Parameters
In this reading, you'll explore the different options for using parameters in a web application and showcase how they are related to the GET, PUT, POST and DELETE operations.
You'll familiarize yourself with the differences between path, query, and body params and how they're associated with HTTP methods such as GET, PUT, POST and DELETE.
The view function in Django is like any other Python function in that it receives its mandatory argument as the request object from the server context. The client may pass additional arguments via different methods.

Path parameter
The client browser sends data along with the URL itself. For example, a URL such as http://example.com/customer/5. Here, the URL endpoint id, **`/customer/5`**, is the variable parameter (this can be any other number).
The parameter linked to the URL's endpoint is called a **`path`** parameter. Note that there may be multiple path parameters in the URL, separated by / symbol.
For now, consider that the browser will use the URL http://localhost:8000/getuser/John/1

The URL dispatcher should identify John as the **`name`** parameter and 1 as the **`id`** parameter.

This pattern is mapped to the **`pathview()`** function with the following path in the URL patterns list in the app's **`url.py`** file.

```
path('getuser/<name>/<id>', views.pathview, name='pathview'),
```

Next, add the **`pathview()`** function in **`views.py`** file.

```
from django.http import HttpResponse
def pathview(request, name, id):
    return HttpResponse("Name:{} UserID:{}".format(name, id))
```

As a result, the parameters in the above URL are parsed as name and id parameters and picked by the **`pathview()`** function, returning the displayed response.
An important thing to understand here is the parameters added inside the **`path()`** function in the **urls.py** file must match the arguments added inside the **`pathview()`** view function associated with it in the **views.py** file.
Path converters

The URL pattern treats the identifiers in angular brackets (**`<..>`**) as the path parameters. By default, it parses the received value to the string type. Other path converters available are:
`str` - Matches any non-empty string and excludes the path separator, '/'. This is the default if a converter isn't included in the expression.
`int` - Matches zero or any positive integer and returns an **`int`**. For example:**`/customer/<int:id>`**

`slug` - Matches any slug string consisting of ASCII letters or numbers, including the hyphen and underscore characters.
`uuid` - Matches a formatted UUID. For example: **075194d3-6885-417e-a8a8-6c931e272f00** and returns a UUID instance.
`path` - Matches any non-empty string and includes the path separator, '**`/`**'.

Query parameter
The client URL may contain a query string linked to the endpoint, for example,
http://localhost:8000/getuser/?name=John&id=1
A query string is a sequence of one or more **`key=value`** pairs concatenated by the **`&`** symbol. Each key is the query parameter. The query string ends with the **`?`** symbol after the URL endpoint.
Query strings are an alternative approach to URL parameters for adding URL configurations.

The URL dispatcher doesn't parse these parameters. They are fetched by the view from the request object it receives.
The request object's **`GET`** property is a dictionary object.
The key-value pairs in the query string are added to the **`request.GET`** property. Hence, the name can be obtained with **`request.GET['name']`** expression.

The next step is to add the following path in the **`urls.py`** file:

```
path('getuser/', views.qryview, name='qryview')
```

Declare the **`qryview`** function in the **`views.py`** file.

```
def qryview(request):
    name = request.GET['name']
    id = request.GET['id']
    return HttpResponse("Name:{} UserID:{}".format(name, id))
```

Now, start the server and use http://localhost:8000/myapp/?name=John&id=1 as the URL. The client gets back the response shown above.
Body parameters
An HTML form sends the data to the URL mentioned in its action attribute using the POST method. The POST method is a more secure way of sending data than the GET method because the data is not revealed in the URL.
Let's construct a simple form containing two text input elements. Then, save it as **`form.html`** in the **`templates`** folder.

```
<form action="/myapp/getform/" method="POST">
    {% csrf_token %}
    <p>Name: <input type="text" name="id"></p>
    <p>UserID :<input type="name" name="name"></p>
    <input type="submit">
</form>
```

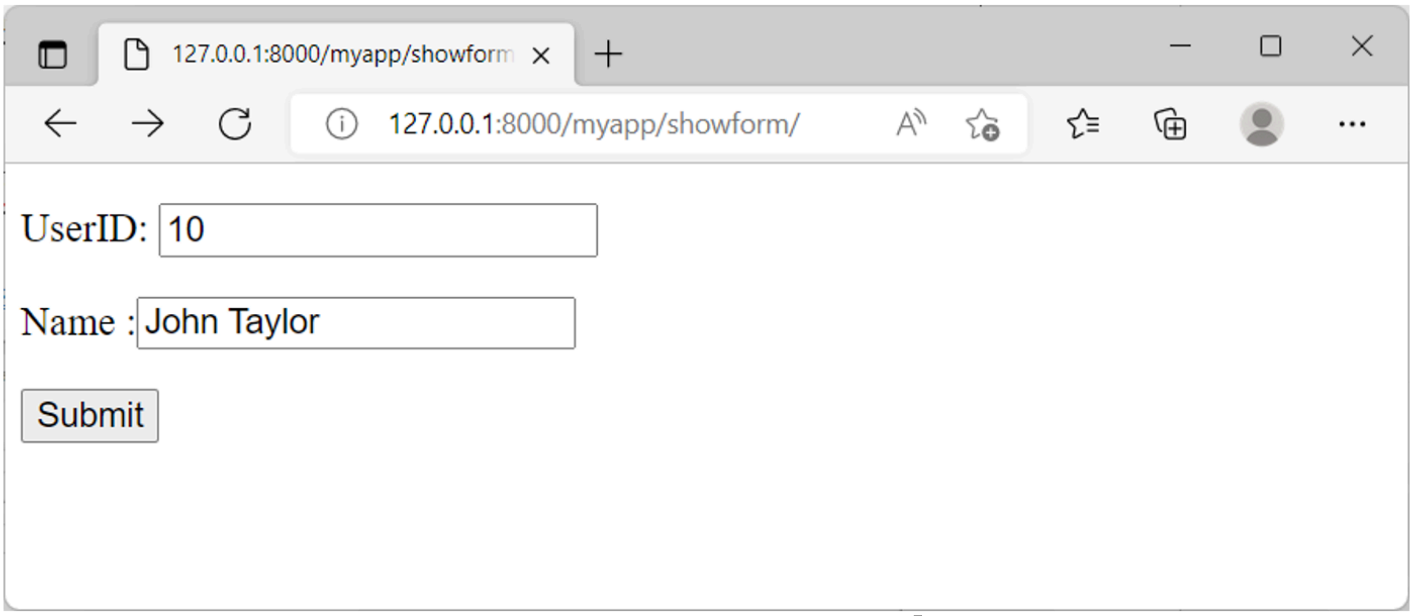
The **`{% csrf_token %}`** tag is necessary to prevent cross-site forgery attacks. You'll learn more about this later on in the course.
Next, you'll need to provide a view that will render this form:

```
def showform(request):
    return render(request, "form.html")
```

The URL patterns list must be updated using the following path:

```
path("showform/", views.showform, name="showform"),
```

The http://localhost:8000/myapp/showform/ URL displays this form to the user:



UserID: 10

Name : John Taylor

When this is submitted, it goes to the http://localhost:8000/myapp/getform/ URL. Now, map it to **getform()** function in the **urls.py** file.

```
path("getform/", views.getform, name='getform'),
```

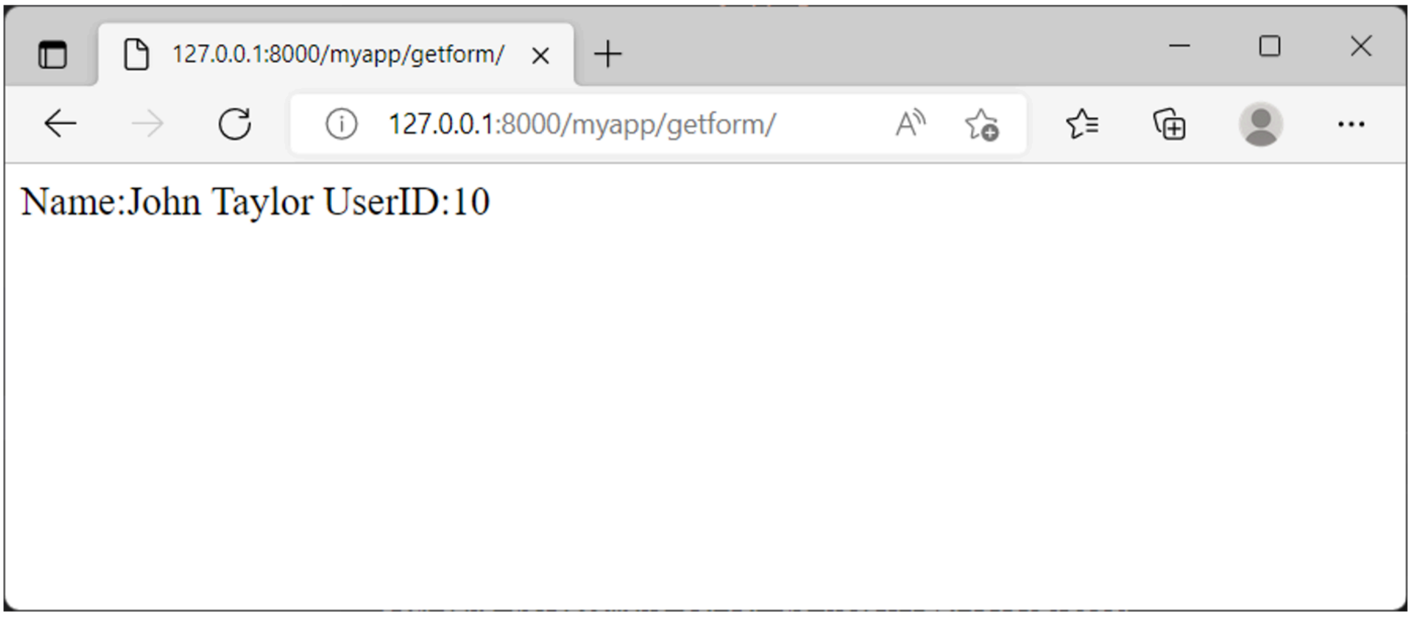
The form data that the user posts becomes part of the request body.

Therefore, the name attribute of each form element becomes a body parameter and the data entered becomes its value.

The view function passes these body parameters from the **request.POST** dictionary-like attribute.

```
def getform(request):
    if request.method == "POST":
        id=request.POST['id']
        name=request.POST['name']
    return HttpResponse("Name:{} UserID:{}".format(name, id))
```

Complete the form and submit it. The **getform()** function returns the data as its response.



```
Name:John Taylor UserID:10
```

In the real world, the received data may be meaningfully processed, such as storing it in a database table.

You'll learn how to do this when you explore Models, Forms and Database Connections.

In this reading, you learned about the path, query and body parameters and how to pass and process them using different HTTP methods.