', 'Positive Votes', and 'Votes'. The table lists six projects with their respective details and a 'View' link.

Project	Positive Votes	Votes		
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.	<a href="#">View</a>
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.	<a href="#">View</a>
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.	<a href="#">View</a>
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.	<a href="#">View</a>
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.	<a href="#">View</a>
Cool tuts	0%	0	June 15, 2021, 10:04 p.m.	<a href="#">View</a>

FOOTER

THAT'S IT!

The data is already stored in the database when we used the `form.save()` method.

Try more data:

**Logo**

[Add Project](#)

## Project Form

Title:

Description:  
It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed

Description:

Demo link:

Source link:

Tags:  React  Django  Python  JavaScript

FOOTER

With the new data:

Project	Positive Votes	Votes	Last updated	Action
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.	<a href="#">View</a>
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.	<a href="#">View</a>
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.	<a href="#">View</a>
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.	<a href="#">View</a>
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.	<a href="#">View</a>
Cool tutz	0%	0	June 15, 2021, 10:04 p.m.	<a href="#">View</a>
Second project	0%	0	June 15, 2021, 10:05 p.m.	<a href="#">View</a>

FOOTER

If you enter the admin panel (127.0.0.1:8000/admin/projects/project) you should be able to see the new added projects to the database:

Django administration

Home > Projects > Projects

AUTHENTICATION AND AUTHORIZATION

- Groups [+ Add](#)
- Users [+ Add](#)

PROJECTS

- Projects [+ Add](#)
- Reviews [+ Add](#)
- Tags [+ Add](#)

Select project to change

Action:  Go 0 of 7 selected

<input type="checkbox"/> PROJECT
<input type="checkbox"/> Cool tuts
<input type="checkbox"/> Yoga Vive
<input checked="" type="checkbox"/> Ecommerce Website
<input type="checkbox"/> Portfolio Website
<input type="checkbox"/> Second project
<input type="checkbox"/> Mumble Social Network
<input type="checkbox"/> Code Sniper

7 projects

ADD PROJECT [+ Add](#)

## Update Form

In this section we'll create the update form for the Projects application.

The functional class for this form will be very close to the `createProject` class, as a matter of fact, we can copy the `createProject` class and do some small modifications to the class:

```

File Edit Selection View Go Run Terminal Help views.py - devsearch - Visual Studio Code
EXPLORER ... views.py x projects.html project_form.html
DEVS... env
projects _pycache_
migrations
templates\projects
project_form.html
projects.html
single-project.html
__init__.py
admin.py
forms.py
models.py
tests.py
urls.py
views.py
templates
main.html
navbar.html
db.sqlite3
manage.py
> OUTLINE
return render(request, "projects/project_form.html", context)

def updateProject(request, pk):
    project = Project.objects.get(id=pk)
    form = ProjectForm(instance=project)

    if request.method == 'POST':
        form = ProjectForm(request.POST, instance=project)
        if form.is_valid():
            form.save()
            return redirect('projects')

    context = {'form': form}
    return render(request, "projects/project_form.html", context)

```

views.py - devsearch - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER ... views.py x projects.html project\_form.html

DEVS... env

projects \_pycache\_

migrations

templates\projects

project\_form.html

projects.html

single-project.html

\_\_init\_\_.py

admin.py

forms.py

models.py

tests.py

urls.py

views.py

templates

main.html

navbar.html

db.sqlite3

manage.py

> OUTLINE

Python 3.9.5 64-bit ('env': venv) ⚡ 0 △ 0 🔍 Live Share ✓ python | ✓ views.py

Ln 33, Col 14 Spaces: 4 UTF-8 CRLF Python ⚡ Go Live 🔍

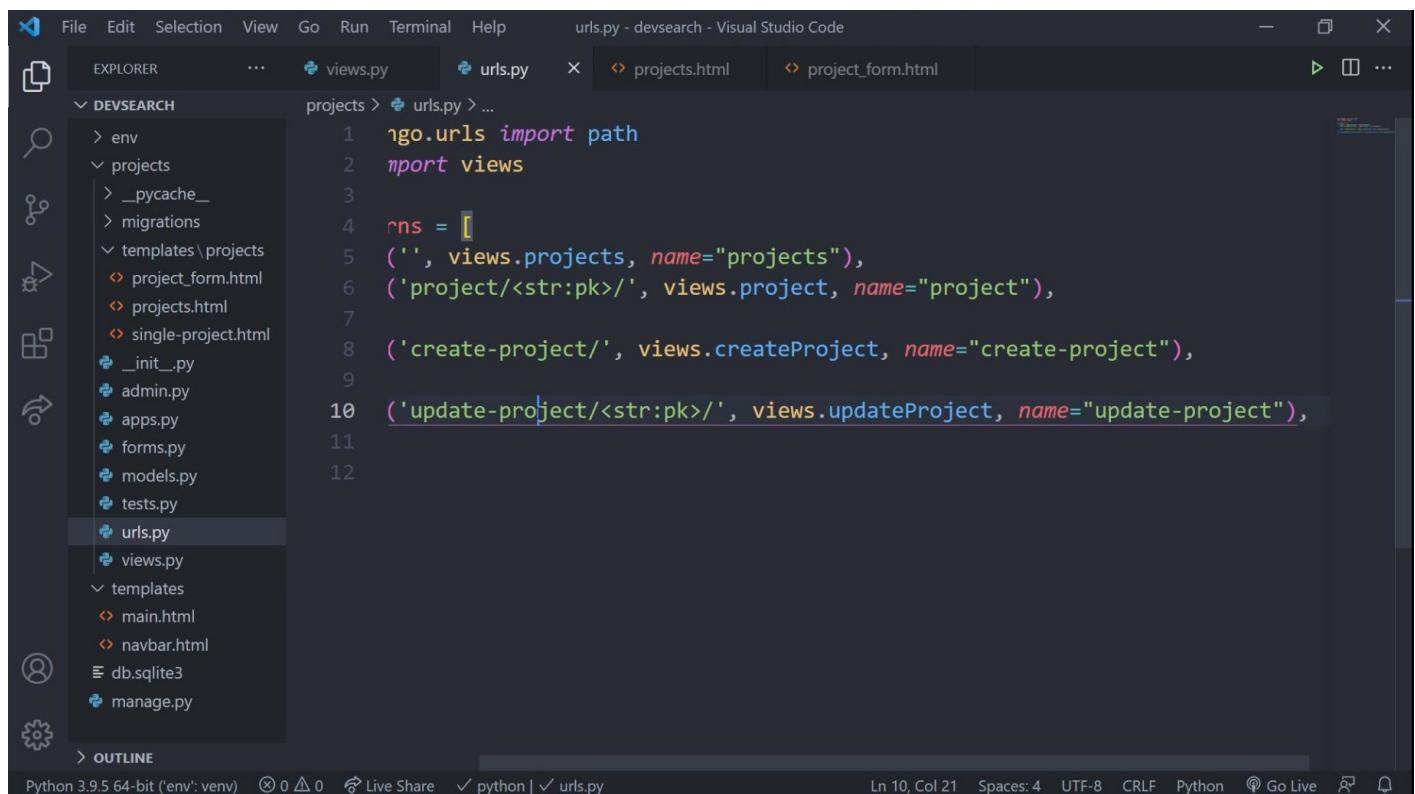
The differences are:

- The extra parameter ***pk***.
- We still need to create the form, but since we're retrieving the data with the ***pk*** information, we need to fill up the form with the same data as in the database, therefore the code:

```
project = Project.objects.get(id=pk)      # Retreives the form data
form = ProjectForm(instance=project)      # Writes the data into the form
```

- We will be implicitly updating the records and thus we'll still use the "POST" HTTP/HTTPS method instead of the "UPDATE" because in this case we're assuming that when the parameter ***pk*** is available, then the data exists in the database and therefore is just an update operation in the database. There's not much difference between "POST" and "UPDATE" except that a design preference. Also "UPDATE" is less commonly used than "POST"
- Since this is an update operation, we'll still be using the same ***project\_form.html***

The next step is to update our /projects/urls.py file to include our new ***updateProject*** class.



The screenshot shows the Visual Studio Code interface with the following details:

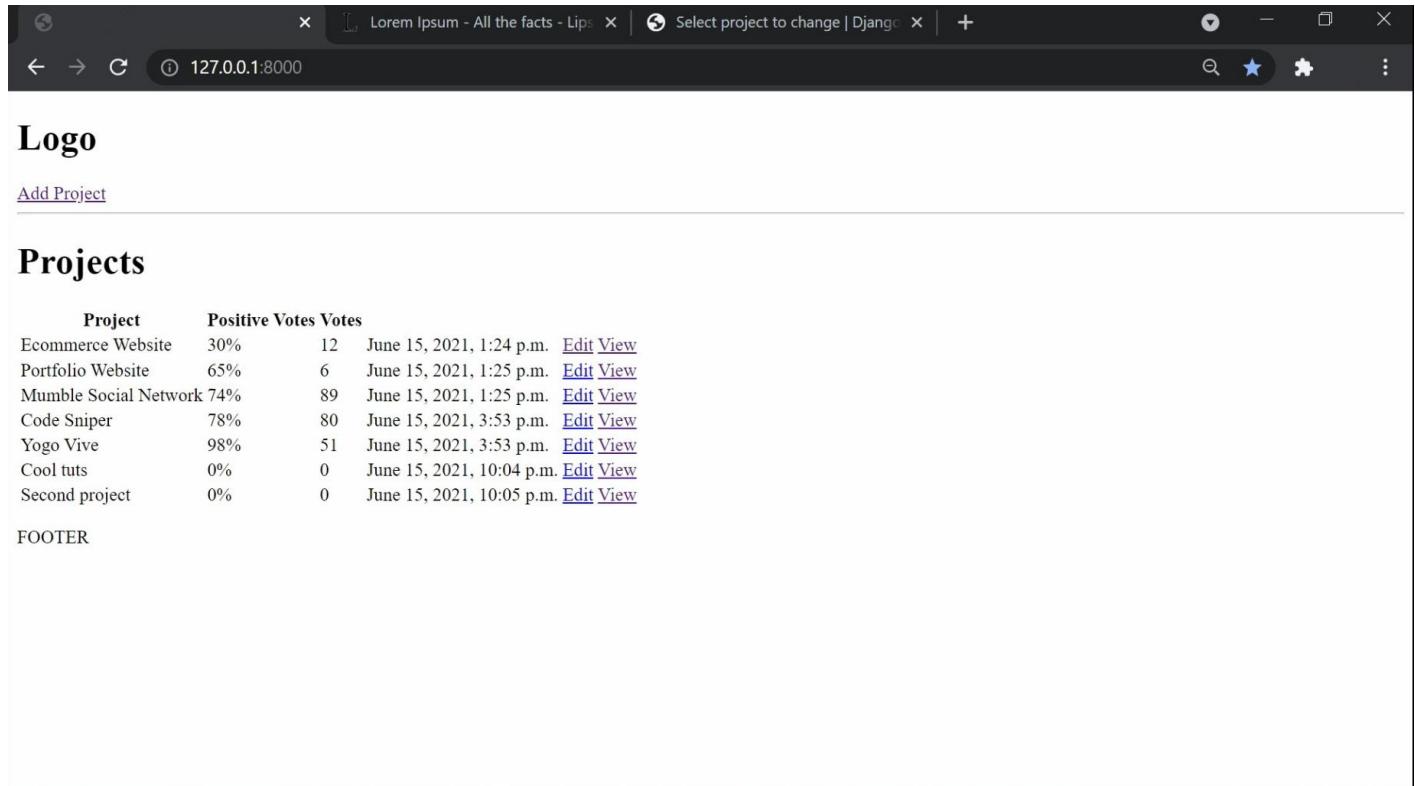
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** urls.py - devsearch - Visual Studio Code.
- Explorer Panel:** Shows the project structure under 'DEVSERACH' (env, projects, migrations, templates\projects, views, urls, db.sqlite3, manage.py).
- Code Editor:** The 'urls.py' file is open, showing URL patterns. The last line added is highlighted in blue: `10 ('update-project/<str:pk>/', views.updateProject, name="update-project"),`
- Status Bar:** Python 3.9.5 64-bit ('env': venv) | Live Share | python | urls.py | Ln 10, Col 21 | Spaces: 4 | UTF-8 | CRLF | Python | Go Live |

We added the code:

```
path('update-project/<str:pk>/', views.updateProject, name="update-project"),
```

to our ***urlpatterns*** list.

Finally, we'll change our projects.html template in /projects/templates/projects/projects.html so we'll be able to see something like the following:

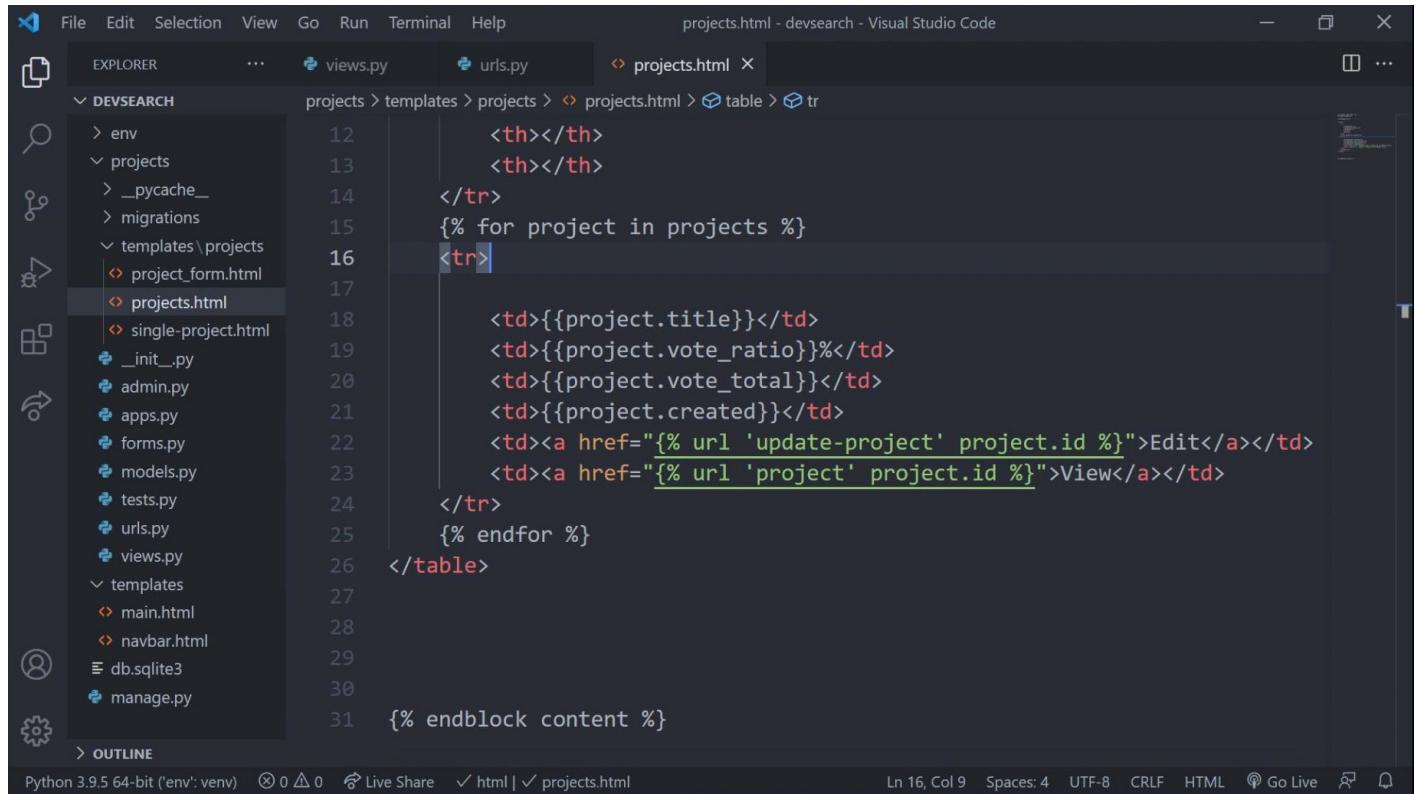


The screenshot shows a web browser window with the URL `127.0.0.1:8000`. The page displays a logo and a heading "Projects". Below the heading is a table listing seven projects. Each row in the table includes the project name, its positive vote percentage, the number of votes, and the date and time of the last update, followed by two hyperlinks: "Edit" and "View".

Project	Positive Votes	Votes	Last Update	Edit	View
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Cool tuts	0%	0	June 15, 2021, 10:04 p.m.	<a href="#">Edit</a>	<a href="#">View</a>
Second project	0%	0	June 15, 2021, 10:05 p.m.	<a href="#">Edit</a>	<a href="#">View</a>

FOOTER

The difference is that we added an hyperlink for form editing. The change in the projects.html is as shown:



```
<th></th>
<th></th>
</tr>
{% for project in projects %}
<tr>

    <td>{{project.title}}</td>
    <td>{{project.vote_ratio}}%</td>
    <td>{{project.vote_total}}</td>
    <td>{{project.created}}</td>
    <td><a href="{% url 'update-project' project.id %}">Edit</a></td>
    <td><a href="{% url 'project' project.id %}">View</a></td>
</tr>
{% endfor %}
</table>
```

we added the code:

```
<td><a href="{% url 'update-project' project.id %}">Edit</a></td>
```

with in the for cycle.

Test the interface by refreshing the page and click on any project Edit hyperlink, for example:

The screenshot shows a Django application's update project form. At the top, there are tabs for "Lorem Ipsum - All the facts - Lips" and "Select project to change | Django". The main content area has a heading "Logo" and a "Project Form" section. Inside, there is a "Title" field containing "Ecommerce Website (Updated)". Below it is a rich text editor with placeholder text about Lorem Ipsum. A "Description" field contains "versions of Lorem Ipsum.". There are also "Demo link" and "Source link" fields, both currently empty. A "Tags" dropdown menu is open, showing options like "React", "Django", "Python", and "JavaScript", with "JavaScript" selected. A "Submit" button is at the bottom left. A "FOOTER" section is at the bottom right.

We should see the change in the title with the "(Updated)" text on the project title:

The screenshot shows a Django application's project list page. At the top, there are tabs for "Lorem Ipsum - All the facts - Lips" and "Select project to change | Django". The main content area has a heading "Logo" and a "Projects" section. Below it is a table listing projects:

Project	Positive Votes	Votes	
Ecommerce Website (Updated)	30%	12	June 15, 2021, 1:24 p.m. <a href="#">Edit</a> <a href="#">View</a>
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m. <a href="#">Edit</a> <a href="#">View</a>
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m. <a href="#">Edit</a> <a href="#">View</a>
Code Sniper	78%	80	June 15, 2021, 3:53 p.m. <a href="#">Edit</a> <a href="#">View</a>
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m. <a href="#">Edit</a> <a href="#">View</a>
Cool tuts	0%	0	June 15, 2021, 10:04 p.m. <a href="#">Edit</a> <a href="#">View</a>
Second project	0%	0	June 15, 2021, 10:05 p.m. <a href="#">Edit</a> <a href="#">View</a>

A "FOOTER" section is at the bottom right.

## Delete Form

Our objective is now, to modify our projects.html template now to include a Delete operation as shown:

Project	Positive Votes	Votes	Last Updated
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.
Cool tuts	0%	0	June 15, 2021, 10:04 p.m.
Second project	0%	0	June 15, 2021, 10:05 p.m.

But first, we'll create a confirmation page that shows us the result of the Delete operation instead of simply having the page disappear from the list of projects.

In our `/projects/templates/projects` folder, we now create a new template `delete_template.html`:

"/>

```

1  {% extends 'main.html' %}

2

3  {% block content %}

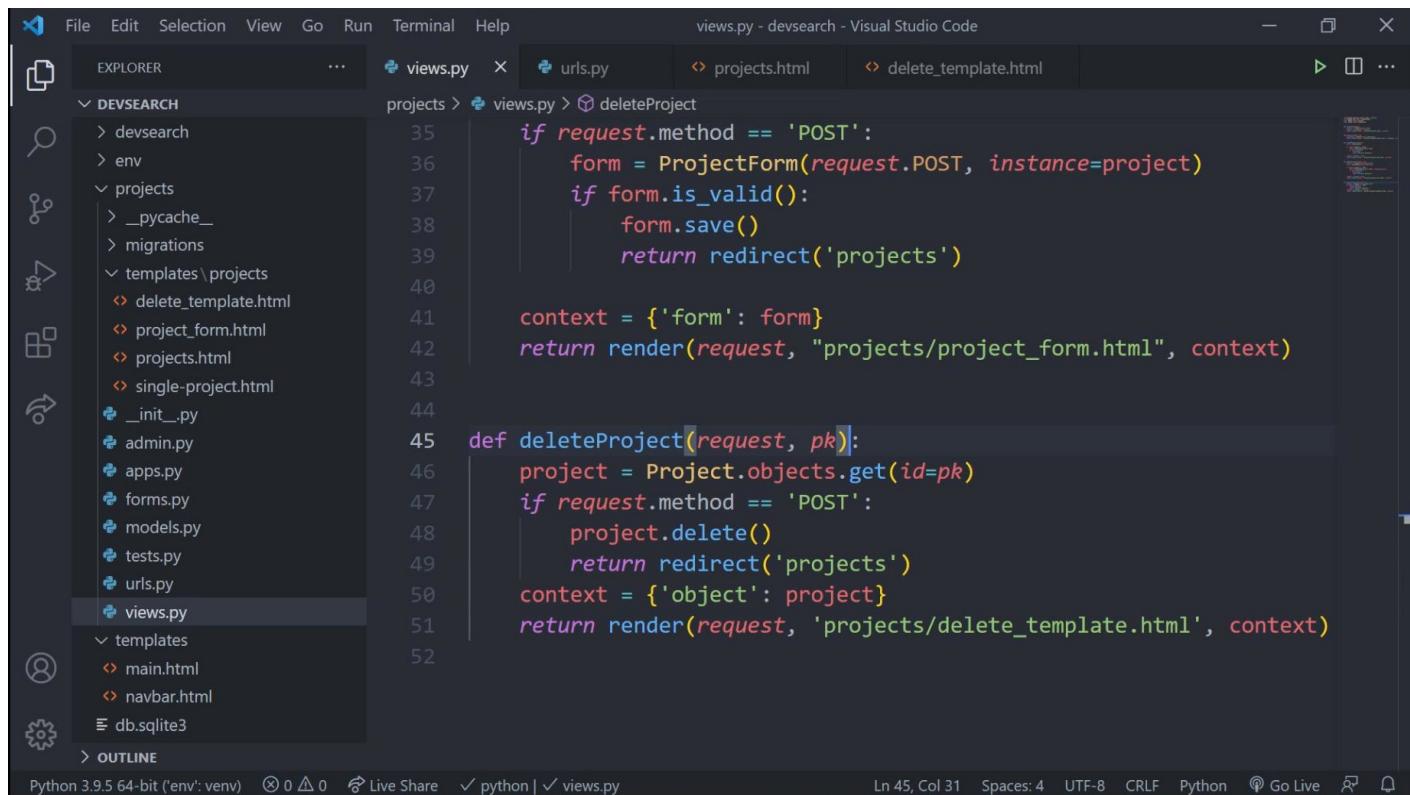
4

5  <form action="" method="POST">
6      {% csrf_token %}
7      <p>Are you sure you want to delete "{{object}}"?</p>
8
9      <a href="{% url 'projects' %}">Go Back</a>
10
11     <input type="submit" value="Confirm" />
12
13
14  {% endblock %}
    
```

Few things to notice:

- The **csfr\_token** that is always present in any template.
- Now our template parameter is "**object**." This makes the template generic enough for other pages since it is not specific to **projects**.
- The template is only for confirmation and therefore uses a button for such confirmation.
- To cancel the **DELETE** operation, we simply return to the **projects** page.

Our next step is to create a routing class for the delete operation in /projects/views.py:



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "DESEARCH". The "views.py" file is selected.
- Code Editor:** Displays the Python code for the "views.py" file, specifically the "deleteProject" function. The code handles POST requests to update a project form and DELETE requests to delete a project from the database.
- Status Bar:** Shows the Python version (3.9.5), environment (env), live share status, and file path (views.py).
- Bottom Right:** Includes status indicators for line (Ln 45), column (Col 31), spaces (Spaces: 4), encoding (UTF-8), line endings (CRLF), Python, Go Live, and a refresh icon.

```

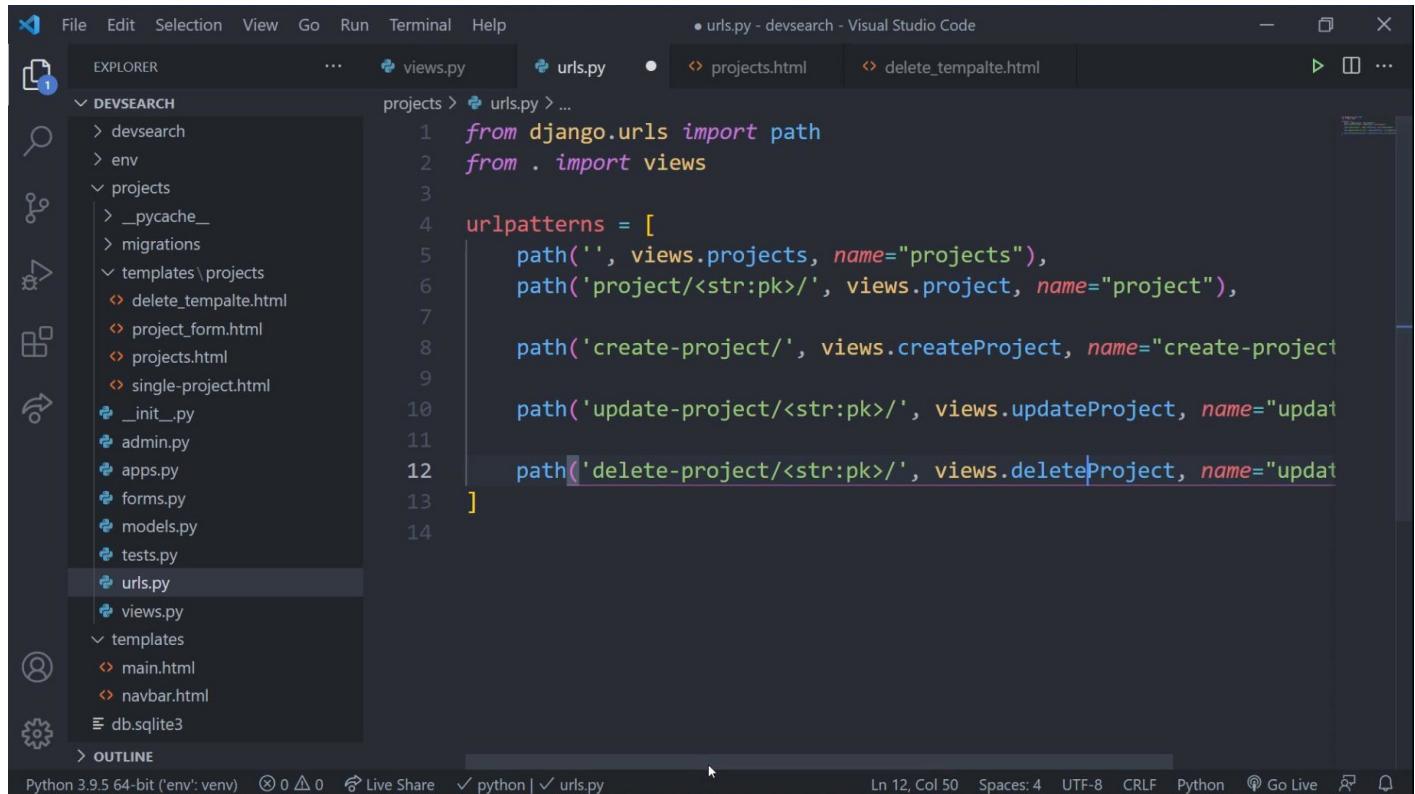
35     if request.method == 'POST':
36         form = ProjectForm(request.POST, instance=project)
37         if form.is_valid():
38             form.save()
39             return redirect('projects')
40
41         context = {'form': form}
42         return render(request, "projects/project_form.html", context)
43
44
45     def deleteProject(request, pk):
46         project = Project.objects.get(id=pk)
47         if request.method == 'POST':
48             project.delete()
49             return redirect('projects')
50         context = {'object': project}
51         return render(request, 'projects/delete_template.html', context)
52

```

Notice three important things:

- Again, we're using the "POST" method, the reason being that it's easier to intercept when dealing with interactive UI. If we were to use "DELETE", we would have had to generate the method manually after clicking the submit button.
- We are using the parameter **pk** again to specify which record has to be deleted from the database.
- The argument **object** is used to pass the data to the confirmation template (**delete\_template.html**)

Still, we need to route the routing class by adding its path in the /projects/urls.py file:



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

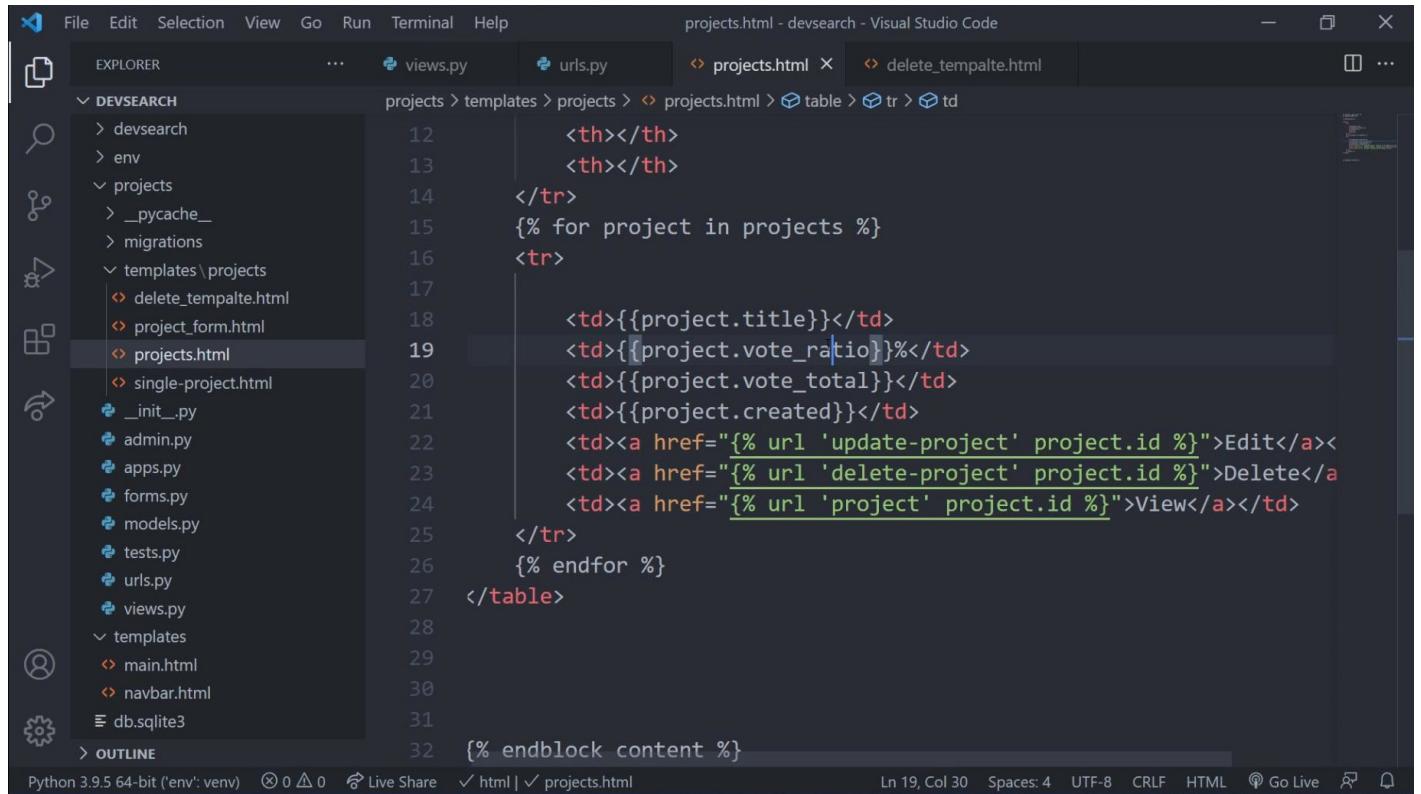
urlpatterns = [
    path('', views.projects, name="projects"),
    path('project/<str:pk>', views.project, name="project"),

    path('create-project/', views.createProject, name="create-project"),
    path('update-project/<str:pk>', views.updateProject, name="update-project"),
    path('delete-project/<str:pk>', views.deleteProject, name="delete-project")
]
```

We added the code:

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

Now we'll be back to our projects template (projects.html) to add the necessary code to include the deletion view:



```
12     <th></th>
13     <th></th>
14   </tr>
15   {% for project in projects %}
16   <tr>
17
18     <td>{{project.title}}</td>
19     <td>{{project.vote_ratio}}%</td>
20     <td>{{project.vote_total}}</td>
21     <td>{{project.created}}</td>
22     <td><a href="{% url 'update-project' project.id %}">Edit</a><
23     <td><a href="{% url 'delete-project' project.id %}">Delete</a>
24     <td><a href="{% url 'project' project.id %}">View</a></td>
25   </tr>
26   {% endfor %}
27 </table>
28
29
30
31
32  [% endblock content %]
```

where we add the code:

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

This should be the last step.

Let's test the deletion process. Open up the projects page:

**Logo**

[Add Project](#)

## Projects

Project	Positive Votes	Votes	Date	Action
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Cool tuts	0%	0	June 15, 2021, 10:04 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>
Second project	0%	0	June 15, 2021, 10:05 p.m.	<a href="#">Edit</a> <a href="#">Delete</a> <a href="#">View</a>

FOOTER

127.0.0.1:8000/.../7f1076a3-e815-43c8-8fa9-b3d5dde1bbeb/

and click on any project, in this case, it's the last project. Then after clicking on the "Delete" hyperlink, you should be able to see the delete confirmation template for the specific project you had chosen:

**Logo**

[Add Project](#)

Are you sure you want to delete "Second project"?

[Go Back](#) [Confirm](#)

FOOTER

The projects page now should look like:

Project	Positive Votes	Votes	Last updated
Ecommerce Website	30%	12	June 15, 2021, 1:24 p.m.
Portfolio Website	65%	6	June 15, 2021, 1:25 p.m.
Mumble Social Network	74%	89	June 15, 2021, 1:25 p.m.
Code Sniper	78%	80	June 15, 2021, 3:53 p.m.
Yogo Vive	98%	51	June 15, 2021, 3:53 p.m.
Cool tuts	↳ 0%	0	June 15, 2021, 10:04 p.m.

FOOTER

## Test Your CRUD

Using the newly created forms, delete a couple of projects, create and edit projects adding some kind of data that shows you that they are actually working on the database.

## Conclusion

With this, we've seen the basic forms that do basic CRUD operations on our database. We checked the files and templates necessary to implement such operations in our site.