Queries in Views

As we've seen before, views have two main components:

1. The views.py file which has all the mechanisms necessary to provide a response to a request, i. e., the functions that discriminate and provide adequate response to any response related to the specific slug. Usually, when presenting information, this requires rendering a template with dynamic variables.
2. The template which is a skeleton in charge of creating the basic scafolding for presenting data as in response to a request.

Database queries can be implemented in either of these two components. Not all queries must be placed on the views.py file but also they can be placed as part of the template scafolding in some places using the available Jinja mechanics we've touch on previous guides.

We're going to revisit our view.py and templates in the projects application in order to have a better understanding of the basic mechanisms available to us when programming our Django applications.

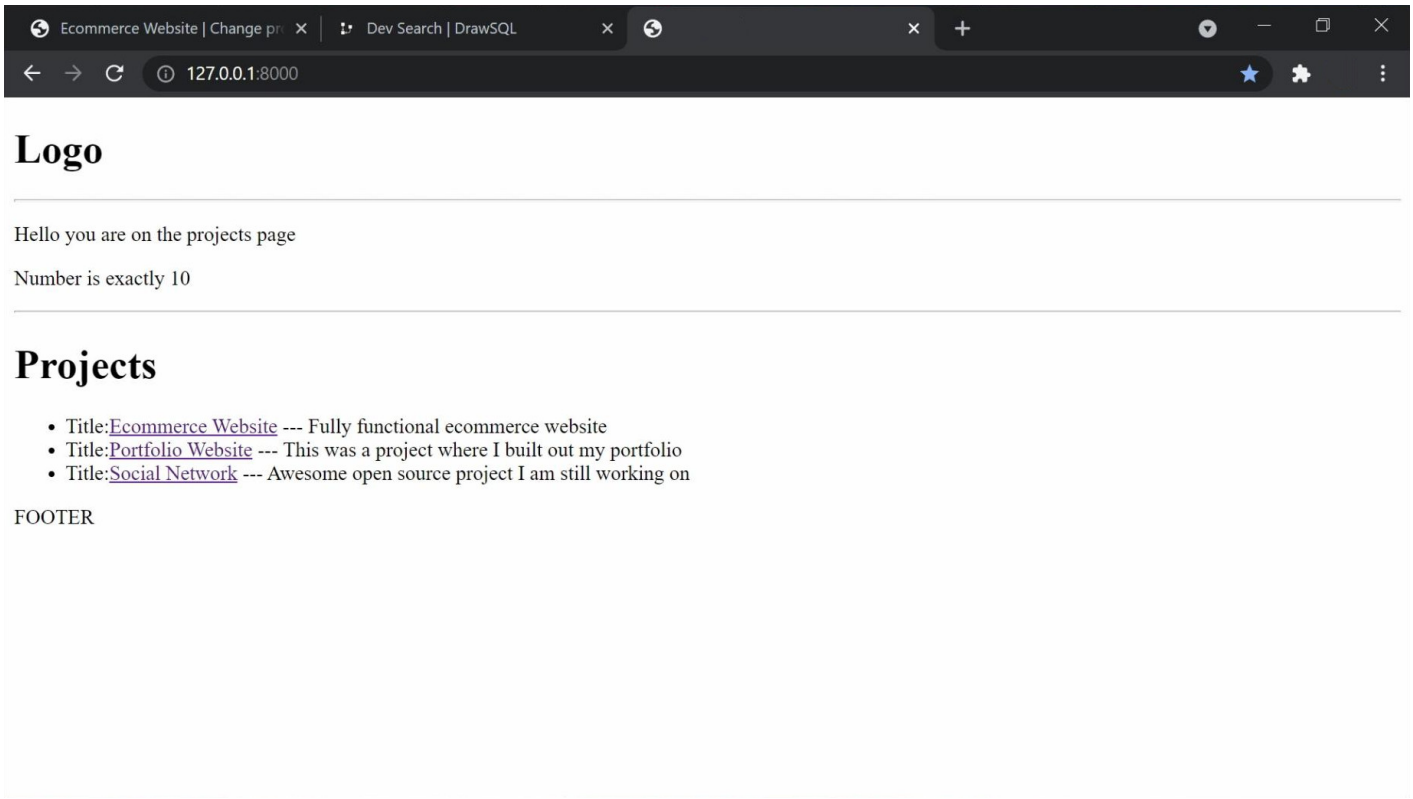
Revisiting our projects/views.py and templates/project.html and template/single-project.html

Since we haven't done any major modification to our urls.py in the main project folder devsite/urls.py since we turned the project/views.py as the main point for the site (the equivalent to index/).

Let's first start the server:

```
python manage.py runserver
```

access the homepage at 127.0.0.1:8000 and we should be able to see:



if we haven't change any settings in the corresponding urls.py files.

If you remember, this is not a dynamic view but a static one. By checking the file project/views.py contains a list object:

```
from django.shortcuts import render
from django.http import HttpResponse


# This was our initial database when we didn't have a database.
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': 'Awsome open source project I am still working on'
    },
```

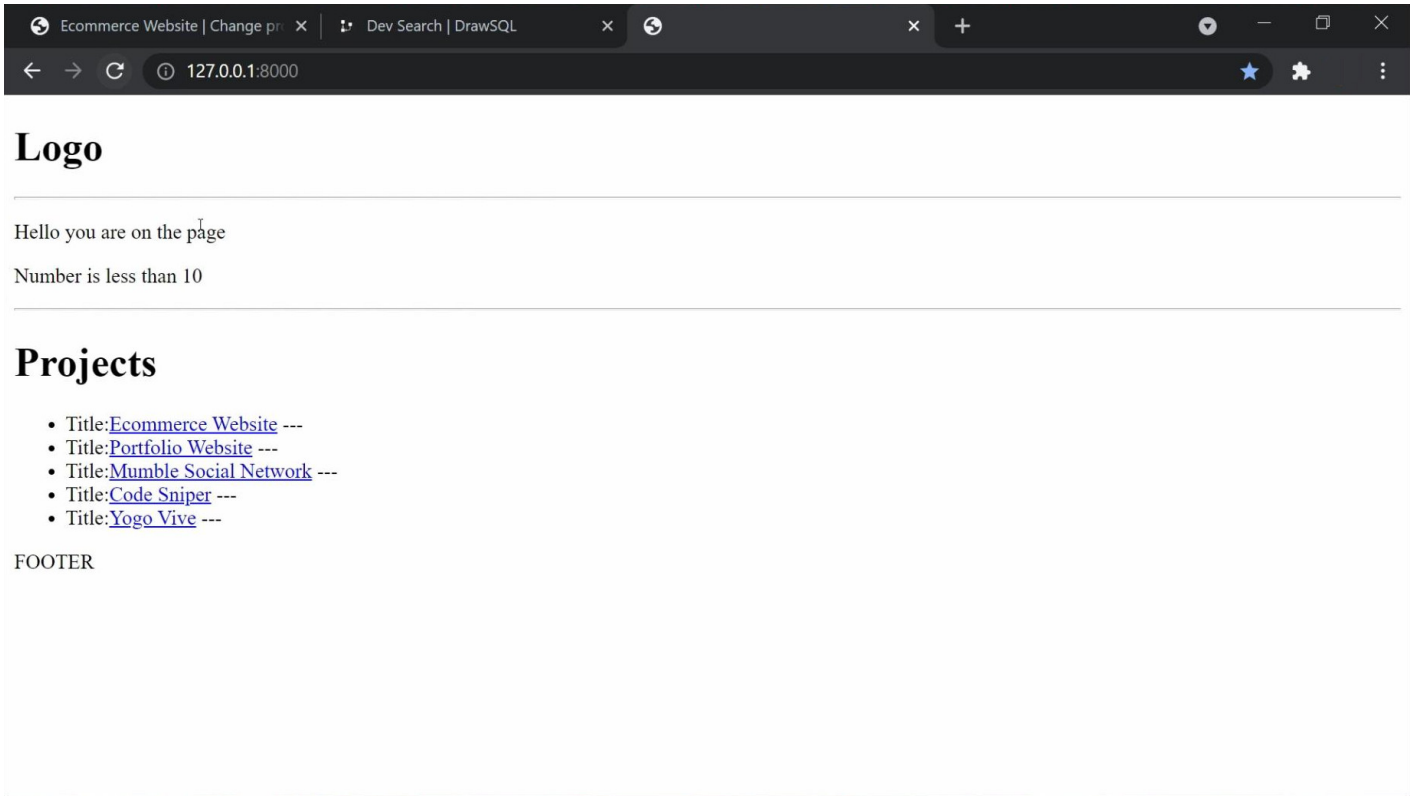
```
]

def project(request):
    page = 'projects'
    number = 10
    context = {'page': page, 'number': number, 'projects': projectsList}
    return render(request, 'projects/projects.html', context)

def project(request, pk):
    projectObj = None
    ids = [project.id for project in projectList]
    if pk not in ids:
        raise ValueError(f'Primary key: {pk} is not in the database')
    projectObj = projectList[ids.index(pk)]
    return render(request, 'projects/single-project.html', {'project':
projectObj})
```

We'll modify the file so we now bring up all the data from the database.

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

# This was our initial database when we didn't have a database.
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': 'Awsome open source project I am still working on'
    },
]

def project(request):
    # Query all the projects
    projects = Project.objects.all()
    # Change the context to send the list reference
```

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

# Do not modify yet
def project(request, pk):
    projectObj = None
    ids = [project.id for project in projectList]
    if pk not in ids:
        raise ValueError(f'Primary key: {pk} is not in the database')
    projectObj = projectList[ids.index(pk)]
    return render(request, 'projects/single-project.html', {'project':
projectObj})
```

Save the file and test the changes by refreshing the page:



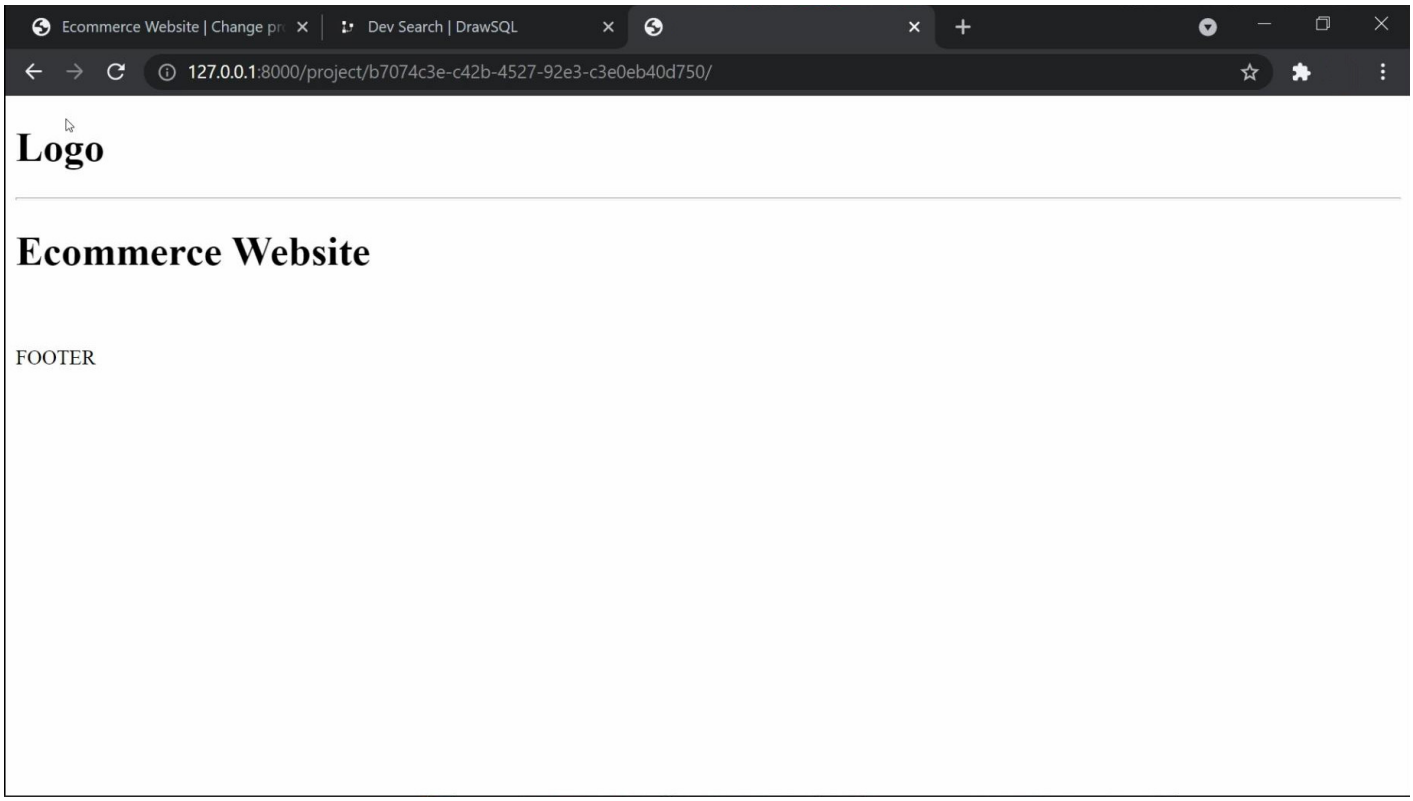
The links shouldn't work since we haven't change the project function view yet! So, even if you click on the hyperlinks, they shouldn't open any functioning view.

Return to our project/views.py file and do some more modifications:

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

# Delete the projectsList
```

```
def project(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':
projectObj})
```

and test it by clicking on the projects page (127.0.0.1:8000 for the time being). You should get single project pages like this one:



Include the tags for each project in the single-project view

Here, we're going to see that we can include queries both at the functional level (views.py) or template level (single-project.html)

Functional Level (views.py)

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

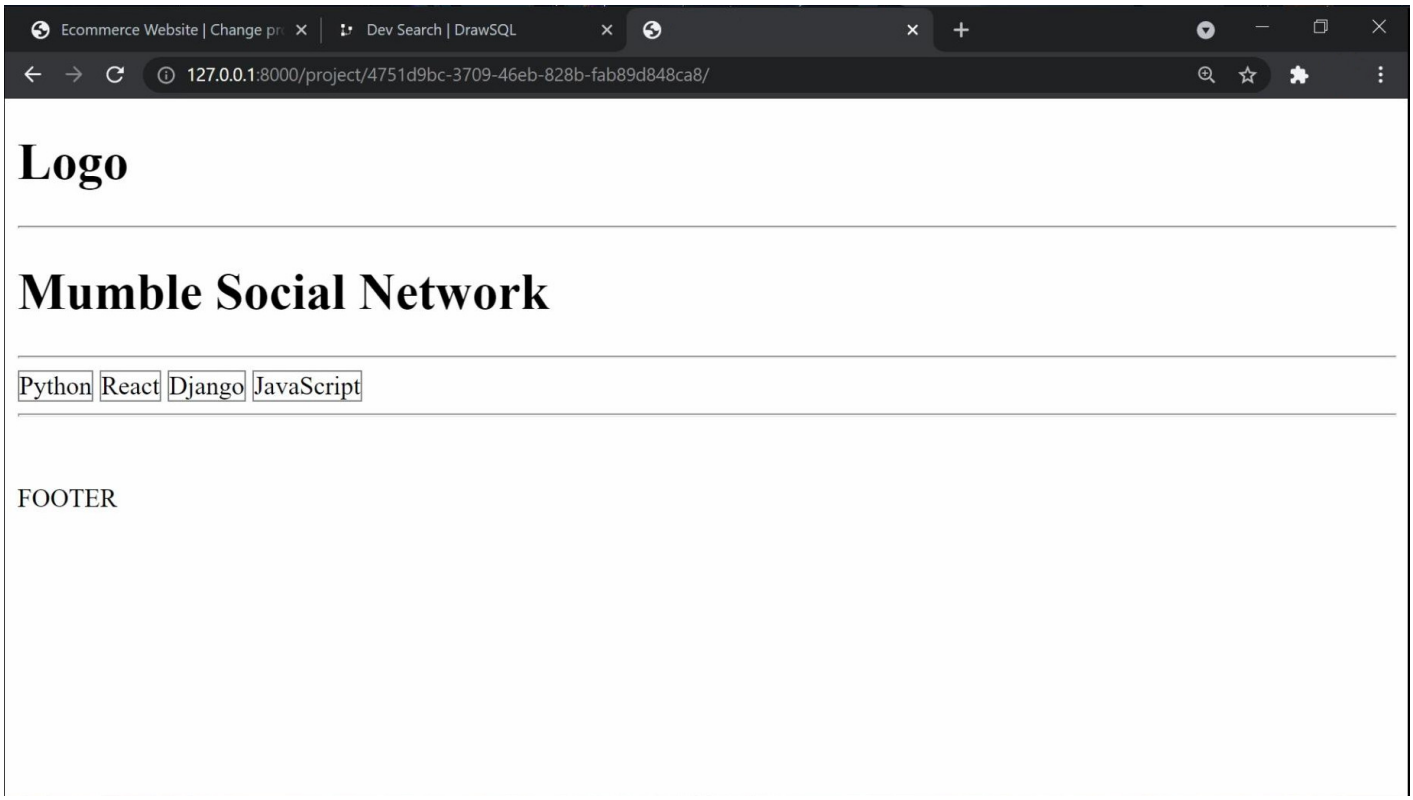
```
# Delete the projectsList

def project(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)

    # We retrieve all the tags for the project here
    tags = ProjectObj.tags.all()

    # And add the 'tags': tags to the context dictionary.
    return render(request, 'projects/single-project.html', {'project':
projectObj, 'tags': tags})
```

Now, modify the single-project.html template to include the changes to the context dictionary.

The next step is to modify the single-project.html template to accomodate the tags.

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

{% block content %}
<h1> {{ project.title }} </h1>
<hr>
{% for tag in tags %}
    <span style="border 1px solid grey"> {{ tag }} </span>
{% endfor %}
</hr>
<br>
<p> {{ project.description }} </p>
```

try it and test it at each individual project by accessing each individual link at the current main page at 127.0.0.1:8000



As you can see, we query the tags at the functional level and we pass the reference to that object to the template.

However, we can also query the tags directly from the template.

Template Level (adding the query to single-project.html)

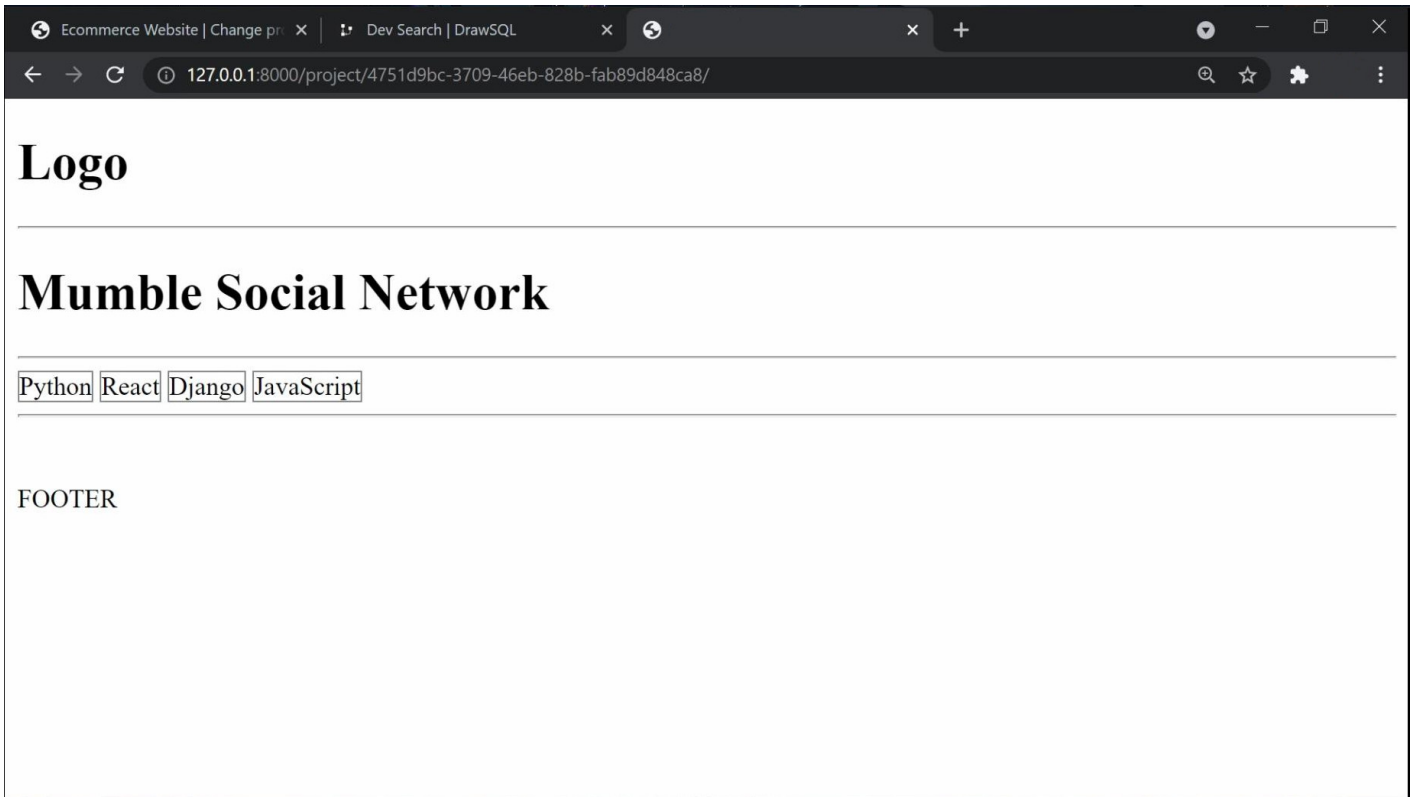
Now our project/views.py file should look like:

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


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

def project(request, pk):
    # We still need to pass the specific projectObj
    projectObj = Project.objects.get(id=pk)
    return render(request, 'projects/single-project.html', {'project':
projectObj})
```

and we modify the template single-project.html

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

{% block content %}
<h1> {{ project.title }} </h1>
<hr>
{% for tag in project.tags.all %}  <!-- With no ()  or it won't work -->
    <span style="border 1px solid grey"> {{ tag }} </span>
{% endfor %}
</hr>
<br>
<p> {{ project.description }} </p>
```

Try an do a test run, the output should be the same as with querying in the functional part.



Projects template clean up.

A final task, let's clean up our projects.html template and get rid of all the unnecessary elements that we won't be using anymore hereforth.

The projects.html should now look like:

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

<h1> Projects </h1>

<table>
    <tr>
    <th> ID             </th>
    <th> Project        </th>
    <th> Positive Votes </th>
    <th> Ratio          </th>
    <th> Created on     </th>
    <th></th>
    </tr>
    {% for project in projects %}
        <tr>
            <td> {{ project.id }} </td>
            <td> {{ project.title }} </td>
            <td> {{ project.vote_total }} </td>
            <td> {{ project.vote_ratio }}% </td>
            <td> {{ project.created }} </td>
            <td> <a href="{% url 'project' project.id %}">View</a> </td>
        </tr>
    {% endfor %}
 </table>

{% endblock content %}
```