**Django Session**

**What are sessions?**

All communication between web browsers and servers is via the HTTP protocol, which is *stateless*.

The fact that the protocol is stateless means that messages between the client and server are completely independent of each other— there is no notion of "sequence" or behaviour based on previous messages.

As a result, if you want to have a site that keeps track of the ongoing relationships with a client, you need to implement that yourself.

Sessions are the mechanism used by Django (and most of the Internet) for keeping track of the "state" between the site and a particular browser.

Sessions allow you to store arbitrary data per browser, and have this data available to the site whenever the browser connects.

Individual data items associated with the session are then referenced by a "key", which is used both to store and retrieve the data.

Django uses a cookie containing a special *session id* to identify each browser and its associated session with the site. The actual session *data* is stored in the site database by default (this is more secure than storing the data in a cookie, where they are more vulnerable to malicious users). You can configure Django to store the session data in other places (cache, files, "secure" cookies), but the default location is a good and relatively secure option.

A session is a mechanism to store information on the server side during the interaction with the web application.

In Django, by default session stores in the database and also allows file-based and cache based sessions. It is implemented via a piece of middleware and can be enabled by using the following code.

Put **django.contrib.sessions.middleware.SessionMiddleware** in MIDDLEWARE and **django.contrib.sessions** in INSTALLED\_APPS of **settings.py** file.

To set and get the session in views, we can use **request.session** and can set multiple times too.

The **class backends.base.SessionBase** is a base class of all session objects. It contains the following standard methods.

|  |  |
| --- | --- |
| **Method** | **Description** |
| \_\_getitem\_\_(key) | It is used to get session value.  Example: **fav\_color = request.session['fav\_color']** |
| \_\_setitem\_\_(key, value) | It is used to set session value.  Example: **request.session['fav\_color'] = 'blue'** |
| \_\_delitem\_\_(key) | It is used to delete session object.  Example: **del request.session['fav\_color']**. This raises **KeyError** if the given **key** isn’t already in the session. |
| \_\_contains\_\_(key) | It checks whether the container contains the particular session object or not.  Example: **'fav\_color' in request.session** |
| get(key, default=None) | It is used to get session value of the specified key.  Example: **fav\_color =request.session.get('fav\_color', 'red')** |

**When sessions are saved?**

By default, Django only saves to the **session database** when the session has been modified – that is if any of its dictionary values have been assigned or deleted.

**set\_expiry(seconds\_*value*)**

Sets the expiration time for the session

If **value** is an integer, the session will expire after that many seconds of inactivity.

**For example:** calling **request.session.set\_expiry(300)** would make the session expire in 5 minutes.

If **value** is **0**, the user’s session cookie will expire when the user’s Web browser is closed.

If **value** is **None**, the session reverts to using the global session expiry policy.

**clear\_expired()**

Removes expired sessions from the session store.

**#views.py**

**from** django.shortcuts **import** render  
**from** django.http **import** HttpResponse  
  
**def** setsession(request):  
 request.session[**'sname'**] = **'naveen'** request.session[**'semail'**] = **'naveen@gmail.com'  
 return** HttpResponse(**"session is set"**)  
  
**def** getsession(request):  
 studentname = request.session[**'sname'**]  
 studentemail = request.session[**'semail'**]  
 **return** HttpResponse(studentname + **" "** + studentemail);

**Url mapping to call both the functions.**

**# urls.py**

**from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** myapp **import** views  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**'index/'**, views.index),  
 path(**'ssession'**, views.setsession),  
 path(**'gsession'**, views.getsession)  
]

**Run the Server**

**Examples**

This simplistic view sets a has\_commented variable to True after a user posts a comment. It doesn’t let a user post a comment more than once:

def post\_comment(request, new\_comment):

if request.session.get('has\_commented', False):

return HttpResponse("You've already commented.")

c = comments.Comment(comment=new\_comment)

c.save()

request.session['has\_commented'] = True

return HttpResponse('Thanks for your comment!')

This simplistic view logs in a “member” of the site:

def login(request):

m = Member.objects.get(username=request.POST['username'])

if m.password == request.POST['password']:

request.session['member\_id'] = m.id

return HttpResponse("You're logged in.")

else:

return HttpResponse("Your username and password didn't match.")

…And this one logs a member out, according to login() above:

def logout(request):

try:

del request.session['member\_id']

except KeyError:

pass

return HttpResponse("You're logged out.")

**When sessions are saved?**

By default, Django only saves to the session database when the session has been modified – that is if any of its dictionary values have been assigned or deleted

**Example**

*# Session is modified.*

request.session['foo'] = 'bar'

*# Session is modified.*

**del** request.session['foo']

*# Session is modified.*

request.session['foo'] = {}

**Django Cookie**

A cookie is a small piece of information which is stored in the client browser. It is used to store user's data in a file permanently (or for the specified time).

Cookie has its expiry date and time and removes automatically when gets expire. Django provides built-in methods to set and get cookie.

The **set\_cookie()** method is used to set a cookie and **get()** method is used to get the cookie.

The **request.COOKIES['key']** array can also be used to get cookie values.

max\_age to set expiry

In **views.py,** two functions setcookie() and getcookie() are used to set and get cookie respectively

**# views.py**

**from** django.shortcuts **import** render  
**from** django.http **import** HttpResponse  
  
**def** setcookie(request):  
 response = HttpResponse(**"Cookie Set"**)  
 response.set\_cookie(**'sathya'**, **'sathyatech.com'**)  
 **return** response  
  
**def** getcookie(request):  
 name = request.COOKIES[**'sathya'**]  
 **return** HttpResponse(**"Naveen@: "** + name);

**And URLs specified to access these functions.**

**# urls.py**

**from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** myapp **import** views  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**'index/'**, views.index),  
 path(**'scookie'**,views.setcookie),  
 path(**'gcookie'**,views.getcookie)  
]

**Run Server**

**Creating CSV with Django**

Django uses Python's built-in CSV library to create Dynamic CSV (Comma Separated Values) file.

We can use this library in our project's view file.

**# views.py**

**from** django.http **import** HttpResponse  
**from** django.shortcuts **import** render  
**import** csv

**def** getfile(request):  
 response = HttpResponse(content\_type=**'text/csv'**)  
 response[**'Content-Disposition'**] = **'attachment; filename="employee.csv"'** writer = csv.writer(response)  
 writer.writerow([**'101'**, **'Ravi'**, 125000.00])  
 writer.writerow([**'102'**, **'Krishna'**, 185000.00])  
 **return** response

**# urls.py**

**Provide url for the function.**

path('csv',views.getfile)

**Dynamic CSV using Database**

**# views.py**

**from** .models **import** Employee  
  
**def** getfiledb(request):  
 response = HttpResponse(content\_type=**'text/csv'**)  
 response[**'Content-Disposition'**] = **'attachment; filename="file.csv"'** employees = Employee.objects.all()  
 writer = csv.writer(response)  
 **for** employee **in** employees:  
 writer.writerow([employee.idno,employee.name,employee.salary])  
 **return** response

**# models.py**

**from** django.db **import** models  
  
**class** Employee(models.Model):  
 idno = models.IntegerField(default=4)  
 name = models.CharField(max\_length=50)  
 salary = models.DecimalField(max\_digits=10,decimal\_places=2)

**Run Server**

**'Content-Disposition'**

**==================**

**The first parameter in the HTTP context is either inline (default value, indicating it can be displayed inside the Web page, or as the Web page) or attachment (indicating it should be downloaded; most browsers presenting a 'Save as' dialog, prefilled with the value of the filename parameters if present).**

**Content-Disposition: inline**

**Content-Disposition: attachment**

**Content-Disposition: attachment; filename="filename.jpg"**

**Django PDF**

To generate PDF, we will use ReportLab Python PDF library that creates customized dynamic PDF.

It is an open source library and can be downloaded easily by using the following command.

1. pip install reportlab

After installing, we can import it by import keyword in the view file.

**Example.**

**from** reportlab.pdfgen **import** canvas  
**from** django.http **import** HttpResponse  
  
**def** getpdf(request):  
 response = HttpResponse(content\_type=**'application/pdf'**)  
 response[**'Content-Disposition'**] = **'attachment; filename="sample.pdf"'** p = canvas.Canvas(response)  
 p.setFont(**"Times-Roman"**, 55)  
 p.drawString(100, 700, **"Hello, DJANGO."**)  
 p.showPage()  
 p.save()  
 **return** response

First, provide MIME (content) type as application/pdf, so that output generates as PDF rather than HTML,

Set Content-Disposition in which provide header as attachment and output file name.

Pass response argument to the canvas and drawstring to write the string after that apply to the save() method and return response.

**# urls.py**

path('pdf',views.getpdf)

**Run server**