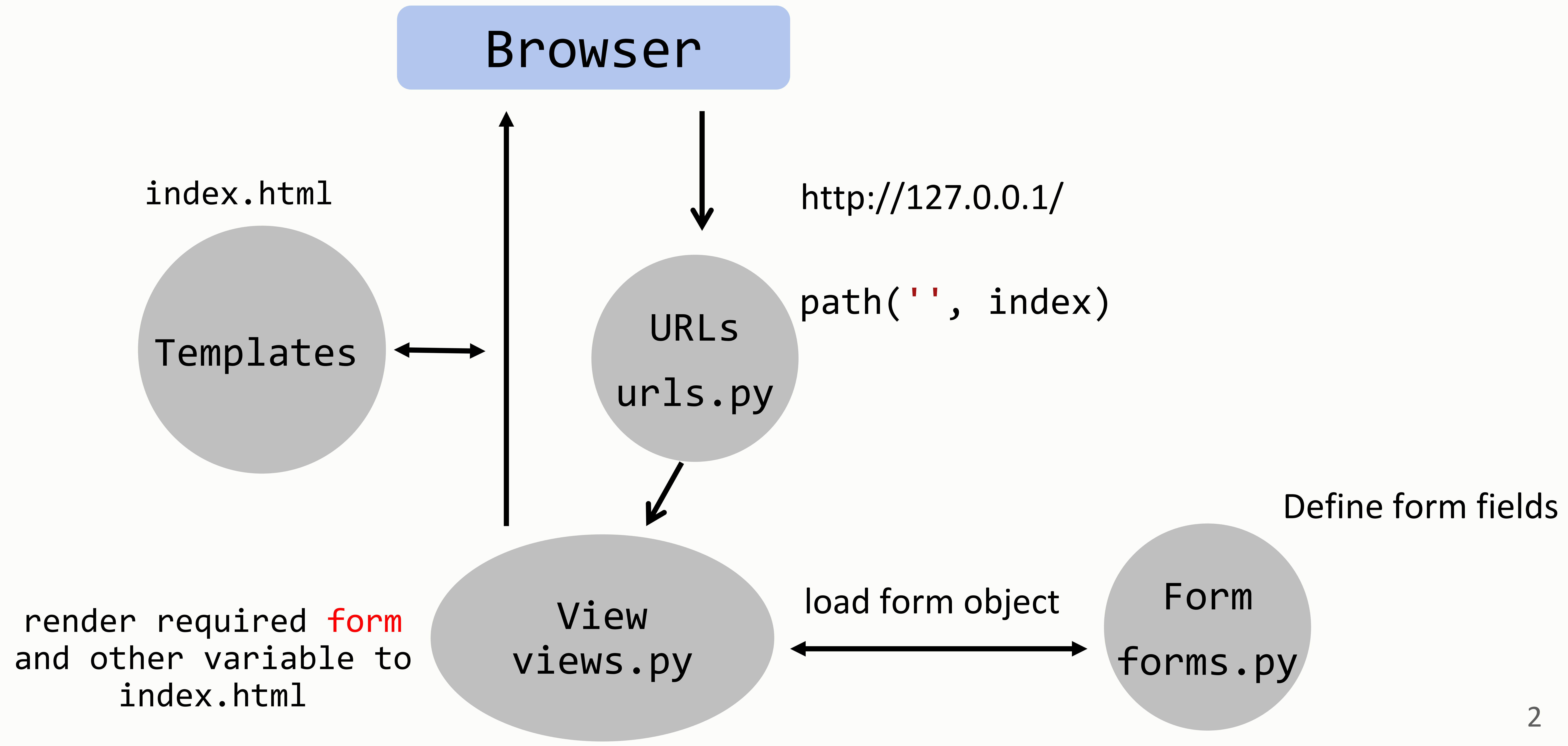
Django (Form / Cookie / Session)

Form (Get)


Browser
http://127.0.0.1/
path('', index)
URLs
urls.py
index.html
Templates
Define form fields
View
views.py
load form object
Form
forms.py
render required form
and other variable to
index.html

Form (Get)

Django provides form function (form class)

- Define the corresponding Form class in forms.py (established by ourself)
- Import this form to views.py, and render it to the template (index.html)
- Modify index.html to load the form: {{ form }}
- Modify urls.py

Form (Get) – forms.py

Inherit forms.Form

Create forms.py by ourself

```
class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Form (Get) – views.py

Modify views.py

```
from week10_website.forms import ContactForm
from django.shortcuts import render

def index(request):
        form = ContactForm()
        return render(request, "index.html", {"form": form})
```

Form (Get) – index.html

Modify index.html

```
<!DOCTYPE html>  <html>
<head>    <title> My Website </title>   </head>
<body>
    <form action="/form_post/" method="post">
        {% csrf_token %}
        {% if form.non_field_errors %}
            <div> {{ form.non_field_errors }}  </div>
        {% endif %}
        {{ form }}
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

Form (Get) – urls.py

Modify urls.py

```
from week10_website.views import index
```

```
path('', index),
```

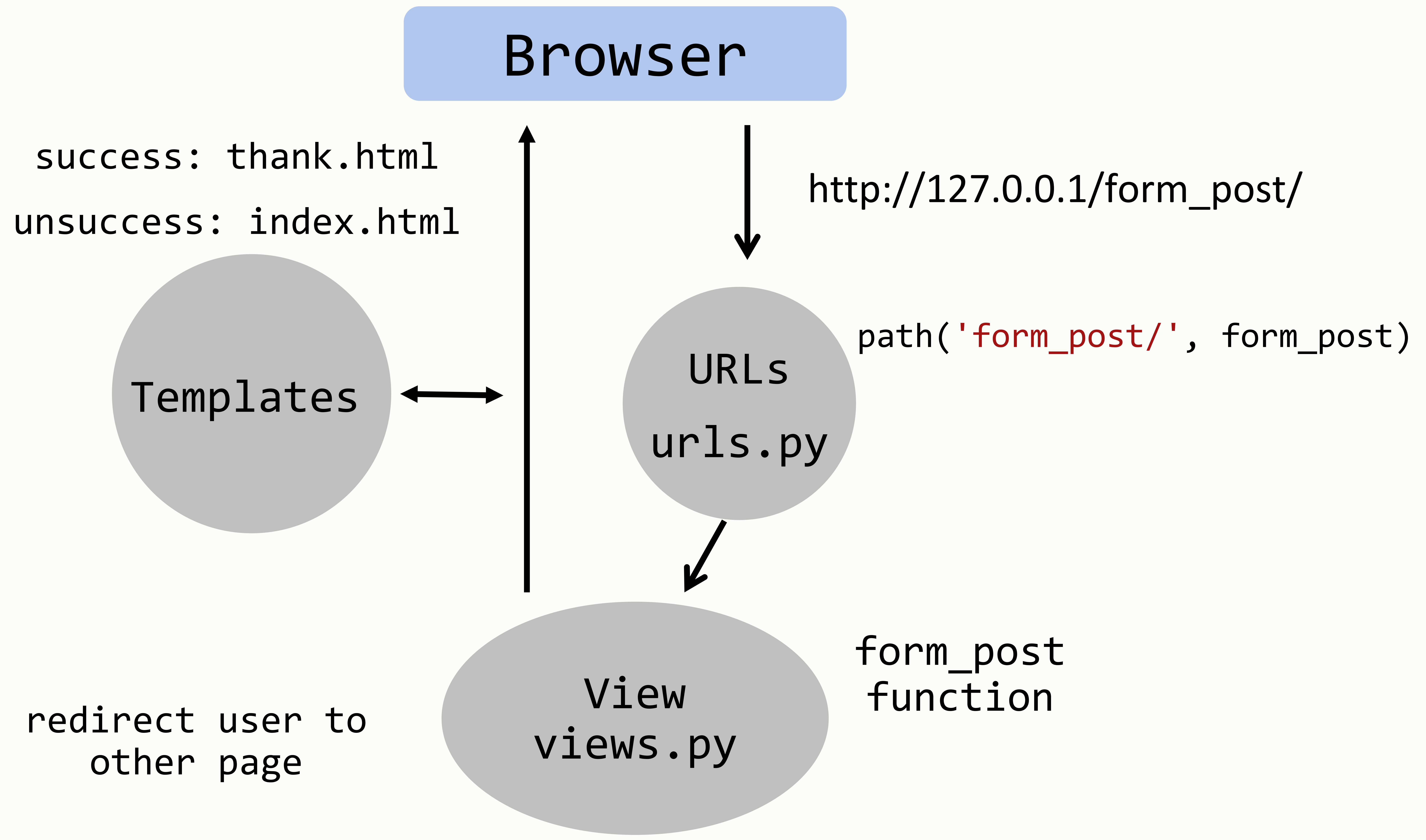
Form - Advantage

- **Built-in Security**
  Automatically removes dangerous input and helps prevent attacks like XSS.

- **Less Repetition**
  Define the form once and reuse it across multiple templates.

- **Cleaner Code**
  Keep your HTML simple by letting Django handle form rendering and validation.

Form (Post)


Browser
success: thank.html
unsuccess: index.html
http://127.0.0.1/form_post/
path('form_post/', form_post)
Templates
URLs
urls.py
View
views.py
form_post
function
redirect user to
other page

Modify views.py

Form (Post) – views.py

```
from mywebsite.forms import ContactForm
from django.http import HttpResponseRedirect
def form_post(request):
    if request.method == "POST":
        form = ContactForm(request.POST)

        # check whether it's valid:
        if form.is_valid(): # process the data in form.cleaned_data as required
            subject = form.cleaned_data["subject"]
            message = form.cleaned_data["message"]
            sender = form.cleaned_data["sender"]
            cc_myself = form.cleaned_data["cc_myself"]
            # redirect to a new URL:
            return HttpResponseRedirect("/thank/")
        else:
            return render(request, 'index.html', {'form': form})
```

redirect to http://127.0.0.1/thank/

render index.html to the current page

Form (Post) – views.py

Add thank.html / Modify views.py for thank function

Modify urls.py

```
from django.contrib import admin
from django.urls import path
from mywebsite.views import index, form_post, thank

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', index),
    path('form_post/', form_post),
    path('thank/', thank),
]
```

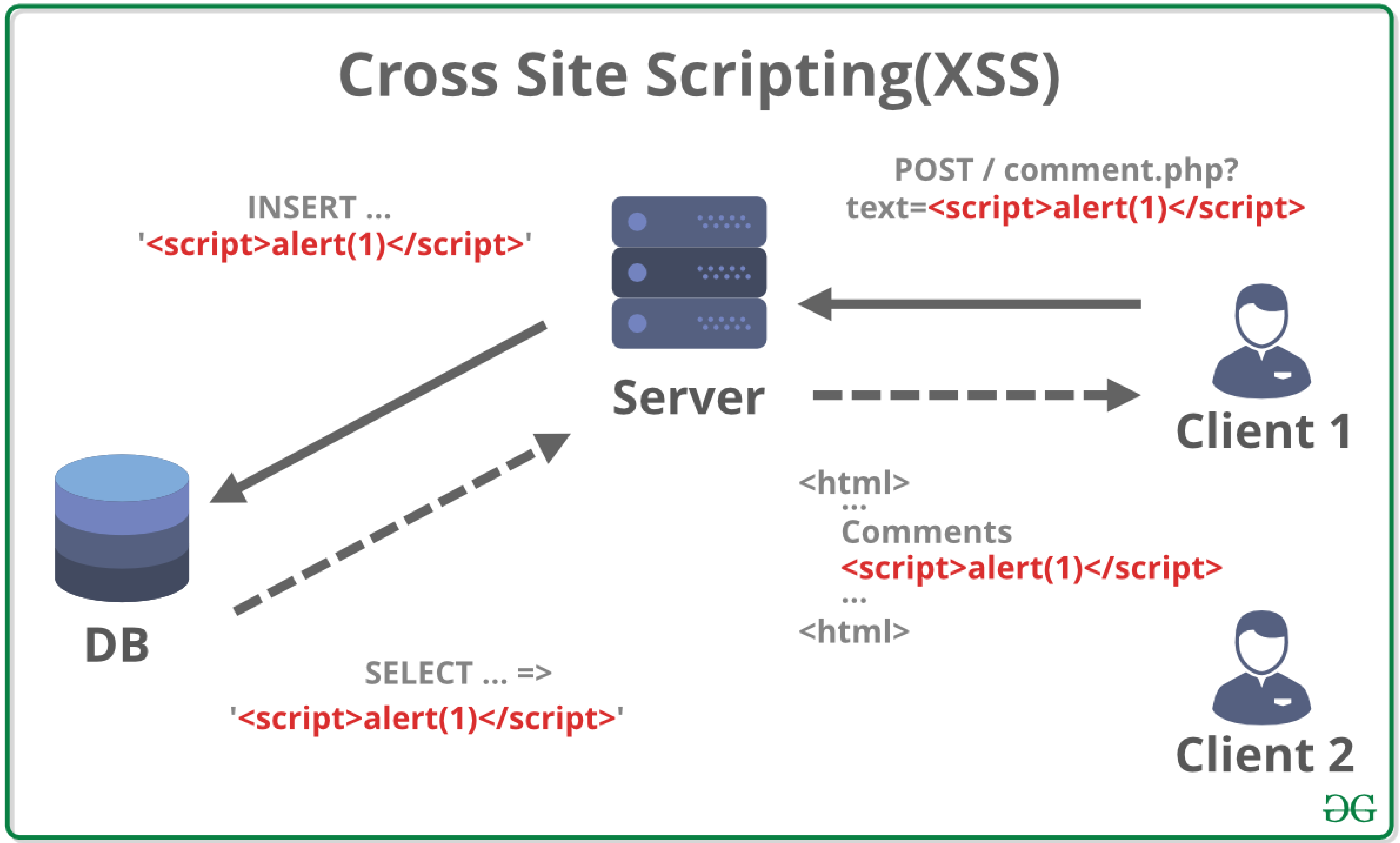
What's cleaned data ?

- **Input is properly validated and filtered**
  We check the data before using it. For example, making sure an email is really an email.

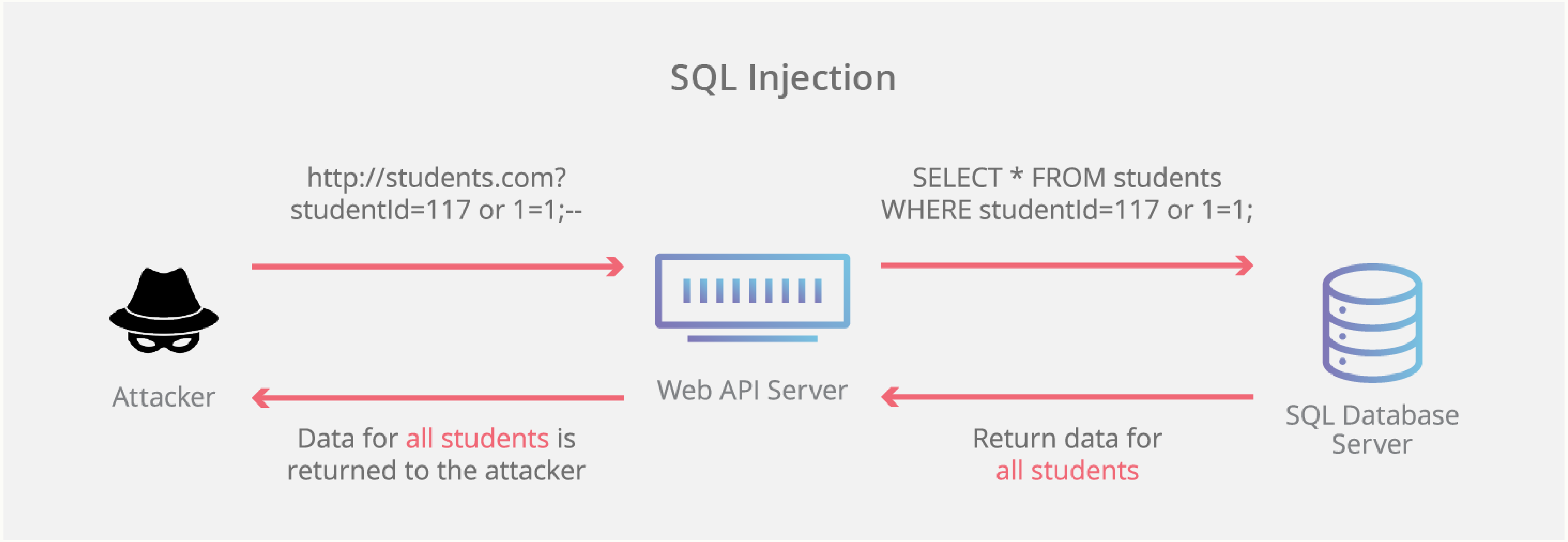
- **Harmful content is removed or escaped**
  Dangerous code like <script> tags won’t run — they get turned into safe text.

- **The data is safe to display or store**
  Clean input won’t break your webpage or database.

What's cleaned data ?
XSS (Cross-Site Scripting)


Cross Site Scripting(XSS)
INSERT ...
'<script>alert(1)</script>'
Server
POST / comment.php?
text=<script>alert(1)</script>
Client 1
DB
<html>
...
Comments
<script>alert(1)</script>
...
<html>
SELECT ... =>
'<script>alert(1)</script>'
Client 2

What's cleaned data ?
SQL Injection

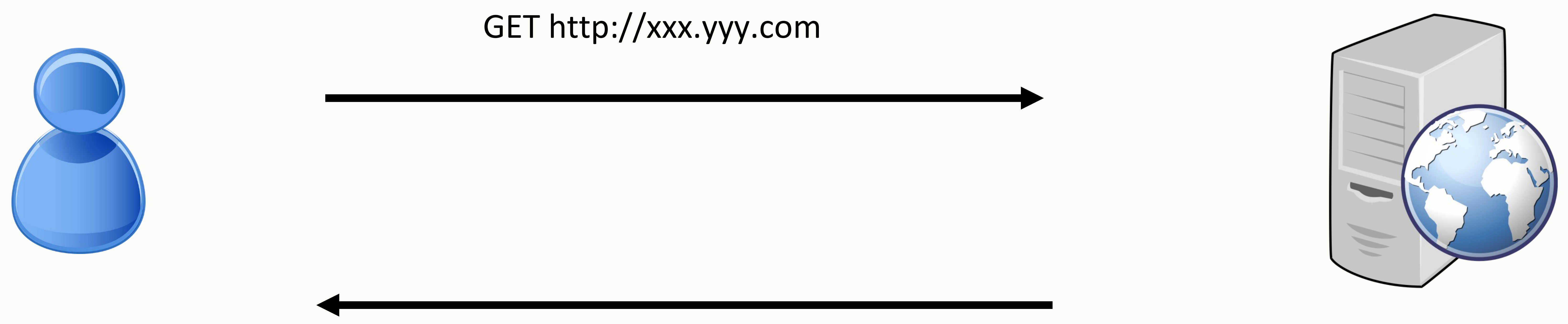


GET: using request.GET.get() & ORM
POST: using request.POST.get() & cleaned_data() & ORM

Cookie

Key and value pairs (key=value) that the server embeds on the user's browser

Cookie are store in a file in the user's browser



HTTP/1.0 200 OK

Set-Cookie: LSID=DQAAAK…Eaem_vYg; Path=/accounts; Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly

Set-Cookie: HSID=AYQEVn…DKrdst; Domain=.foo.com; Path=/; Expires=Wed, 13 Jan 2021 22:23:01 GMT; HttpOnly

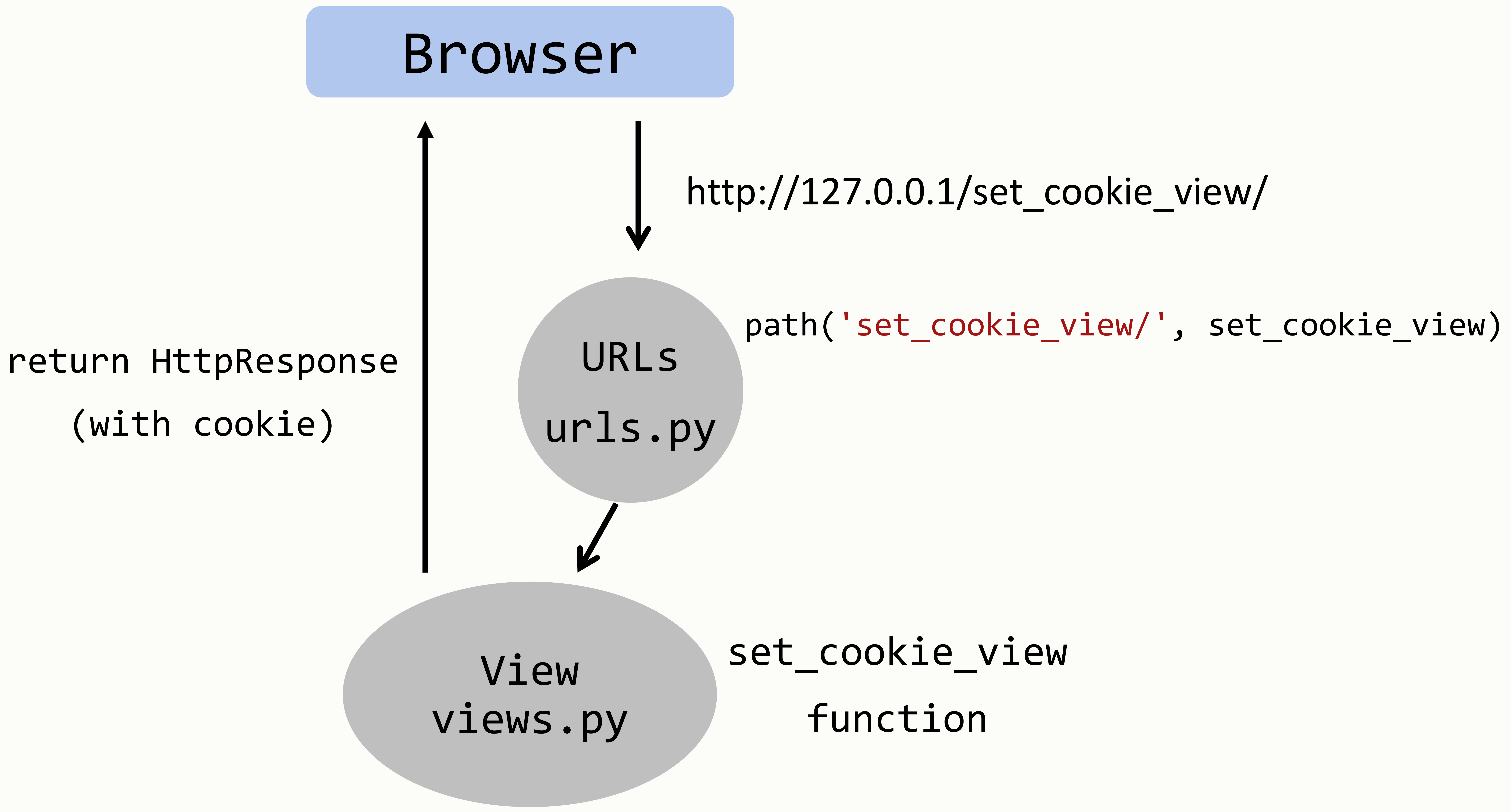
Set-Cookie: SSID=Ap4P…GTEq; Domain=foo.com; Path=/; Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly

/accounts

/accounts/xxx

/accounts/xxx/yyy

Cookie in Django – Set Cookie



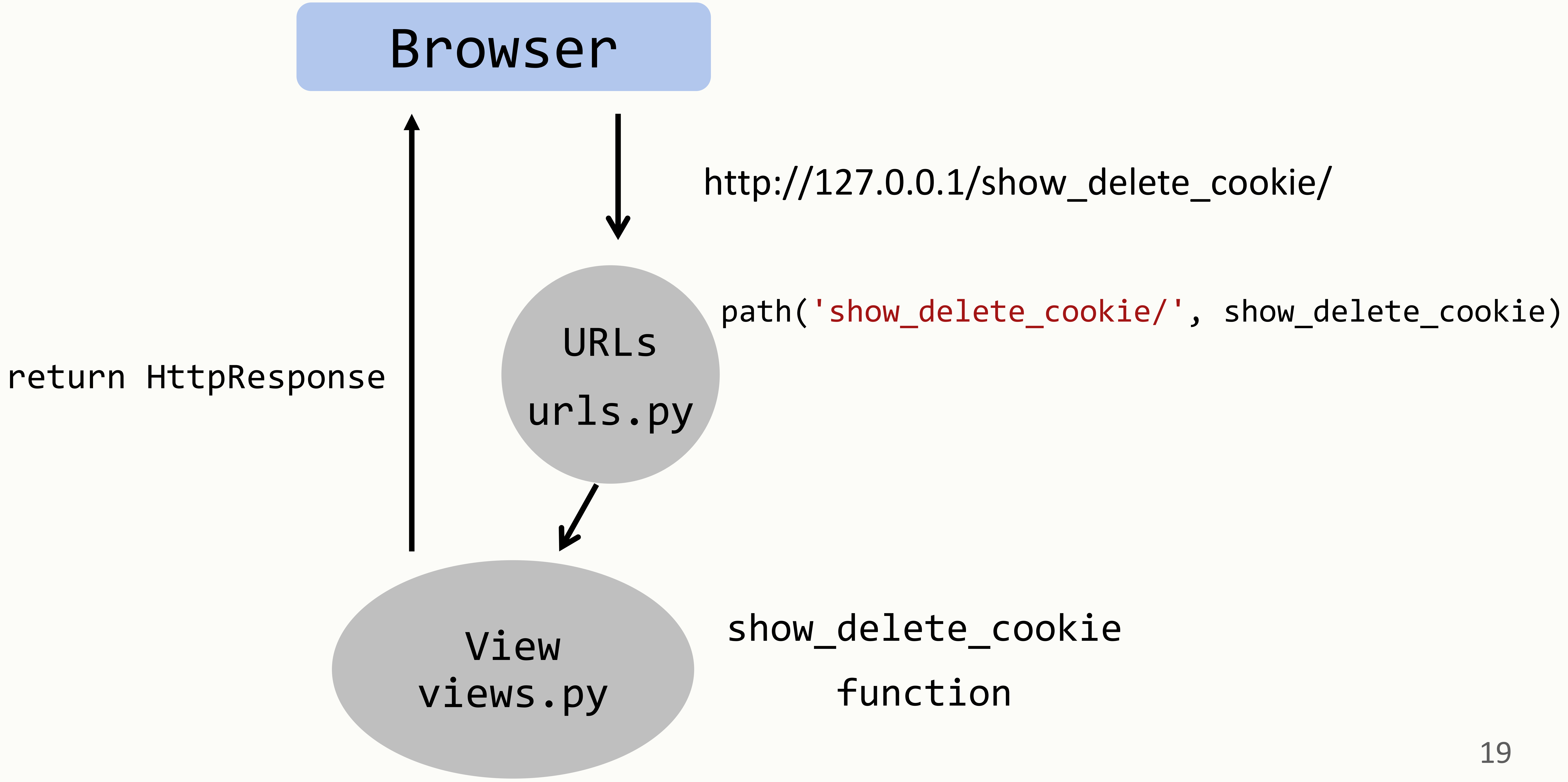
Cookie in Django – Set Cookie – views.py

```
from django.http import HttpResponse

def set_cookie_view(request):
    response = HttpResponse("Cookie has been set!")
    response.set_cookie('my_cookie', 'cookie_value', max_age=3600)
    # Expires in 1 hour
    return response

# response = render(...)
# response.set_cookie(...)
# return response
```

Cookie in Django – Set Cookie – urls.py

```
from mywebsite.views import set_cookie_view

path('set_cookie_view/', set_cookie_view)
```

Cookie in Django – Show and Delete Cookie


Browser
http://127.0.0.1/show_delete_cookie/
URLs
urls.py
path('show_delete_cookie/', show_delete_cookie)
return HttpResponse
View
views.py
show_delete_cookie
function

Cookie in Django – Show and Delete Cookie – views.py

```
from django.http import HttpResponse

def show_delete_cookie(request):
    value = request.COOKIES.get('my_cookie')
    # get cookie
    print(value)

    response = HttpResponse("Cookie has been deleted!")
    response.delete_cookie('my_cookie')
    # delete cookie

    return response
```

Cookie in Django – Show and Delete Cookie – urls.py

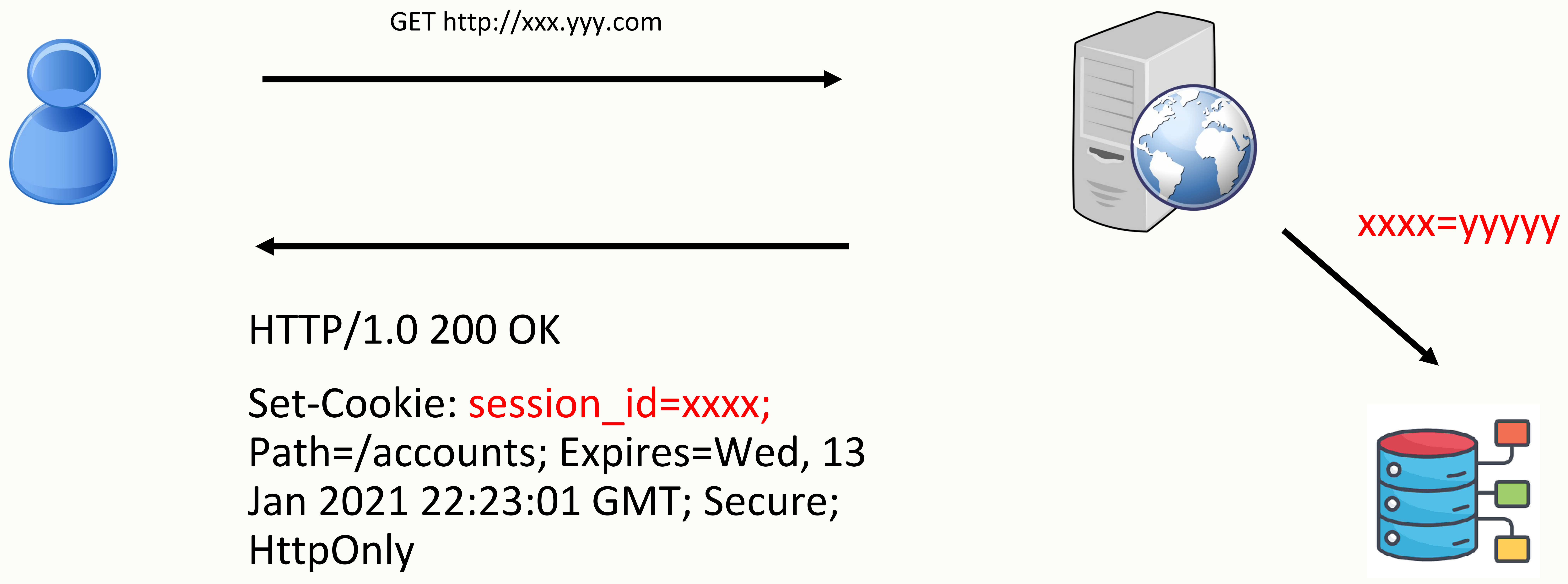
```
from mywebsite.views import show_delete_cookie

path('show_delete_cookie/', show_delete_cookie)
```

Cookie - Problem

1. May be closed/modify by the user (cookie is stored in the browser)

2. HTTP protocol is unsafe, cookie may be interception, tampering, forgery

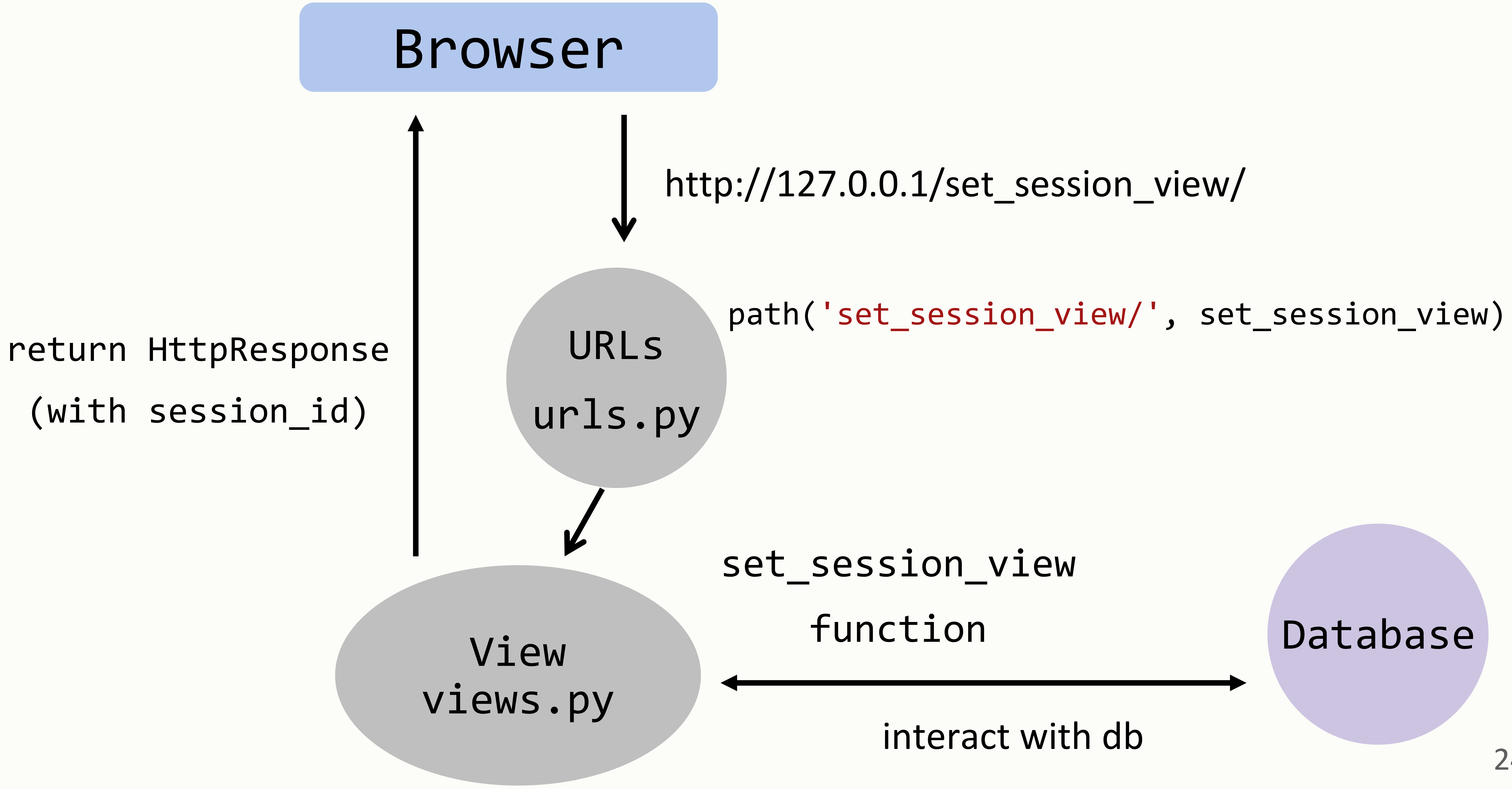
Session



HTTP/1.0 200 OK

Set-Cookie: session_id=xxxx; Path=/accounts; Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly

Session in Django – Set Session



Session in Django – Set Session – Setting

Edit the `MIDDLEWARE` in settings.py

`'django.contrib.sessions.middleware.SessionMiddleware’.`

Edit the `INSTALLED_APPS` in settings.py

`'django.contrib.sessions'`

# if you open a new project by yourself:

python .\manage.py migrate

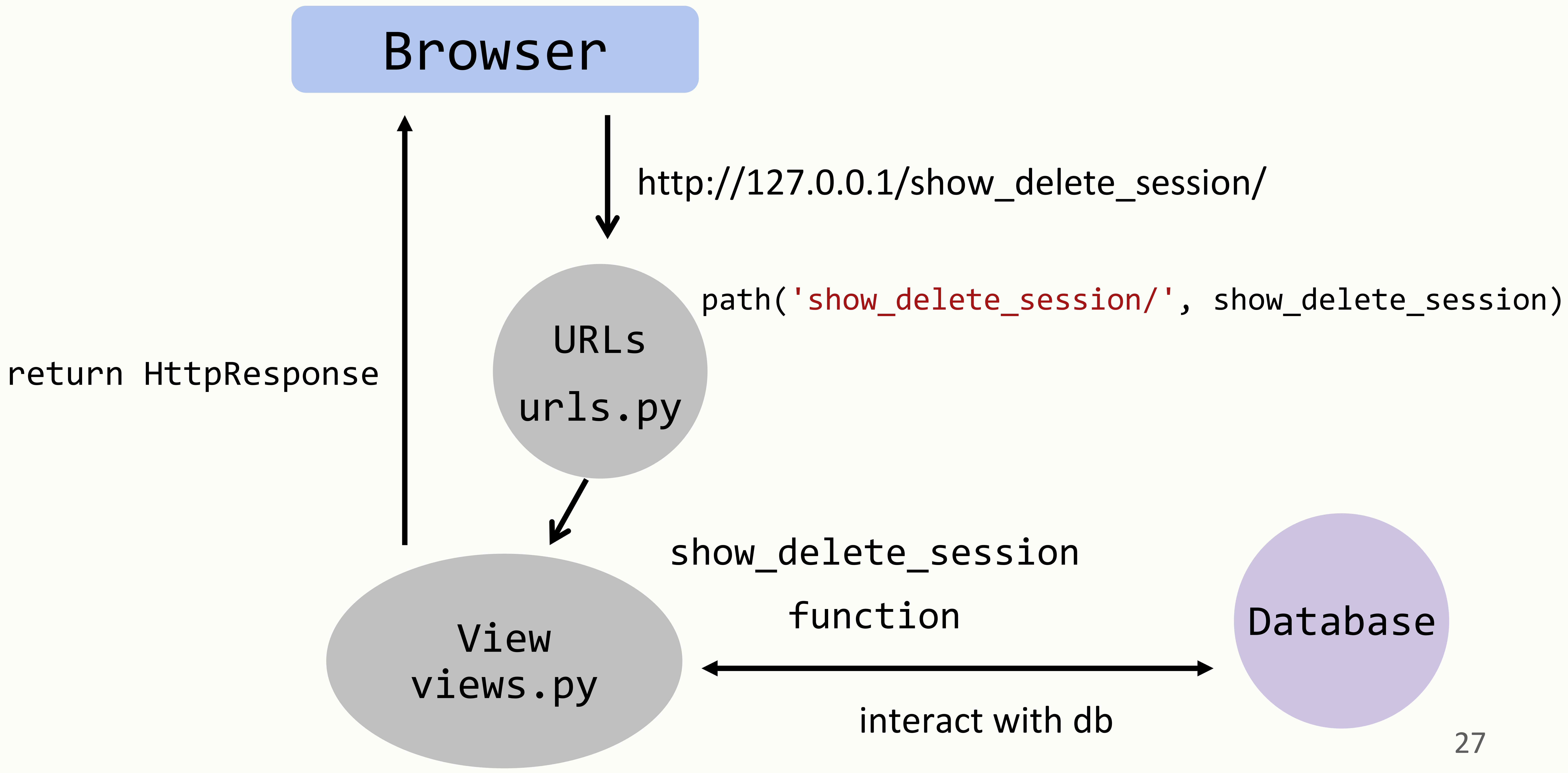
To apply migrations: sessions

Session in Django – Set Session views.py & urls.py

```
def set_session_view(request):
    request.session['my_session'] = 'This is a session'
    return HttpResponse("Session has been set.")
```

```
path('set_session_view/', set_session_view),
```

Session in Django – Show and Delete Session



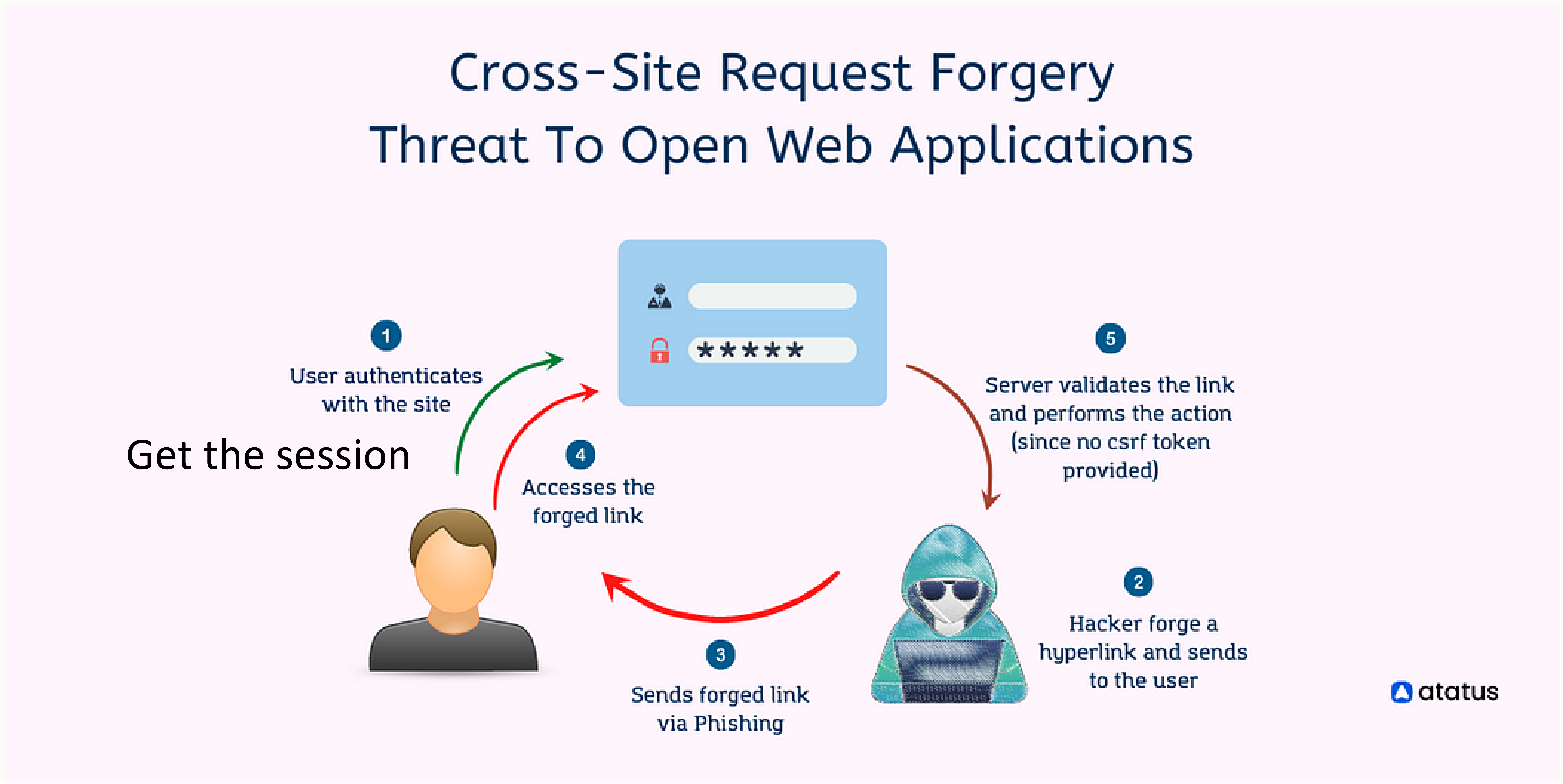
Session in Django – Show and Delete Session views.py

```
def show_delete_session(request):
    session_content = request.session.get('my_session', 'Some default content')

    try:
        # delete a column in the session
        del request.session['my_session']
    except KeyError:
        pass

    # remove the whole session
    request.session.flush()
    return HttpResponse("The session data stored in the db is: " + session_content)
```

Session in Django – Show and Delete Session urls.py

```
path('show_delete_session/', show_delete_session)
```

Cross-Site Request Forgery (CSRF)


Cross-Site Request Forgery
Threat To Open Web Applications
*****
1
User authenticates with the site
4
Accesses the forged link
3
Sends forged link via Phishing
2
Hacker forge a hyperlink and sends to the user
5
Server validates the link and performs the action (since no csrf token provided)
atatus

Get the session