Model Forms

What's a model form? In short, a model form is a view. A view which also uses an HTML template and that can also be stylized using CSS or JavaScript frameworks.

The main difference with previous views is that forms are used to capture manually input data from a user. They are interactive graphical user interfaces (IGUI) for data capture.

The Easy Way

Fortunatelly, there's not much to do in matter of setting up data types and entry data forms. Django does most of the work for us. However, the downside of this approach is that the styling is lacking and it's only functional. In order to improve the visuals for the forms, we need to use a styling framework such as bootstrap or doing ourselves via CSS styling on the template. For the more dedicated of us, we can also interface with other fron-end frameworks such as ReactJS.

Firs step: Build the template

Open up your /projects/templates/projects (\projects\templates\projects if Windows) folder and create a new file project_form.html.

We'll use this template for two site pages. We'll change the functionality depending on the needs of each page.

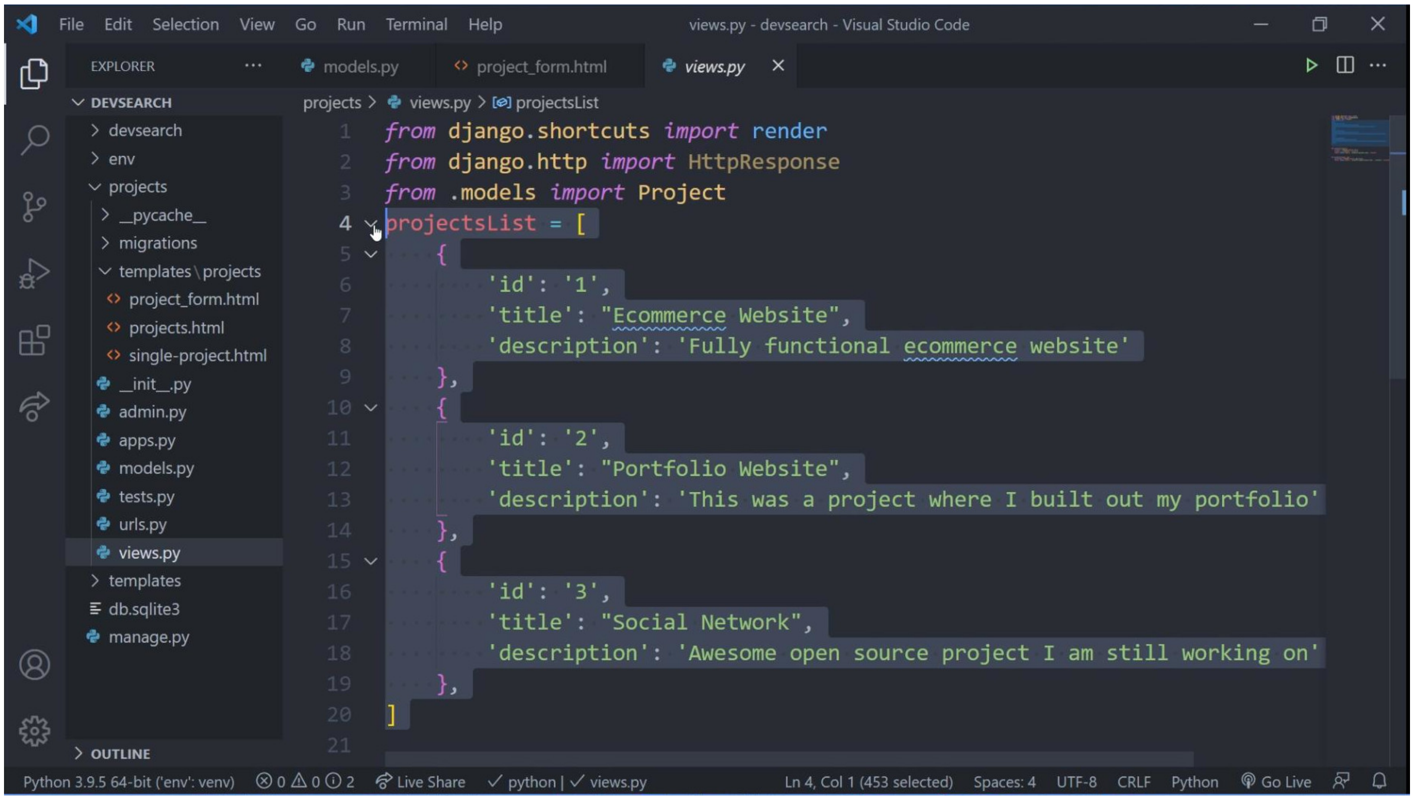
This is the first form structure we can use. Type the code:

```
<!-- \projects\templates\projects\project_form.html --->
{% extends 'main.html' %}

{% block content %}

<h1>Project Form</h1>

<!-- Declare the form here -->

<!-- The method="POST" is required on every form when data is being sent to
the view -->
<form method="POST">

    <!-- Every single page requires a CSRF token to avoid form hacking -->
    {% csrf_token %}

    <input type="submit">
```

```
</form>

{% endblock %}
```

This is our very basic form.

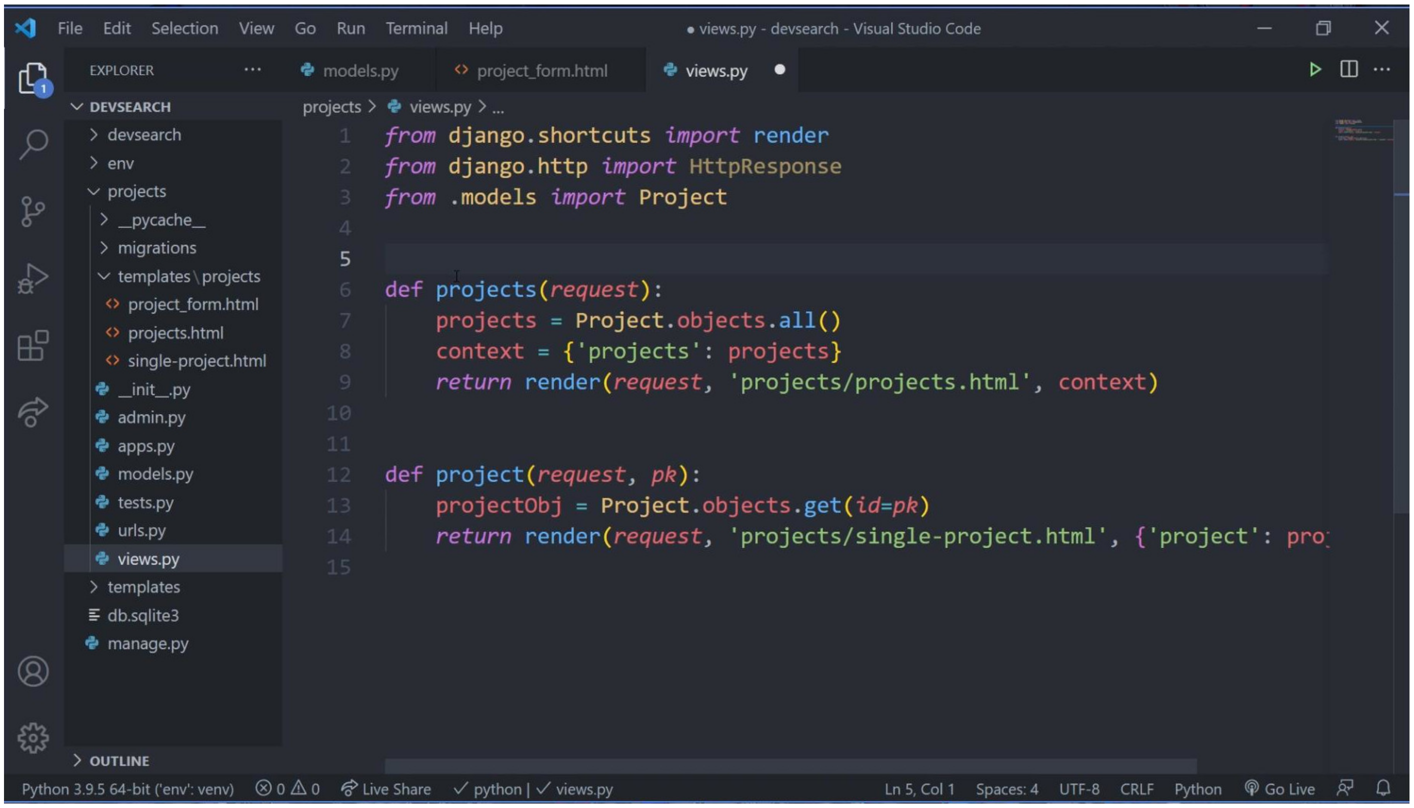
Next, let's create our view for the form. You should see something like this:



```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Project
projectsList = [
    {
        'id': '1',
        'title': "Ecommerce Website",
        'description': 'Fully functional ecommerce website'
    },
    {
        'id': '2',
        'title': "Portfolio Website",
        'description': 'This was a project where I built out my portfolio'
    },
    {
        'id': '3',
        'title': "Social Network",
        'description': 'Awesome open source project I am still working on'
    },
]
```

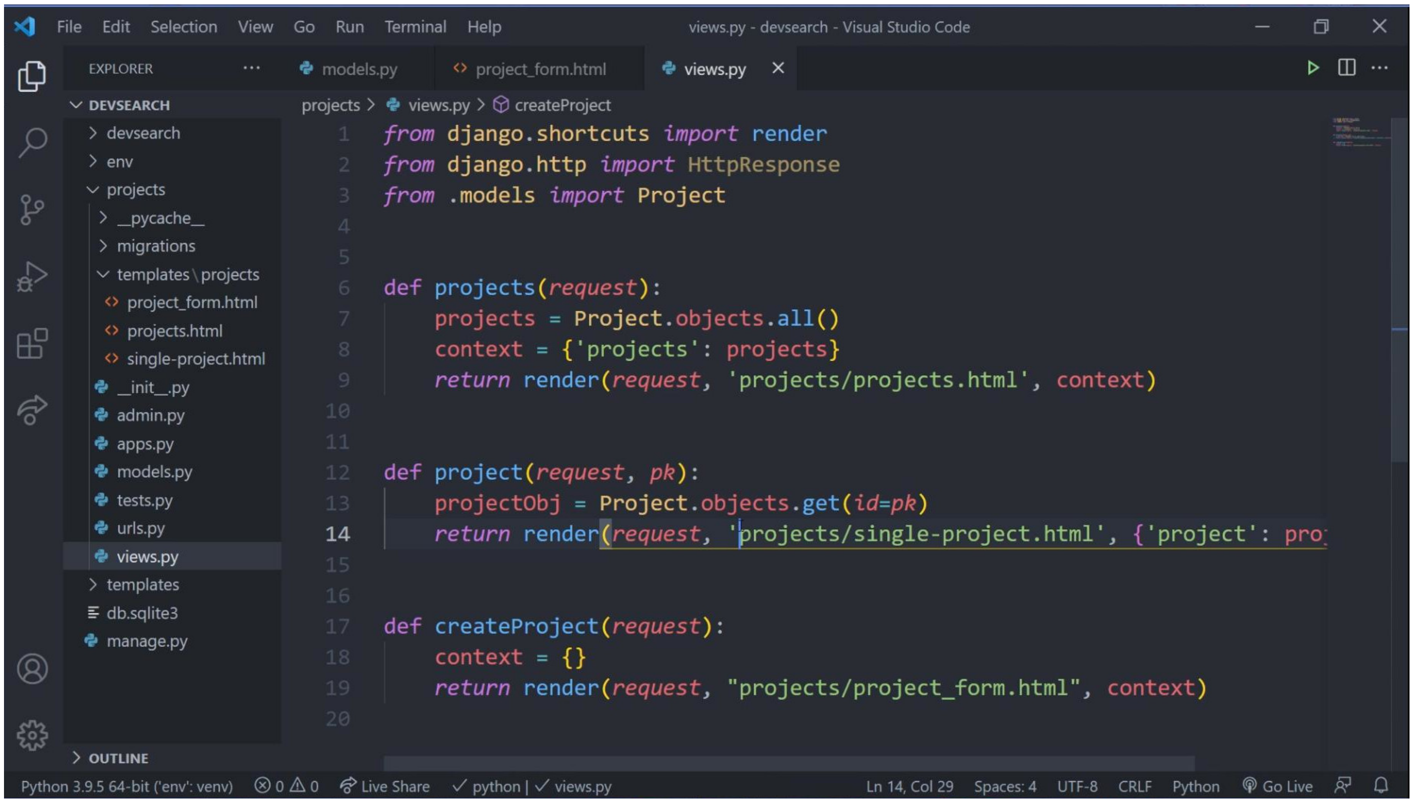
-Delete the projectList object since we're no longer using it because we already have a database working.

It should look now like this:



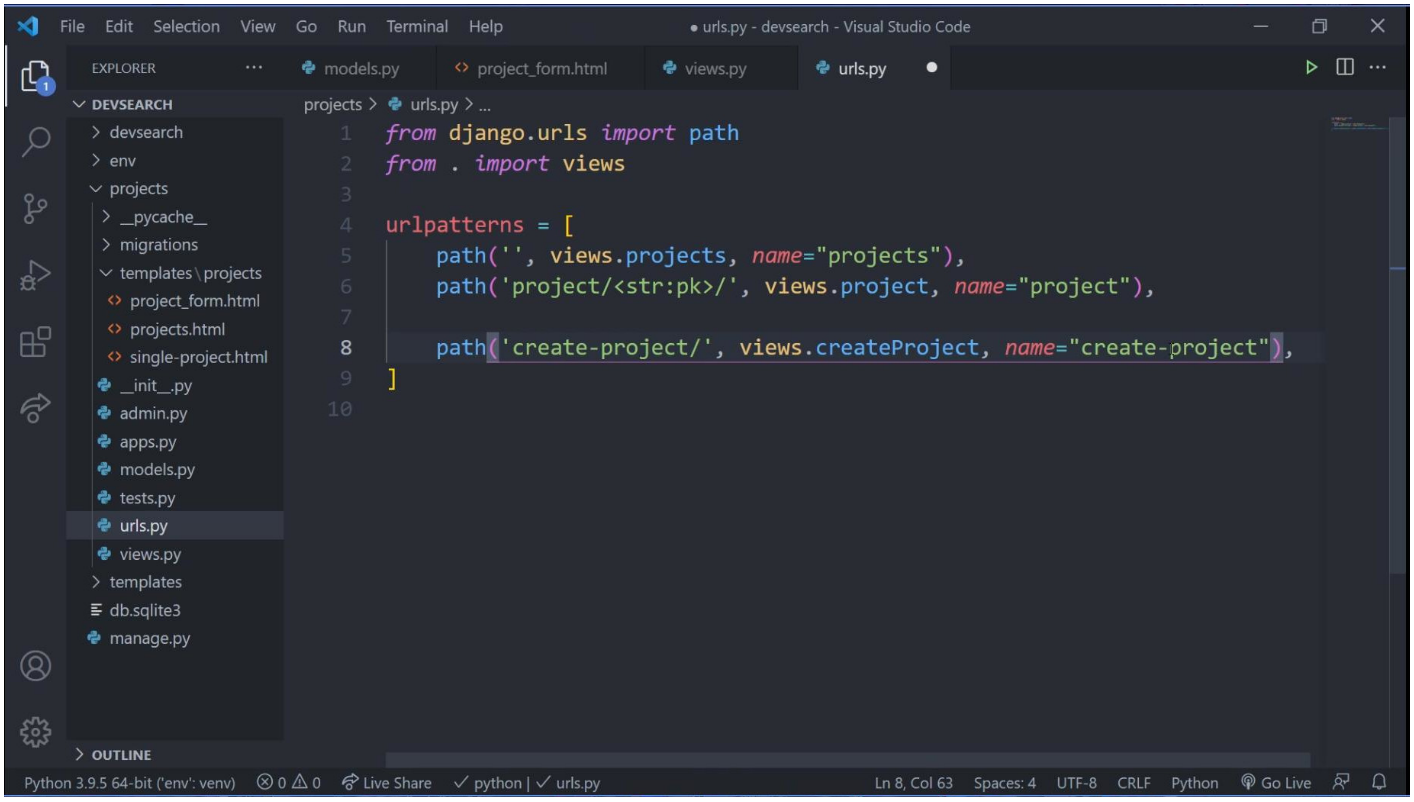
```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Project


def projects(request):
    projects = Project.objects.all()
    context = {'projects': projects}
    return render(request, 'projects/projects.html', context)


def project(request, pk):
    projectObj = Project.objects.get(id=pk)
    return render(request, 'projects/single-project.html', {'project': pro
```

Now, let's create a new view called createProject:



```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Project


def projects(request):
    projects = Project.objects.all()
    context = {'projects': projects}
    return render(request, 'projects/projects.html', context)


def project(request, pk):
    projectObj = Project.objects.get(id=pk)
    return render(request, 'projects/single-project.html', {'project': pro


def createProject(request):
    context = {}
    return render(request, "projects/project_form.html", context)
```

The following step is to activate the view by assigning a route to it: - Open /projects/urls.py - Assign the route to the new form view.


File Edit Selection View Go Run Terminal Help
● urls.py - devsearch - Visual Studio Code
EXPLORER
models.py project_form.html views.py urls.py
DEVSEARCH
devsearch
env
projects
pycache
migrations
templates \ projects
project_form.html
projects.html
single-project.html
init.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py
templates
db.sqlite3
manage.py
OUTLINE
projects > urls.py > ...
Python 3.9.5 64-bit ('env': venv) ⊗ 0 ⚠ 0 Live Share ✓ python | ✓ urls.py Ln 8, Col 63 Spaces: 4 UTF-8 CRLF Python Go Live

```python
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"),
]
```

- The new line is

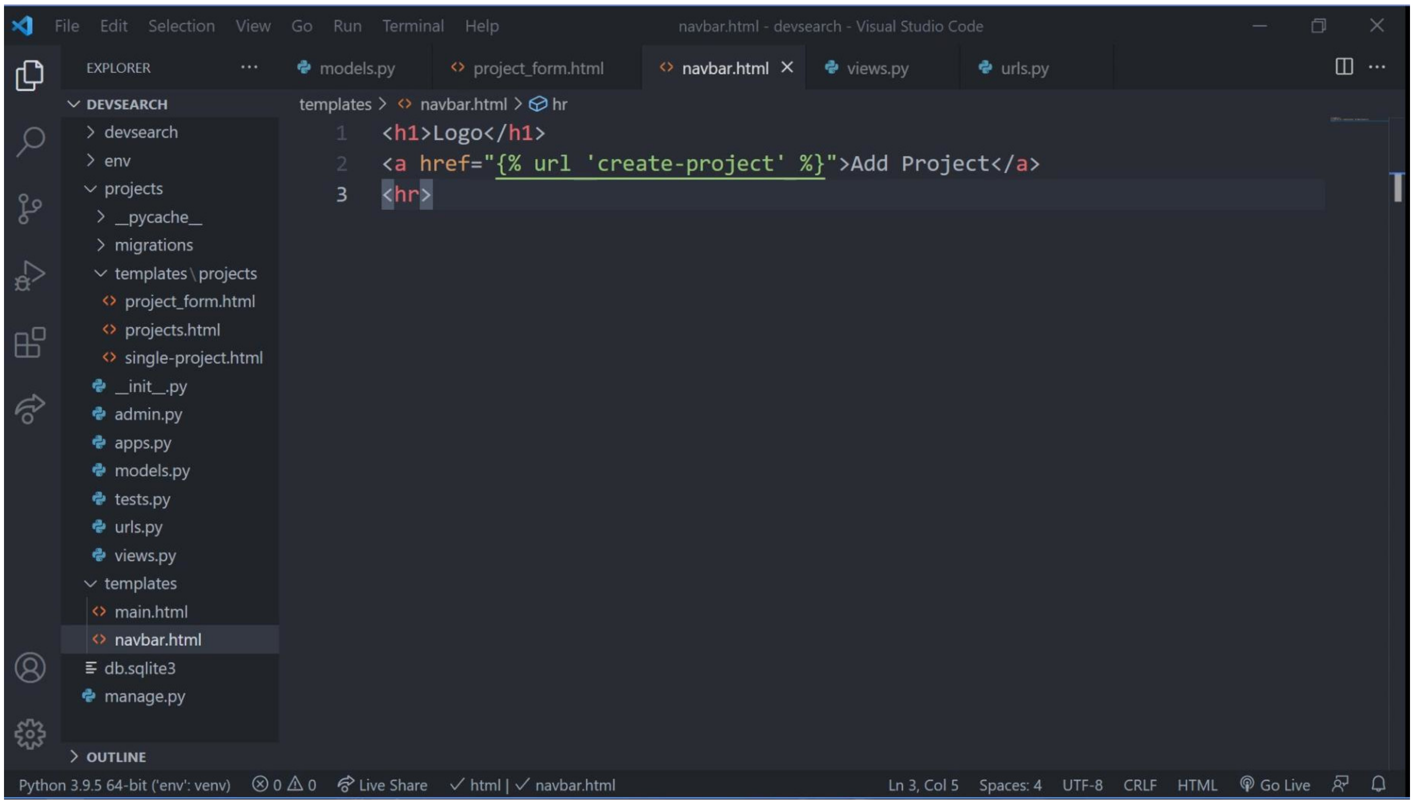
```
    path('create-project/', views.createProject, name="create-project"),
```

the function structure is:

```
path(new_route, rendering_function, view_assigned_name),
```

We're almost done with the form set up steps. We're still missing a way for the site to access our form. For simplicity in this case, we're going to use the NavBar to access the form (this is applicable for the time being, there other ways to do this.)

To add the form to the Navbar template, open up the general django project navbar.html file in /templates/navbar.html



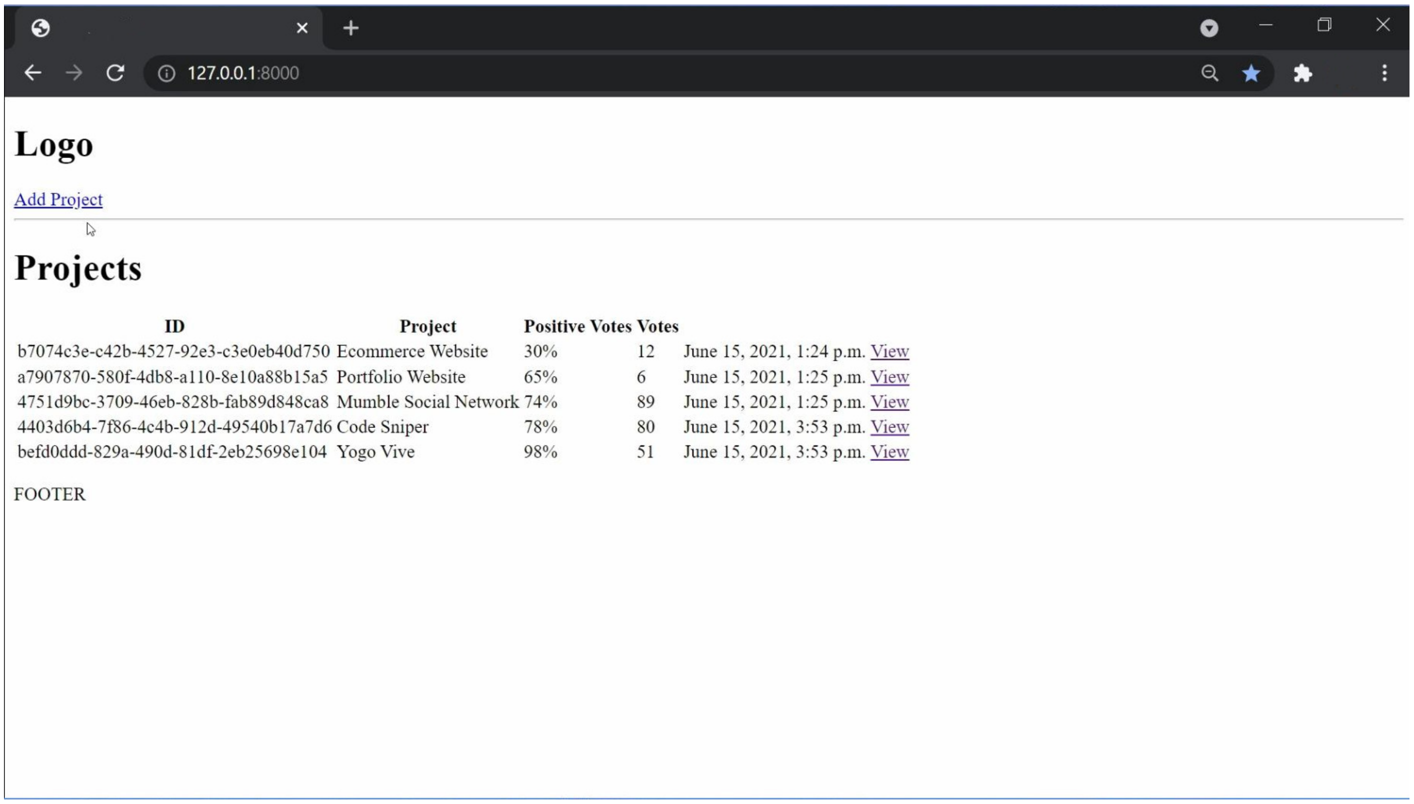
```
<h1>Logo</h1>
<a href="{% url 'create-project' %}">Add Project</a>
<hr>
```

Note: Please notice that the string 'create-project' is exactly de same as the string assigned to the parameter ***view_assigned_name***. This is the dynamic name assignation for the routes. We're using an html anchor:

```
<a href="{% url 'create-project' %}">Add Project</a>
```

To attach the insertion form to the navbar.

To test the insertion form open up your browser to acces the main page, that should be set up to the projects view for the time being:



127.0.0.1:8000

Logo

Add Project

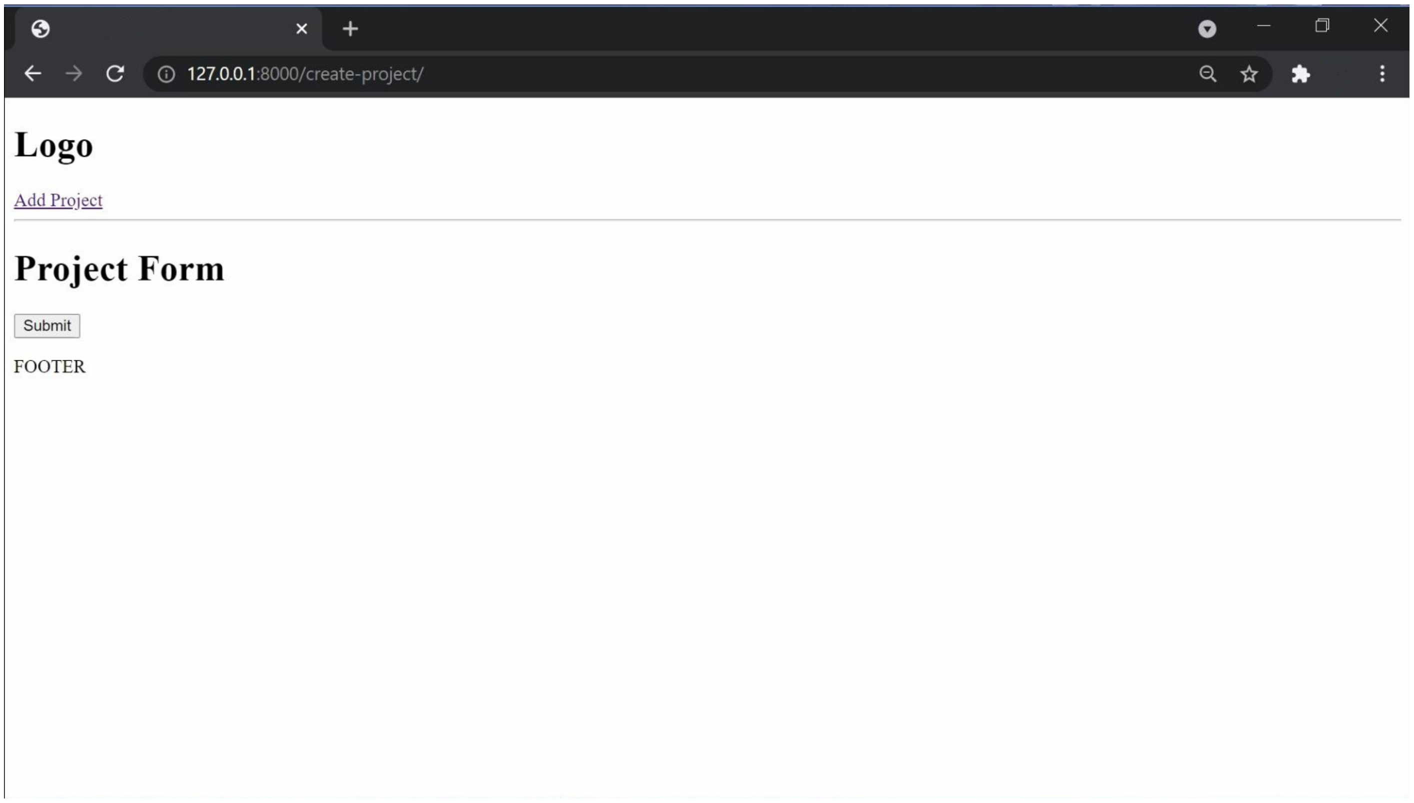
Projects

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

FOOTER

you should see the hyperlink "Add Project" in the navbar area.

Click on the hyperlink and you should see the form view:



127.0.0.1:8000/create-project/

Logo

Add Project

Project Form

Submit

FOOTER

As you can see, the form right now is very basic and it doesn't contain the fields neccesary to create a new project. We'll add the fields next.

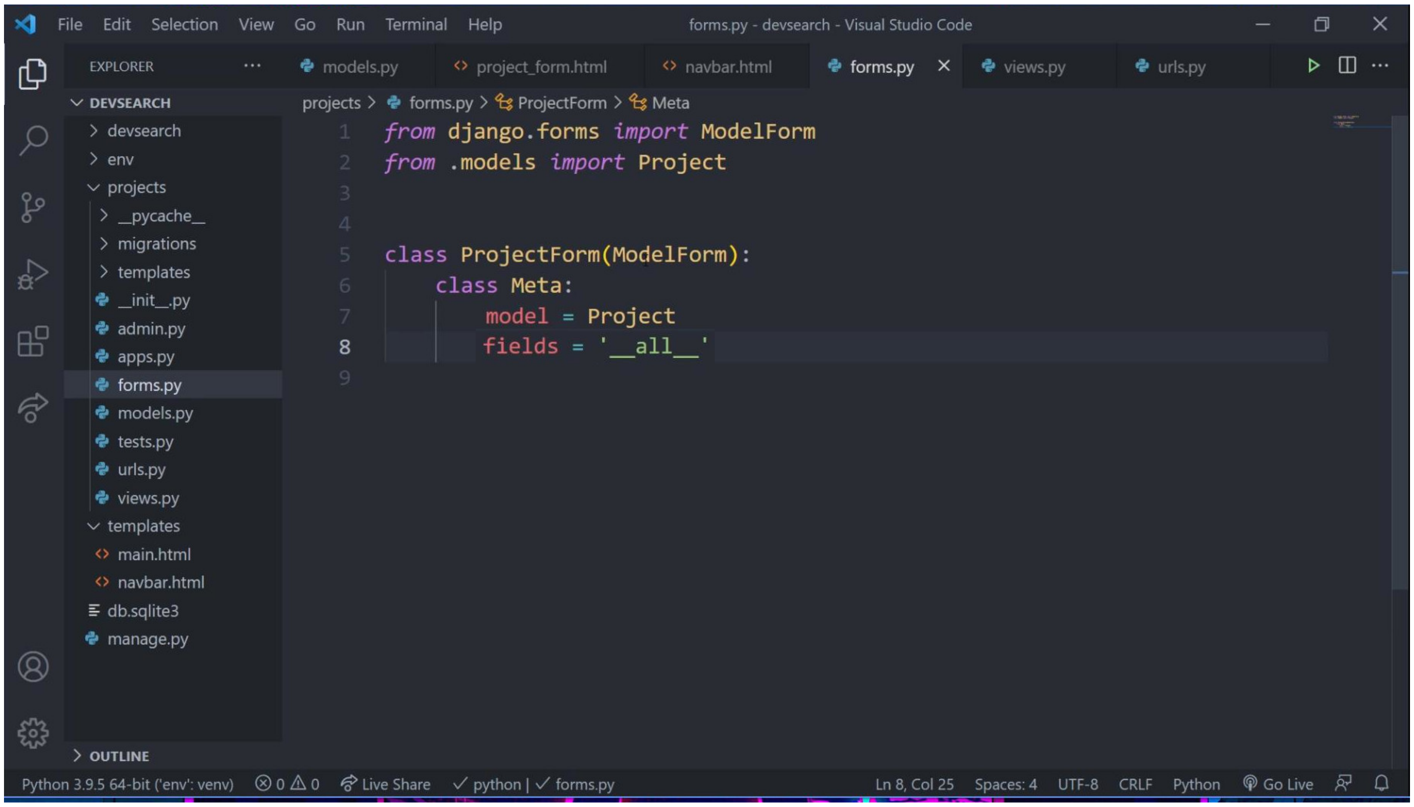
Add the Fields to the Form

In order to include the fields, we need to "specialize" our form, that is, we will subclass the generic form for the specific model we want to insert its data.

Again, we could manually include the necesary fields within the specialized form view, but there's an easier way to do that directly from our projects model.

Let's create a new file in our projects application. Specifically, create the file /projects/forms.py. This file we'll create all the forms for the django application, we won't create the forms for the django project. Each django application should contain its own forms.py file.

The file should look like:



```python
from django.forms import ModelForm
from .models import Project


class ProjectForm(ModelForm):
    class Meta:
        model = Project
        fields = '__all__'
```

Explanation:

What we are doing here is linking two modules: django.forms with our model definition.

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

ModelForm is a base class that contains all the field descriptions necessary for data input based on the fields we used on our database model definition.

The way we create the functionality between the model and the ModelForm class is through the subclassing of the Meta class included in our ProjectForm(ModelForm) subclass.

The two property members necessary to describe our model to the form are:

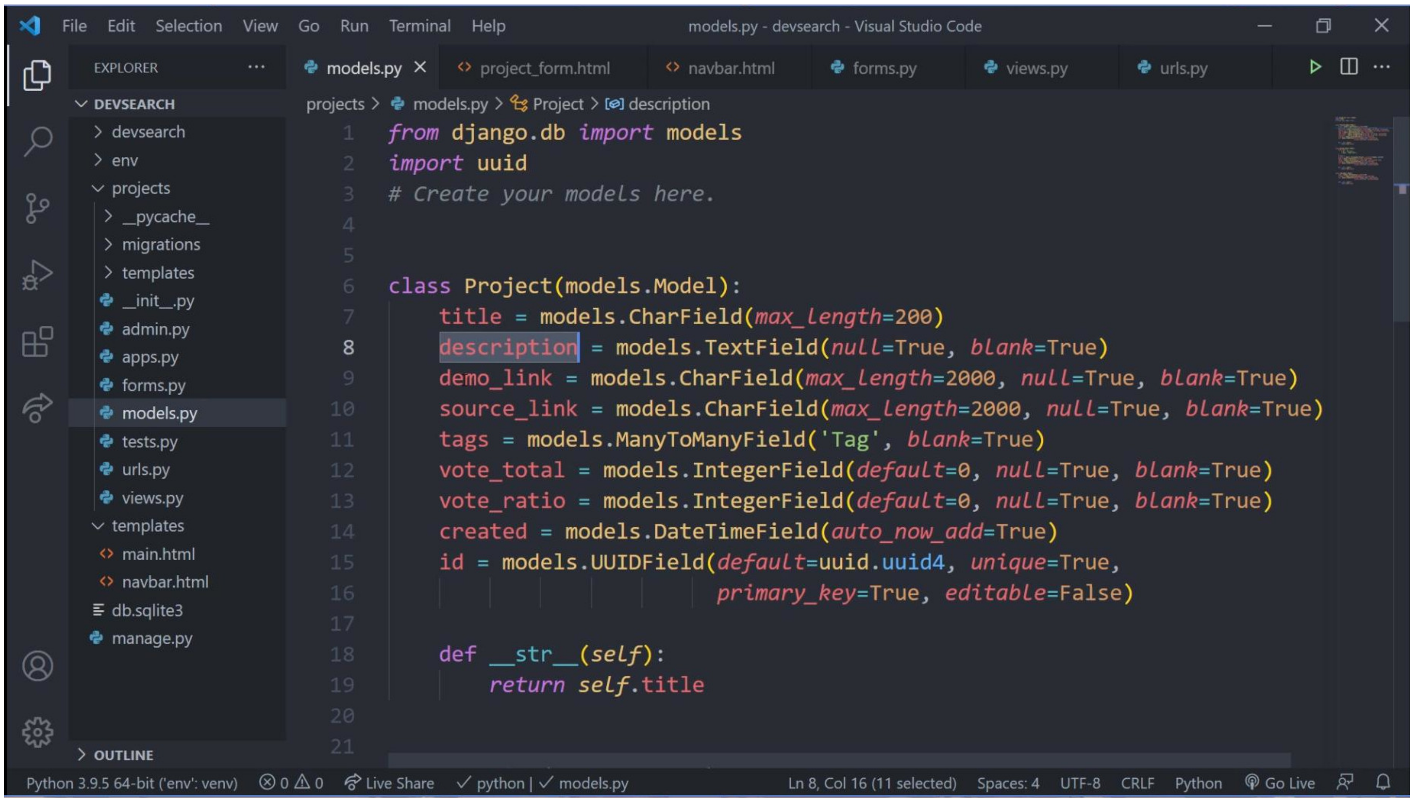
```
model = Project
fields = '__all__'
```

The parameter *field* with argument '__all__' indicates to the Meta class that it should use all the fields defined in the model with *name* = Project

If we don't want to use all the fields defined in our model, then we have to replace the argument '__all__' with a list containing all the specific field names we want to use:

```
model = Project
fields = ['field1', 'field2', ...]
```

So if our model looks like:



```
from django.db import models
import uuid
# Create your models here.


class Project(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField(null=True, blank=True)
    demo_link = models.CharField(max_length=2000, null=True, blank=True)
    source_link = models.CharField(max_length=2000, null=True, blank=True)
    tags = models.ManyToManyField('Tag', blank=True)
    vote_total = models.IntegerField(default=0, null=True, blank=True)
    vote_ratio = models.IntegerField(default=0, null=True, blank=True)
    created = models.DateTimeField(auto_now_add=True)
    id = models.UUIDField(default=uuid.uuid4, unique=True,
                          primary_key=True, editable=False)

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

We can very well choose any field:

```
model = Project
fields = ['title', 'description', 'demo_link', ...]
```

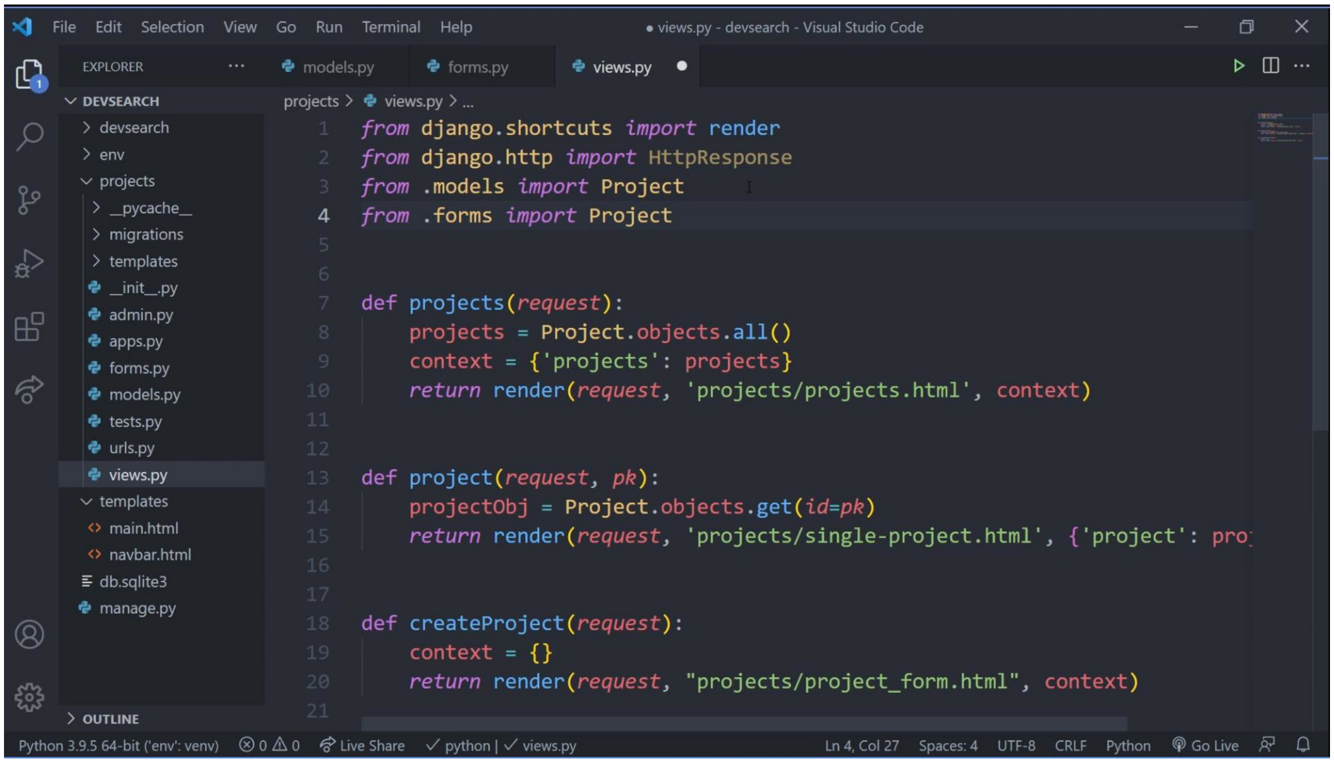
However, we can only choose those field that are not automatically generated, i. e., fields such as *'id'* or *'created'* won't be included in either '__all__' or within our specific list.

But since we're going to use '__all__', django will generate all the fields in the form that are included into our model definition.

How To Use Our Custom Form

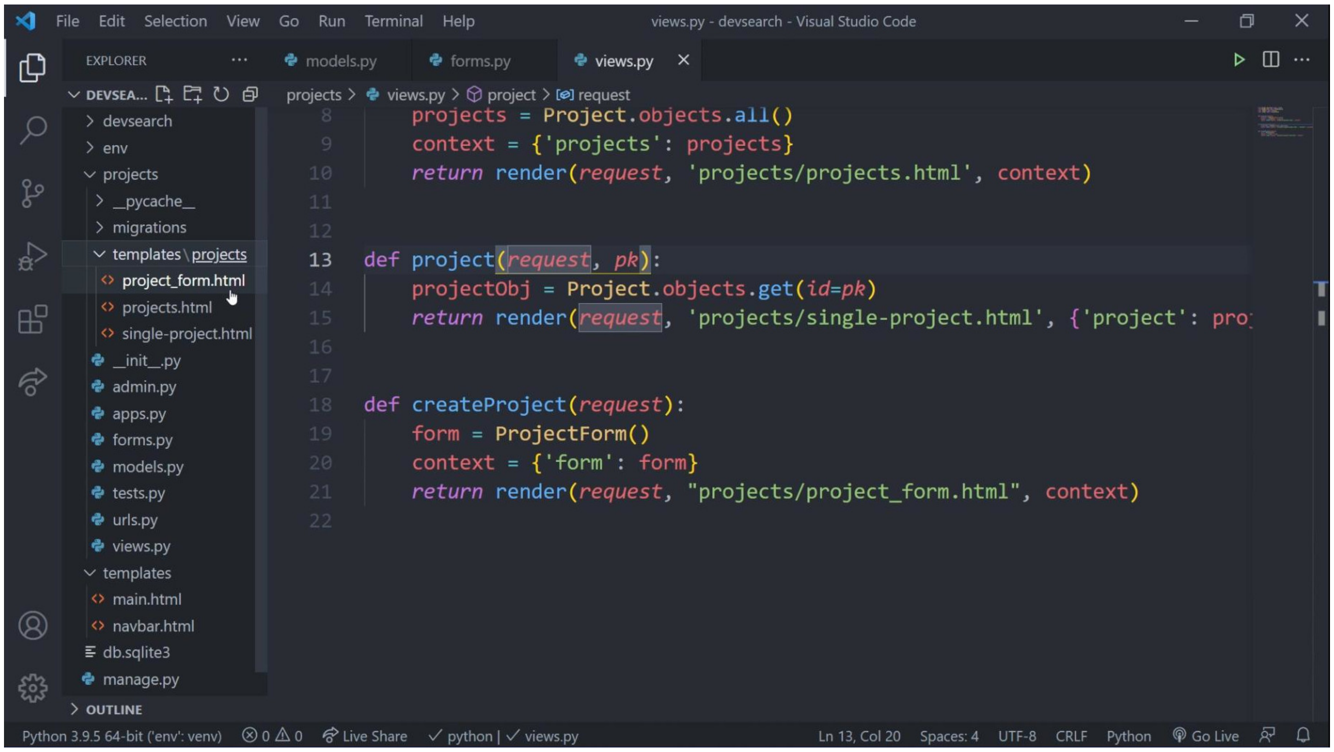
Now, in order to use our custom form, we need to include the form definition in our django application's views.py. Open up the file /projects/views.py and modify the views.py file as follows:

1. import our Project model into the views.py file:



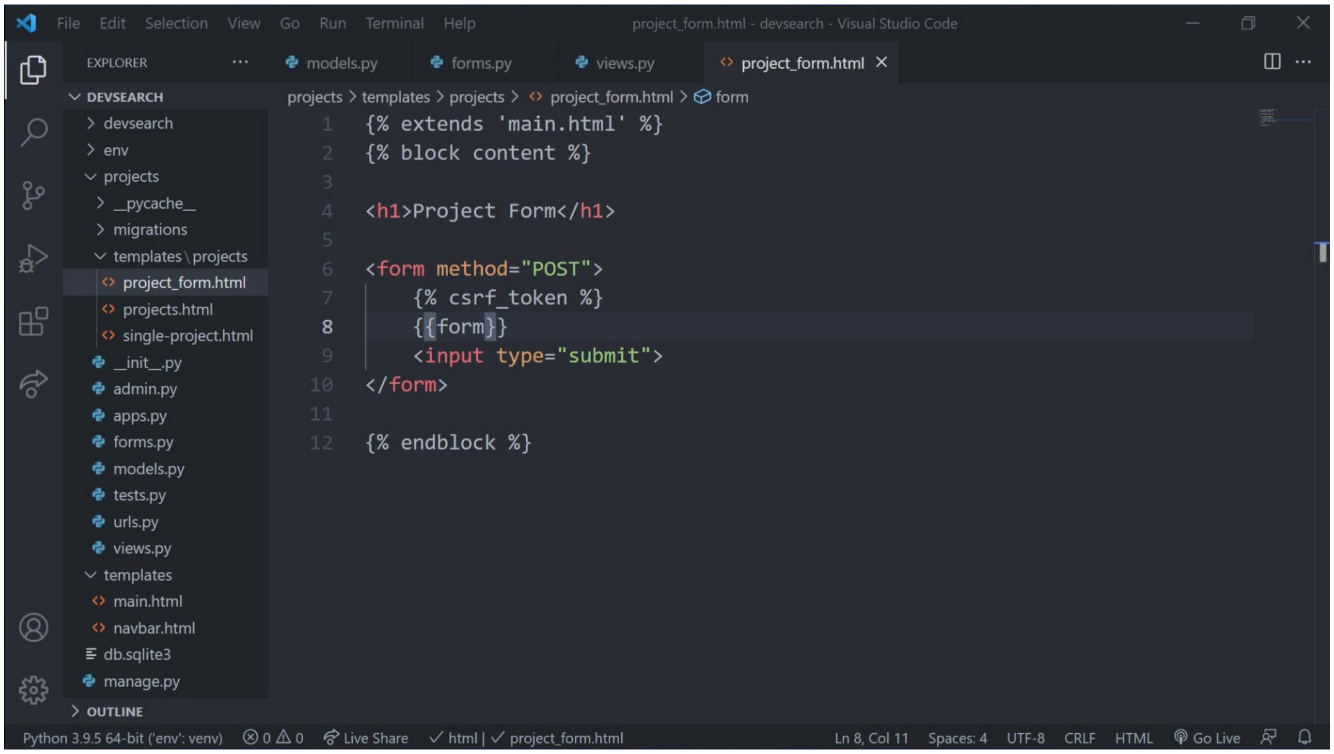
```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Project
from .forms import Project


def projects(request):
    projects = Project.objects.all()
    context = {'projects': projects}
    return render(request, 'projects/projects.html', context)


def project(request, pk):
    projectObj = Project.objects.get(id=pk)
    return render(request, 'projects/single-project.html', {'project': pro


def createProject(request):
    context = {}
    return render(request, "projects/project_form.html", context)
```

2. modify the function createProject to use the form and add the form to the context dictionary



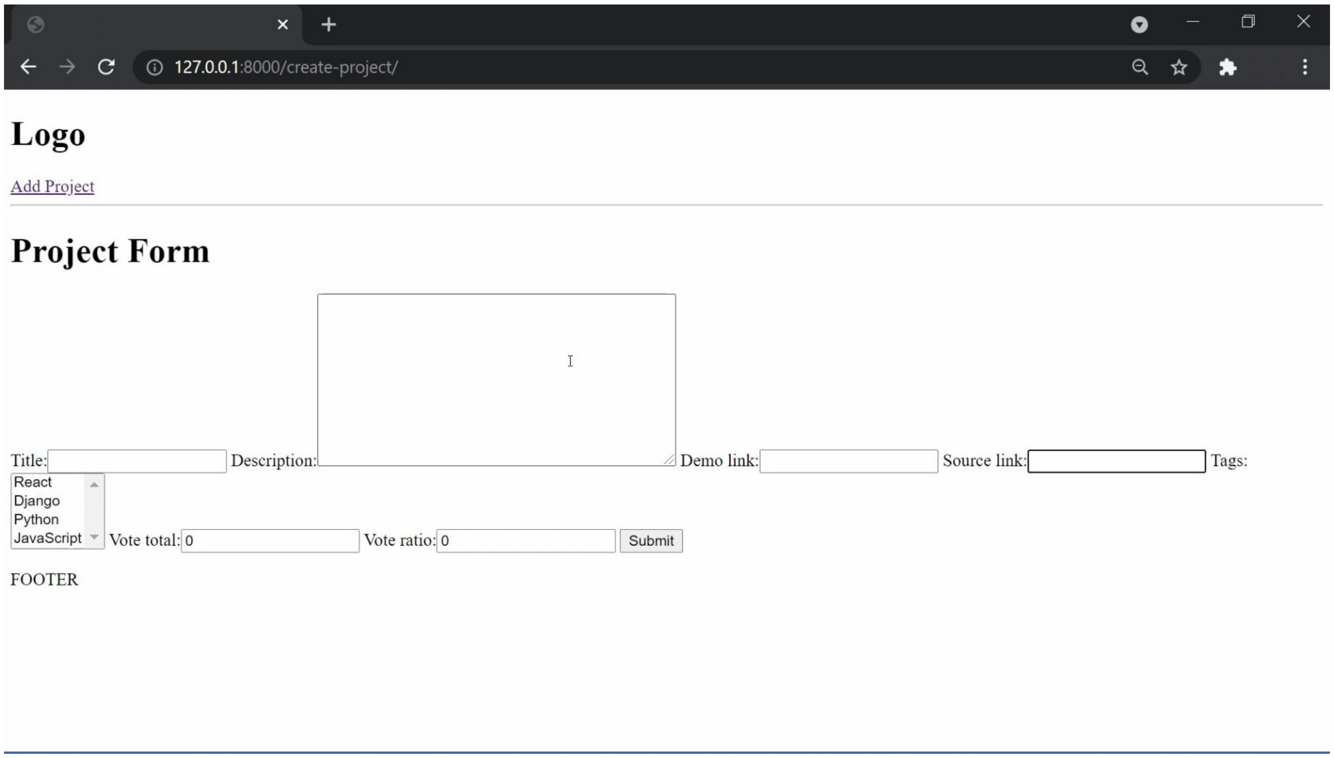
```python
    projects = Project.objects.all()
    context = {'projects': projects}
    return render(request, 'projects/projects.html', context)


def project(request, pk):
    projectObj = Project.objects.get(id=pk)
    return render(request, 'projects/single-project.html', {'project': pro


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

3. Again, we need to modify our template project_form.html in order to catch the *form* variable from our views.createProject function:



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

<h1>Project Form</h1>

<form method="POST">
    {% csrf_token %}
    {{form}}
    <input type="submit">
</form>

{% endblock %}
```

4. Try the form by accessing the form view via your web browser

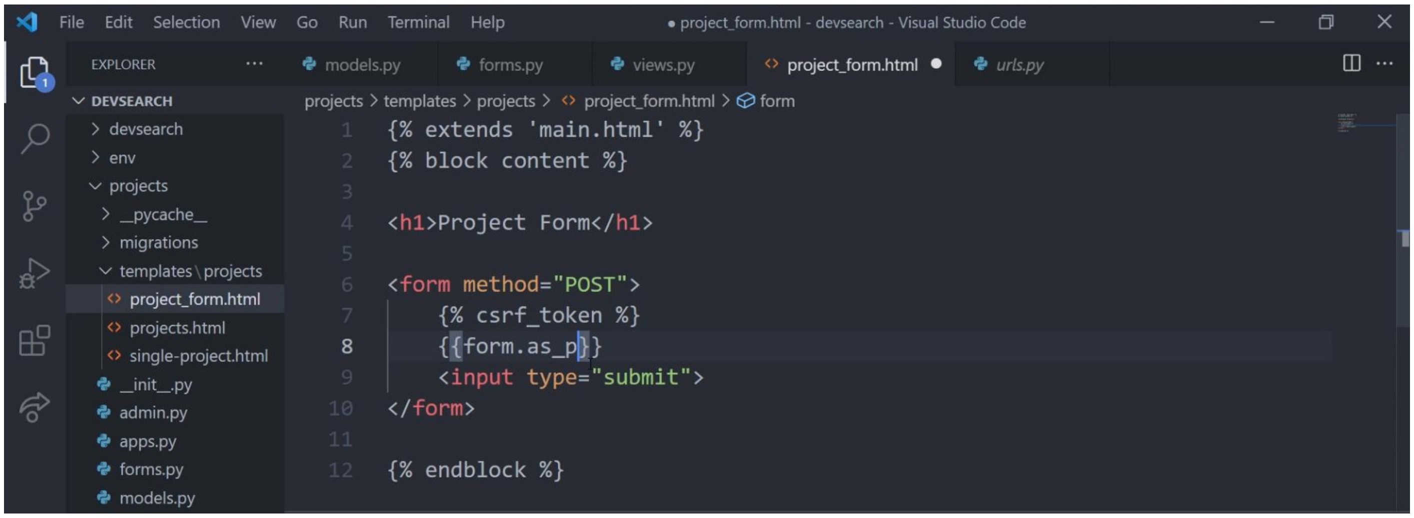


As you can see, the form is not visually appealing. This will change in the future, but for now is more than enough to test the functionality.

A Few Tricks

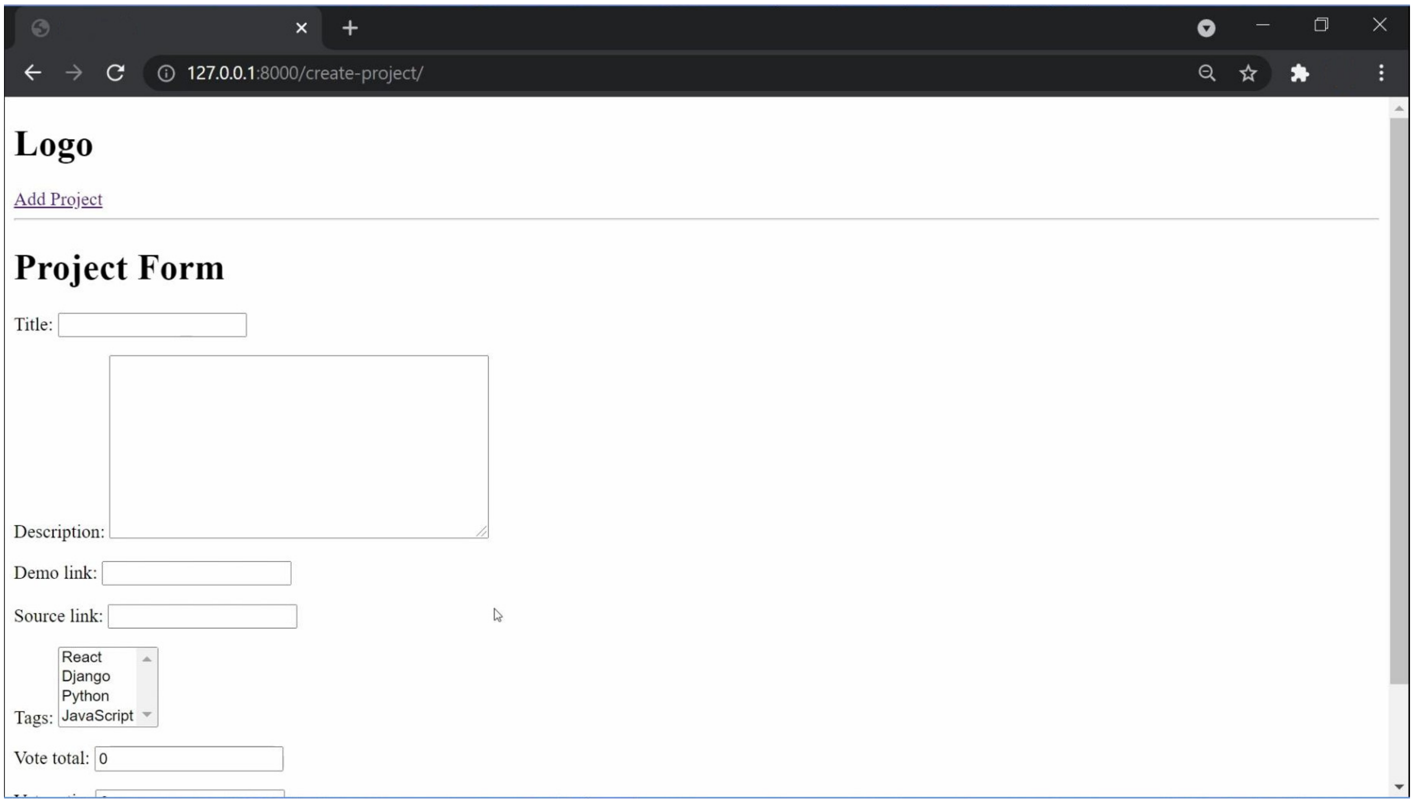
Though we are not styling the form right now, it's possible to improve the visuals with a few simple HTML tricks that are included with the JINJA template language.

The first one is to use {{ variable.as_p }}:



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

<h1>Project Form</h1>

<form method="POST">
    {% csrf_token %}
    {{form.as_p}}
    <input type="submit">
</form>

{% endblock %}
```

This will wrap all the form fields into a <p></p> tag which will improve the visuals a little bit.



Project Form

Title:

Description:

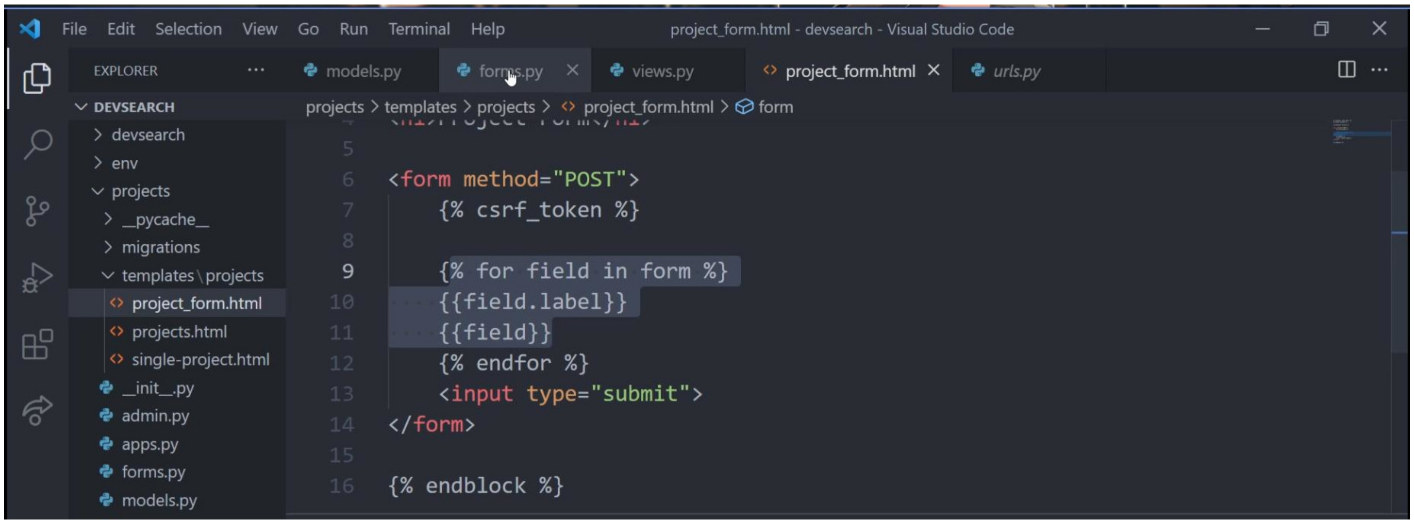
Demo link:

Source link:

Tags: React Django Python JavaScript

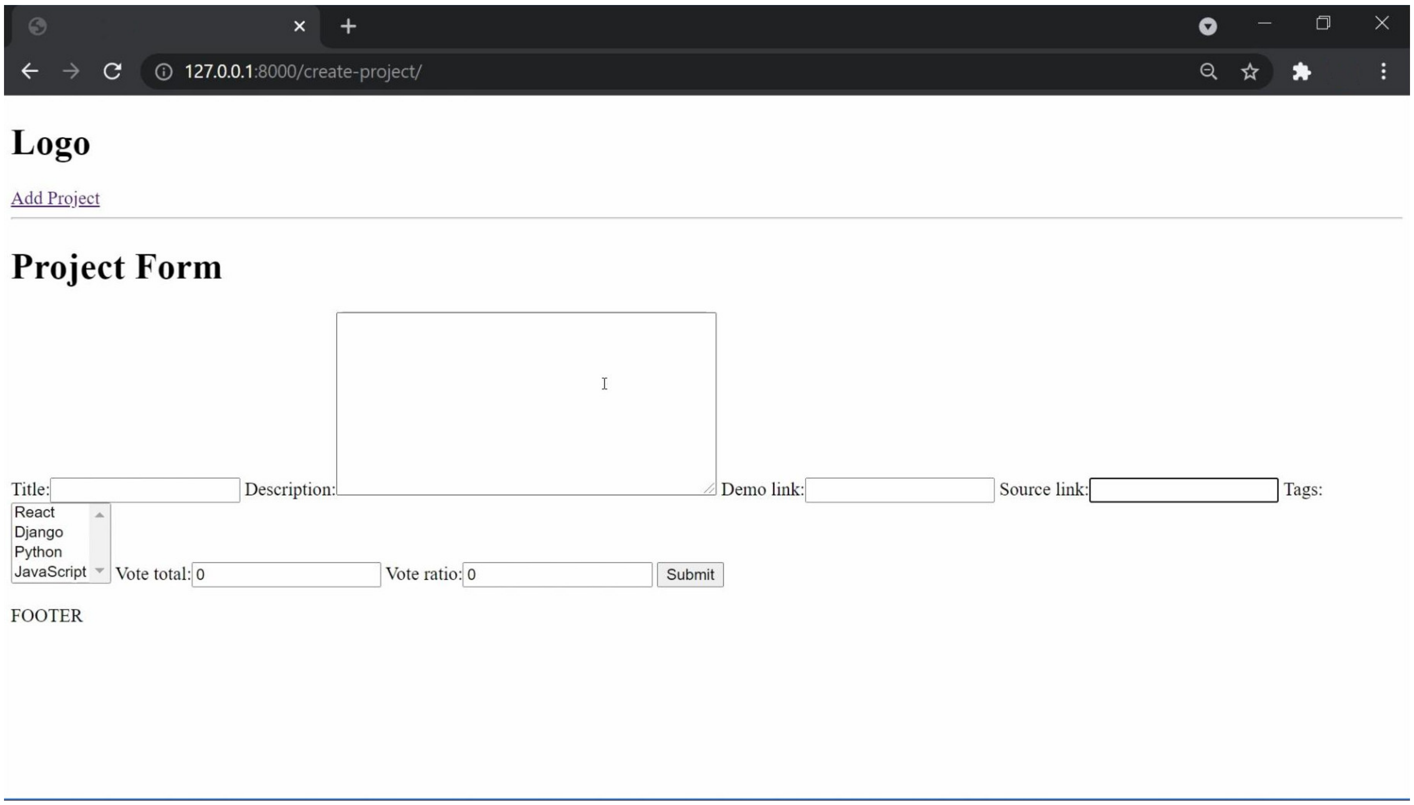
Vote total: 0

Another option is to output each field individually by modifying the project_form.html with a for-loop:



```
<form method="POST">
    {% csrf_token %}

    {% for field in form %}
    {{field.label}}
    {{field}}
    {% endfor %}
    <input type="submit">
</form>

{% endblock %}
```

which will bring us back to an unstyled page:



Logo

Add Project

Project Form

Title: Description: Demo link: Source link: Tags:

React
Django
Python
JavaScript

Vote total: 0 Vote ratio: 0 Submit

FOOTER

However, we could re-style the for-loop by adding <p>...</p> tags or <div>...</div> around the field label and field: <div>{{ field.label }} {{ field }}</div>

Strong Suggestion:

Check the documentation on Django Forms at Django Forms at Mozilla and Django Forms Documentation

Next CRUD Operations and Forms