

## Full client session

**Author :** Frederic De Zorzi

**Contact :** [fredz@\\_nospam\\_pimentech.fr](mailto:fredz@_nospam_pimentech.fr)

**Revision :** 8259

**Date :** 2009-05-29

**Copyright :** This document has been placed in the public domain.

**Tags :** django

### Status

In production.

### Important update

A severe bug has been found, caused by spaces and comma characters in values (not handled by Safari). Please update the middleware and the JavaScript.

### Abstract

With this middleware, Django session values are stored in client cookies, accessible (r/w) directly in JavaScript (except private variables). **Extremely useful if one wants to use specific session variables on cached pages with JavaScript.**

### Introduction

Putting pages in cache for medium to high-traffic sites is essential, but one might have to display some internet-user specific informations on these cached pages.

For example, we rewrote in *Django* the old php-based [\\*Century21 France\\* internet site](#). This site delivers 1.000.000 dynamic pages per day, the database contains all the Information System with 100.000 active properties modified in real-time.

We found a way to cache/uncache dynamically property detail, but we had to show the user's selection of properties on this page :

With this middleware, each session variable is stored in a client cookie in *json*, crypted for private session variables, like “\_auth\_user\_id”.

So for example with the following Python code :

```
request.session["user"] = { "surname" : "Fred", "name" : "De Zorzi" }
```

The user cookie 'dj\_session.user' will contain the jsonified value “{ ”surname“ : ”Fred“, ”name“ : ”De Zorzi“ }”. One has the possibility to get/set this value in JavaScript, with the modified [jQuery cookie plugin](#) (provided below) :

```
$("#username").html ($.cookie("dj_session.user")[name]);
```

### Discussion

Session mechanism is used to store users specific context. Traditionally, sessions variables are stored in server, coupled with client cookie session keys.

These user contexts are volatile by definition and represent generally a small amount of data. So why using server session instead of saving all the user context in cookies ?

- Historically, not all internet users accepted cookie (probably because, on Windows OS, one can choose the “super security level” without cookies, while continuing to eat viruses :).
- Dealing with cookies on server side is not easy, because one has repeatedly to deal with “get” and “response” requests to get / set cookies.
- For security reasons, one does not always want to have session values accessibles and modifiable by the internet user.

With this Django middleware, one can switch to this session storage *without modifying the code*.

Pros :

- Session is accessible for reading and writing in JavaScript.
- Nothing is written on server side, which could increase performances (not benchmarked yet)
- Private variables are transparently crypted/decrypted with AES (C library) : simply prefix cookie variable names with “\_”.

Cons :

- Session variable values are limited to 4k each.

## Installation

- get PimenTech libcommonDjango :

```
svn checkout http://svn.pimentech.org/pimentech/libcommonDjango
```

- install it with “make install”
- You also need to install python-crypto for AES
- in your settings.py, put :

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.pimentech.middleware.full_client_session.PreSessionMiddleware',

    ...other middlewares...

    'django.pimentech.middleware.full_client_session.PostSessionMiddleware'
)
```

That’s all.

## Notes

- Private session variables (beginning with ‘\_’) are crypted with AES, using site settings.SECRET\_KEY. Each session variable is accessible in JavaScript by the internet user, so do not forget to use private prefix with sensible session data.

These are restrictions regarding cookie usage :

- In JavaScript dictionary keys are always strings. So if you have python dictionaries in your session, use strings !
- Data per cookie is limited to 4 Ko.

## Source

### Python MiddleWare

See it in [trac](#)

```
# -*- coding: utf-8 -*-

import time

from django.conf import settings
from django.utils.cache import patch_vary_headers
from django.utils.http import cookie_date
from django.contrib.sessions.backends.base import SessionBase
from django.utils import simplejson
from Crypto.Cipher import AES
import binascii
import urllib
```

```

crypt = AES.new(settings.SECRET_KEY[:32], AES.MODE_ECB)
from django import VERSION

class PreSessionMiddleware(object):
    def process_request(self, request):
        session_key = request.COOKIE.get(settings.SESSION_COOKIE_NAME, None)
        request.session = SessionStore(request, session_key)

class PostSessionMiddleware(object):
    def process_response(self, request, response):
        # If request.session was modified, or if response.session was set, save
        # those changes and set a session cookie.
        try:
            accessed = request.session.accessed
            modified = request.session.modified
        except AttributeError:
            pass
        else:
            if accessed:
                patch_vary_headers(response, ('Cookie',))
            if modified or settings.SESSION_SAVE_EVERY_REQUEST:
                if VERSION[0] == 0:
                    # django 0.xx : session expires at browser close
                    max_age = None
                    expires = None
                else:
                    if request.session.get_expire_at_browser_close():
                        max_age = None
                        expires = None
                    else:
                        max_age = request.session.get_expiry_age()
                        expires_time = time.time() + max_age
                        expires = cookie_date(expires_time)
                # Save the session data and refresh the client cookie.
                request.session.save(request, response, max_age, expires)
                response.set_cookie(settings.SESSION_COOKIE_NAME,
                                    request.session.session_key, max_age=max_age,
                                    expires=expires, domain=settings.SESSION_COOKIE_DOMAIN,
                                    path=settings.SESSION_COOKIE_PATH,
                                    secure=settings.SESSION_COOKIE_SECURE or None)

        return response

class SessionStore(SessionBase):
    """
    A cache-based session store.
    """
    def __init__(self, request, session_key=None):
        self.request = request
        super(SessionStore, self).__init__(session_key)

    def load(self):
        session = {}

```

```

for key, value in self.request.COOKIES.items():
    if key.startswith('dj_session'):
        key = key[11:]
        if key[0] == '_':
            try:
                value = crypt.decrypt(binascii.a2b_base64(value)).strip()
            except:
                value = ""
        else:
            value = urllib.unquote(value)
        try:
            value = simplejson.loads(value)
        except ValueError:
            value = None
        session[key] = value
return session

def create(self):
    return

def set_cookie(self, response, key, value=None, max_age=None, expires=None):
    if value is None:
        response.delete_cookie('dj_session.' + key,
                               path=settings.SESSION_COOKIE_PATH)
        return
    value = simplejson.dumps(value, simplejson.JSONEncoder)
    if key[0] == '_':
        value = value + (16 - len(value) % 16) * ' '
        value = binascii.b2a_base64(crypt.encrypt(value))
    else:
        value = urllib.quote(value)
    if len(value) >= 4096:
        raise ValueError, "Value of session param %s is too long : %s" % (key, value)
    response.set_cookie('dj_session.' + key, value,
                       max_age=max_age,
                       expires=expires, domain=settings.SESSION_COOKIE_DOMAIN,
                       path=settings.SESSION_COOKIE_PATH,
                       secure=settings.SESSION_COOKIE_SECURE or None)

def save(self, request, response, max_age, expires, must_create=False):
    for key in request.COOKIES.keys():
        if key.startswith('dj_session.'):
            key = key[11:]
            value = self.get(key, None)
            self.set_cookie(response, key, value)
    keys = request.COOKIES.keys()
    for key, value in self.items():
        if key and 'dj_session.' + key not in keys and value is not None:
            self.set_cookie(response, key, value)

def exists(self, session_key):
    return False

def delete(self, session_key=None):

```

```
for key in self.keys():  
    del self[key]  
self.modified = True
```

## JavaScript Plugin

We have modified [jQuery cookie plugin](http://ftp.pimentech.net/src/pimentech-scripts/scripts/src/js/jquery.cookie.js). You can get it at : <http://ftp.pimentech.net/src/pimentech-scripts/scripts/src/js/jquery.cookie.js>