django-browserid Documentation

Release 0.9

Paul Osman, Michael Kelly

August 25, 2013

CONTENTS

django-browserid is a library that integrates BrowserID authentication into Django. By default it relies on the Persona Identity Provider.

django-browserid provides an authentication backend, `BrowserIDBackend`, that verifies BrowserID assertions using a BrowserID verification service and authenticates users. It also provides `verify`, which lets you build more complex authentication systems based on BrowserID.

django-browserid is a work in progress. Contributions are welcome. Feel free to fork and contribute!

CHAPTER

ONE

SETUP

1.1 Installation

You can use pip to install django-browserid and requirements:

```
pip install django-browserid
```

1.2 Configuration

To use `django-browserid`, you'll need to make a few changes to your `settings.py` file:

```
# Add 'django_browserid' to INSTALLED_APPS.
INSTALLED_APPS = (
    # ...
    'django.contrib.auth',
    'django_browserid',  # Load after auth
    # ...
)

# Add the django_browserid authentication backend.
AUTHENTICATION_BACKENDS = (
   # ...
   'django.contrib.auth.backends.ModelBackend', # required for admin
   'django_browserid.auth.BrowserIDBackend',
   # ...
)

# Add the django_browserid context processor.
TEMPLATE_CONTEXT_PROCESSORS = (
   # ...
   'django_browserid.context_processors.browserid',
   # ...
)

# Set your site url for security
SITE_URL = 'https://example.com'
```

Note: BrowserID uses an assertion and an audience to verify the user. This `SITE_URL` is used to determine the audience. It can be a string or an iterable of strings.

For security reasons, it is *very important* that you set `SITE_URL` correctly.

Note: `TEMPLATE_CONTEXT_PROCESSORS` is not in the settings file by default. You can find the default value in the Context Processor documentation.

Next, edit your `urls.py` file and add the following:

```
urlpatterns = patterns('',
    # ...
    (r'^browserid/', include('django_browserid.urls')),
    # ...
)
```

You can also set the following optional settings in `settings.py`:

```
# Path to redirect to on successful login.
LOGIN_REDIRECT_URL = '/'

# Path to redirect to on unsuccessful login attempt.
LOGIN_REDIRECT_URL_FAILURE = '/'

# Path to redirect to on logout.
LOGOUT_REDIRECT_URL = '/'
```

Finally, you'll need to add the login button to your templates. There are three things you will need to add to your templates:

1. `{% browserid_info %}`: Outputs an invisible element that stores info about the current user. Must be within the `<body>` tag and appear only **once**.
2. `{% browserid_js %}`: Outputs the `<script>` tags for the button JavaScript. Must be somewhere on the page, typically at the bottom right before the `</body>` tag to allow the page to visibly load before executing.
3. `{% browserid_css %}`: Outputs `<link>` tags for optional CSS that styles login buttons to match Persona.
4. `{% browserid_login %}` and `{% browserid_logout %}`: Outputs the HTML for the login and logout buttons.

A complete example:

```
{% load browserid %}
<html>
  <head>
    {% browserid_css %}
  </head>
  <body>
    {% browserid_info %}
    <header>
      <h1>My Site</h1>
      <div class="authentication">
        {% if user.is_authenticated %}
          {% browserid_logout text='Logout' %}
        {% else %}
          {% browserid_login text='Login' color='dark' %}
        {% endif %}
      </div>
    </header>
    <article>
      <p>Welcome to my site!</p>
    </article>
```

```
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    {% browserid_js %}
  </body>
</html>
```

If you're using Jinja2 as your templating system, you can use the functions passed to your template by the context processor:

```
<html>
  <head>
    {{ browserid_css() }}
  </head>
  <body>
    {{ browserid_info() }}
    <header>
      <h1>My Site</h1>
      <div class="authentication">
        {% if user.is_authenticated() %}
          {{ browserid_logout(text='Logout') }}
        {% else %}
          {{ browserid_login(text='Login', color='dark') }}
        {% endif %}
      </div>
    </header>
    <article>
      <p>Welcome to my site!</p>
    </article>
    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    {{ browserid_js() }}
  </body>
</html>
```

Note: The JavaScript assumes you have jQuery 1.7 or higher on your site.

Note: For more information about the template helper functions, check out the *API* document.

1.3 Deploying to Production

There are a few changes you need to make when deploying your app to production:

- BrowserID uses an assertion and an audience to verify the user. The `SITE_URL` setting is used to determine the audience. For security reasons, it is *very important* that you set `SITE_URL` correctly.

  `SITE_URL` should be set to the domain and protocol users will use to access your site, such as `https://affiliates.mozilla.org`. This URL does not have to be publicly available, however, so sites limited to a certain network can still use django-browserid.

1.4 Static Files

`browserid_js` and `browserid_css` use Form Media and the Django staticfiles app to serve the static files for the buttons. If you don't want to use the static files framework, you'll need to include the JavaScript and CSS manually on any page you use the `browserid_button` function.

For `browserid_js` the files needed are the Persona JavaScript shim, which should be loaded from `https://login.persona.org/include.js` in a script tag, and `django_browserid/static/browserid/browserid.js`, which is part of the django-browserid library.

For `browserid_css` the file needed is `django_browserid/static/browserid/persona-buttons.css`, which is also part of the django-browserid library.

1.5 Content Security Policy

If your site uses Content Security Policy, you will have to add directives to allow the external persona.org JavaScript, as well as an iframe used as part of the login process.

If you're using django-csp, the following settings will work:

```
CSP_SCRIPT_SRC = ("'self'", 'https://login.persona.org')
CSP_FRAME_SRC = ("'self'", 'https://login.persona.org')
```

1.6 Alternate Template Languages (Jingo/Jinja)

If you are using a library like Jingo in order to use a template language besides the Django template language, you may need to configure the library to use the Django template language for django-browserid templates. With Jingo, you can do this using the `JINGO_EXCLUDE_APPS` setting:

```
JINGO_EXCLUDE_APPS = ('browserid',)
```

1.7 Troubleshooting Issues

If you run into any issues while setting up django-browserid, try the following steps:

1. Check for any warnings in the server log. You may have to edit your development server's logging settings to output `django_browserid` log entries. Here's an example `LOGGING` setup to start with:

   ```
   LOGGING = {
       'version': 1,
       'handlers': {
           'console':{
               'level': 'DEBUG',
               'class': 'logging.StreamHandler'
           },
       },
       'loggers': {
           'django_browserid': {
               'handlers': ['console'],
               'level': 'DEBUG',
           }
       },
    }
   ```

2. Check the *Troubleshooting* document for commonly-reported issues.
3. Ask for help in the #webdev channel on irc.mozilla.org.
4. Post an issue on the django-browserid Issue Tracker.

CHAPTER

TWO

ADVANCED USAGE

2.1 Automatic Account Creation

`django-browserid` will automatically create a user account for new users. The user account will be created with the verified email returned from the BrowserID verification service, and a URL safe base64 encoded SHA1 of the email with the padding removed as the username.

To provide a customized username, you can provide a different algorithm via your settings.py:

```
# settings.py
BROWSERID_CREATE_USER = True
def username(email):
    return email.rsplit('@', 1)[0]
BROWSERID_USERNAME_ALGO = username
```

You can can provide your own function to create users by setting `BROWSERID_CREATE_USER` to a string path pointing to a function:

```
# module/util.py
def create_user(email):
    return User.objects.create_user(email, email)

# settings.py
BROWSERID_CREATE_USER = 'module.util.create_user'
```

You can disable account creation, but continue to use the `browserid_verify` view to authenticate existing users with the following:

```
BROWSERID_CREATE_USER = False
```

2.2 Custom Verification

If you want to customize the verification view, you can do so by subclassing `django_browserid.views.Verify` and overriding the methods to insert your custom logic.

If you want complete control over account verification, you should create your own view and use `django_browserid.verify()` to manually verify a BrowserID assertion with something like the following:

```
from django_browserid import get_audience, verify
from django_browserid.forms import BrowserIDForm
```

```
def myview(request):
    # ...
    if request.method == 'POST':
        form = BrowserIDForm(data=request.POST)
        if form.is_valid():
            result = verify(form.cleaned_data['assertion'], get_audience(request))
            if result:
                # check for user account, create account for new users, etc
                user = my_get_or_create_user(result['email'])
```

See `django_browserid.verify()` for more info on what `verify` returns.

2.3 Custom User Model

Django 1.5 allows you to specify a custom model to use in place of the built-in User model with the `AUTH_USER_MODEL` setting. `django-browserid` supports custom User models, but you will most likely need to add a few extra customizations to make things work properly:

- `django_browserid.BrowserIDBackend` has three methods that deal with User objects: `create_user`, `get_user`, and `filter_users_by_email`. You may have to subclass `BrowserIDBackend` and override these methods to work with your custom User class.
- `browserid_login` assumes that your custom User class has an attribute called `email` that contains the user's email address. You can either add an email field to your model, or add a property to the model that returns the user's email address.

2.4 Custom Verify view

You can override which class is the view class for doing the verification. This can be useful in the case where you want to override certain methods that you need to work differently. To do this, set `BROWSERID_VERIFY_CLASS` to the path of your own preferred class.

Here's an example:

```
# settings.py
BROWSERID_VERIFY_CLASS = 'myapp.MyVerifyClass'

# myapp.py
from django_browserid.views import Verify
class MyVerifyClass(Verify):
    @property
    def success_url(self):
        if self.user.username == 'Satan':
            return '/hell'
        # the default behaviour
        return getattr(settings, 'LOGIN_REDIRECT_URL', '/')
```

CHAPTER

THREE

SETTINGS

3.1 Core Settings

`django.conf.settings.`**`SITE_URL`**
Default: No default

Domain and protocol used to access your site. BrowserID uses this value to determine if an assertion was meant for your site.

Can be a string or an iterable of strings.

Note that this does not have to be a publicly accessible URL, so local URLs like `localhost:8000` or `127.0.0.1` are acceptable as long as they match what you are using to access your site.

3.2 Redirect URLs

`django.conf.settings.`**`LOGIN_REDIRECT_URL`**
Default: `'/accounts/profile'`

Path to redirect to on successful login. If you don't specify this, the default Django value will be used.

`django.conf.settings.`**`LOGIN_REDIRECT_URL_FAILURE`**
Default: `'/'`

Path to redirect to on an unsuccessful login attempt.

`django.conf.settings.`**`LOGOUT_REDIRECT_URL`**
Default: `'/'`

Path to redirect to on logout.

3.3 Customizing the Login Popup

`django.conf.settings.`**`BROWSERID_REQUEST_ARGS`**
Default: `{}`

Controls the arguments passed to `navigator.id.request`, which are used to customize the login popup box. To see a list of valid keys and what they do, check out the navigator.id.request documentation.

3.4 Customizing the Verify View

`django.conf.settings.`**`BROWSERID_VERIFY_VIEW`**
Default: `django_browserid.views.Verify`

Allows you to substitute a custom class-based view for verifying assertions. For example, the string 'myapp.users.views.Verify' would import *Verify* from *myapp.users.views* and use it in place of the default view.

When using a custom view, it is generally a good idea to subclass the default Verify and override the methods you want to change.

`django.conf.settings.`**`BROWSERID_CREATE_USER`**
Default: `True`

If `True` or `False`, enables or disables automatic user creation during authentication.

If set to a string, it is treated as an import path pointing to a custom user creation function. See *Automatic Account Creation* for more information.

`django.conf.settings.`**`BROWSERID_DISABLE_SANITY_CHECKS`**
Default: False

Controls whether the `Verify` view performs some helpful checks for common mistakes. Useful if you're getting warnings for things you know aren't errors.

3.5 Using a Different Identity Provider

`django.conf.settings.`**`BROWSERID_VERIFICATION_URL`**
Default: `'https://browserid.org/verify`

Defines the URL for the BrowserID verification service to use.

`django.conf.settings.`**`BROWSERID_SHIM`**
Default: 'https://login.persona.org/include.js'

The URL to use for the BrowserID JavaScript shim.

3.6 Customizing Verification

`django.conf.settings.`**`BROWSERID_DISABLE_CERT_CHECK`**
Default: `False`

Disables SSL certificate verification during BrowserID verification. *Never disable this in production!*

`django.conf.settings.`**`BROWSERID_CACERT_FILE`**
Default: `None`

CA cert file used during validation. If none is provided, the default file included with requests is used.

CHAPTER

FOUR

TROUBLESHOOTING

4.1 CSP WARN: Directive "..." violated by https://browserid.org/include.js

This warning appears in the Error Console when your site uses Content Security Policy without making an exception for the persona.org external JavaScript include.

To fix this, include https://persona.org in your script-src and frame-src directive. If you're using the django-csp library, the following settings will work:

```
CSP_SCRIPT_SRC = ("'self'", https://login.persona.org')
CSP_FRAME_SRC = ("'self'", 'https://login.persona.org')
```

4.2 Login fails silently due to SESSION_COOKIE_SECURE

If you try to login on a local instance of a site and login fails without any error (typically redirecting you back to the login page), check to see if you've set *SESSION_COOKIE_SECURE* to True in your settings.

SESSION_COOKIE_SECURE controls if the *secure* flag is set on the session cookie. If set to True on a local instance of a site that does not use HTTPS, the session cookie won't be sent by your browser because you're using an HTTP connection.

The solution is to set *SESSION_COOKIE_SECURE* to False on your local instance, typically by adding it to *settings/local.py*:

```
SESSION_COOKIE_SECURE = False
```

4.3 Login fails silently due to cache issues

Another possible cause of silently failing logins is an issue with having no cache configured locally. Several projects (especially projects based on playdoh, which uses django-session-csrf) store session info in the cache rather than the database, and if your local instance has no cache configured, the session information will not be stored and login will fail silently.

To solve this issue, you should configure your local instance to use an in-memory cache with the following in your local settings file:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'unique-snowflake'
    }
}
```

CHAPTER

FIVE

API

5.1 Template Helpers

django_browserid.helpers.**browserid_info**()

Output the HTML for the login form and the info tag. Should be called once at the top of the page just below the <body> tag.

django_browserid.helpers.**browserid_login**(*text='Sign in'*, *color=None*, *next=None*, *link_class='browserid-login'*, *attrs=None*, *fallback_href='#'*)

Output the HTML for a BrowserID login link.

Parameters

- **text** – Text to use inside the link. Defaults to 'Sign in', which is not localized.
- **color** – Color to use for the login button; this will only work if you have included the default CSS provided by `django_browserid.helpers.browserid_css()`.

  Supported colors are: *'dark'*, *'blue'*, and *'orange'*.
- **next** – URL to redirect users to after they login from this link. If omitted, the LOGIN_REDIRECT_URL setting will be used.
- **link_class** – CSS class for the link. *browserid-login* will be added to this automatically.
- **attrs** – Dictionary of attributes to add to the link. Values here override those set by other arguments.

  If given a string, it is parsed as JSON and is expected to be an object.
- **fallback_href** – Value to use for the href of the link. If the user has disabled JavaScript, the login link will bring them to this page, which can be used as a non-JavaScript login fallback.

django_browserid.helpers.**browserid_logout**(*text='Sign out'*, *link_class='browserid-logout'*, *attrs=None*)

Output the HTML for a BrowserID logout link.

Parameters

- **text** – Text to use inside the link. Defaults to 'Sign out', which is not localized.
- **link_class** – CSS class for the link. *browserid-logout* will be added to this automatically.
- **attrs** – Dictionary of attributes to add to the link. Values here override those set by other arguments.

  If given a string, it is parsed as JSON and is expected to be an object.

django_browserid.helpers.**browserid_js**(*include_shim=True*)
Returns <script> tags for the JavaScript required by the BrowserID login button. Requires use of the staticfiles app.

Parameters **include_shim** – A boolean that determines if the persona.org JavaScript shim is included in the output. Useful if you want to minify the button JavaScript using a library like django-compressor that can't handle external JavaScript.

5.2 Verification Functions

django_browserid.**verify**(*assertion*, *audience*, *extra_params=None*, *url=None*)
Verify assertion using an external verification service.

Parameters

- **assertion** – The string assertion received in the client from `navigator.id.request()`.
- **audience** – This is domain of your website and it must match what was in the URL bar when the client asked for an assertion. You probably want to use `django_browserid.get_audience()` which sets it based on `SITE_URL`.
- **extra_params** – A dict of additional parameters to send to the verification service as part of the POST request.
- **url** – A custom verification URL for the service. The service URL can also be set using the `BROWSERID_VERIFICATION_URL` setting.

Returns

A dictionary similar to the following:

```
{
    u'audience': u'https://mysite.com:443',
    u'email': u'myemail@example.com',
    u'issuer': u'browserid.org',
    u'status': u'okay',
    u'expires': 1311377222765
}
```

Raises BrowserIDException: Error connecting to remote verification service.

django_browserid.**get_audience**(*request*)
Uses Django settings to format the audience.

To figure out the audience to use, it does this:

1. If settings.DEBUG is True and settings.SITE_URL is not set or empty, then the domain on the request will be used.

   This is *not* secure!

2. Otherwise, settings.SITE_URL is checked for the request domain and an ImproperlyConfigured error is raised if it is not found.

Examples of settings.SITE_URL:

```
SITE_URL = 'http://127.0.0.1:8001'
SITE_URL = 'https://example.com'
SITE_URL = 'http://example.com'
```

```
SITE_URL = (
    'http://127.0.0.1:8001',
    'https://example.com',
    'http://example.com'
)
```

5.3 Views

class django_browserid.views.**Verify**(***kwargs*)
Bases: django.views.generic.edit.BaseFormView

Login view for django-browserid. Takes in an assertion and sends it to the remote verification service to be verified, and logs in the user upon success.

failure_url
URL to redirect users to when login fails. This uses the value of settings.LOGIN_REDIRECT_URL_FAILURE, and defaults to '/' if the setting doesn't exist.

success_url
URL to redirect users to when login succeeds if next isn't specified in the request. This uses the value of settings.LOGIN_REDIRECT_URL, and defaults to '/' if the setting doesn't exist.

login_success()
Log the user into the site and redirect them to the post-login URL.

If next is found in the request parameters, it's value will be used as the URL to redirect to. If next points to a different host than the current request, it is ignored.

login_failure(*error=None*)
Redirect the user to a login-failed page, and add the bid_login_failed parameter to the URL to signify that login failed to the JavaScript.

Parameters **error** – If login failed due to an error raised during verification, this will be the BrowserIDException instance that was raised.

form_valid(*form*)
Send the given assertion to the remote verification service and, depending on the result, trigger login success or failure.

Parameters **form** – Instance of BrowserIDForm that was submitted by the user.

form_invalid(**args*, ***kwargs*)
Trigger login failure since the form is invalid.

get(**args*, ***kwargs*)
Trigger login failure since we don't support GET on this view.

get_failure_url()
Retrieve *failure_url* from the class. Raises ImproperlyConfigured if the attribute is not found.

dispatch(*request*, **args*, ***kwargs*)
Run some sanity checks on the request prior to dispatching it.

5.4 Signals

`django_browserid.signals.`**`user_created`** **= <django.dispatch.dispatcher.Signal object at 0x206b490>**
Signal triggered when a user is automatically created during authentication.

Parameters

- **sender** – The function that created the user instance.
- **user** – The user instance that was created.

5.5 Exceptions

exception `django_browserid.base.`**`BrowserIDException`**`(`*exc*`)`
Raised when there is an issue verifying an assertion with `django_browserid.base.verify()`.

`exc` **= None**
Original exception that caused this to be raised.

CHAPTER

SIX

JAVASCRIPT API

The JavaScript file that comes with django-browserid, `browserid.js`, includes a a few public functions that are exposed through the `django_browserid` global object.

`django_browserid.`**`login`**`(`[*next*, *requestArgs*]`)`
Manually trigger BrowserID login.

Arguments

- **next** (*string*) – URL to redirect the user to after login.
- **requestArgs** (*object*) – Options to pass to navigator.id.request.

`django_browserid.`**`logout`**`(`[*next*]`)`
Manually trigger BrowserID logout.

Arguments

- **next** (*string*) – URL to redirect the user to after logout.

`django_browserid.`**`isUserAuthenticated`**`()`
Check if the current user has authenticated via django_browserid. Note that this relies on the `#browserid-info` element having been loaded into the DOM already. If it hasn't, this will return false.

Returns True if the user has authenticated, false otherwise.

`django_browserid.`**`getAssertion`**`(`*callback*`)`
Retrieve an assertion from BrowserID and execute the given callback with the assertion as the single argument.

Arguments

- **callback** (*function*) – Callback to execute after the assertion has been retrieved.

`django_browserid.`**`verifyAssertion`**`(`*assertion*[, *redirectTo*]`)`
Verify an assertion, and redirect to a URL on success. Calling this method submits a form to the server and changes the current page as if the user was attempting to login.

Arguments

- **assertion** (*string*) – Assertion to verify.
- **redirectTo** (*string*) – URL to redirect to on success.

CHAPTER

SEVEN

DEVELOPER GUIDE

7.1 Developer Setup

Check out the code from the github project:

```
git clone git://github.com/mozilla/django-browserid.git
cd django-browserid
```

Create a virtualenv (the example here uses virtualenvwrapper) and install all development packages:

```
mkvirtualenv django-browserid
pip install -r requirements.txt
```

Here is how to run the test suite:

```
python runtests.py
```

Here is how to build the documentation:

```
make -C docs/ html
```

7.2 Changelog

7.2.1 History

0.9 (2013-08-25)

- Add `BROWSERID_VERIFY_CLASS` to make it easier to customize the verification view.
- Add hook to authentication backend for validating the user's email.
- Ensure backend attribute exists on user objects authenticated by django-browserid.
- Prevent installation of the library as an unpackaged egg.
- Add incomplete Python 3 support.
- Fix an issue where users who logged in without Persona were being submitted to `navigator.id.watch` anyway.
- Add CSS to make the login/logout buttons prettier.
- Support for `SITE_URL` being an iterable.

- Add support for lazily-evaluated `BROWSERID_REQUEST_ARGS`.
- Add a small JavaScript API available on pages that include `browserid.js`.
- Support running tests via *python setup.py test*.
- Fix an infinite loop where logging in with a valid Persona account while `BROWSERID_CREATE_USER` is true would cause an infinite redirection.

0.8 (2013-03-05)

- #97: Add BrowserIDException that is raised by verify when there are issues connecting to the remote verification servie. Update the Verify view to handle these errors.
- #125: Prevent the Verify view from running reverse on user input and add check to not redirect to URLs with a different host.
- Remove ability to set a custom name for the Verify redirect parameter: it's just `next`.
- Replace `browserid_button` with `browserid_login` and `browserid_logout`, and make `browserid_info` a function.
- #109: Fix issue with unicode strings in the `extra_params` kwarg for `verify`.
- #110: Fix bug where kwargs to `authenticate` get passed as `extra_params` to verify. Instead, you can pass any extra parameters in `browserid_extra`. But please don't, it's undocumented for a reason. <3
- #105: General documentation fixes, add more debug logging for common issues. Add `BROWSERID_DISABLE_SANITY_CHECKS` setting and remove the need to set `SITE_URL` in development.
- Add `form_extras` parameter to `browserid_button`.
- #101, #102: Update the default JavaScript to pass the current user's email address into `navigator.id.watch` to avoid unnecessary auto-login attempts.
- Add template functions/tags to use for embedding login/logout buttons instead of using your own custom HTML.
- Add a `url` kwarg to `verify` that lets you specify a custom verification service to use.
- Add documentation for setting up the library for development.
- #103: `BrowserIDForm` now fails validation if the assertion given is non-ASCII.
- Fix an error in the sample urlconf in the documentation.
- #98: Fix a bug where login or logout buttons might not be detected by the default JavaScript correctly if `<a>` element contained extra HTML.
- Add `pass_mock` kwarg to `mock_browserid`, which adds a new argument to the front of the decorated method that is filled with the Mock object used in place of `_verify_http_request`.
- Any extra kwargs to `BrowserIDBackend.authenticate` are passed in the verify request as POST arguments (this will soon be removed, don't rely on it).

0.7.1 (2012-11-08)

- Add support for a working logout button. Switching to the Observer API in 0.7 made the issue that we weren't calling `navigator.id.logout` more pronounced, so it makes sense to make a small new release to make it easier to add a logout button.

0.7 (2012-11-07)

- Actually start updating the Changelog again.
- Remove deprecated functions `django_browserid.auth.get_audience` and `django_browserid.auth.BrowserIDBackend.verify`, as well as support for `DOMAIN` and `PROTOCOL` settings.
- Add small fix for infinite login loops.
- Add automated testing for Django 1.3.4, 1.4.2, and 1.5a1.
- Switch to using `format` for all string formatting (**breaks Python 2.5 compatibility**).
- Add support for Django 1.5 Custom User Models.
- Fix request timeouts so that they work properly.
- Add ability to customize BrowserID login popup via arguments to `navigator.id.request`.
- Update JavaScript to use the new Observer API.
- Change `browserid.org` urls to `login.persona.org`.

7.3 Authors

`django-browserid` is written and maintained by various contributors:

7.3.1 Current Maintainer

- Michael Kelly <mkelly@mozilla.com>

7.3.2 Previous Maintainers

- Paul Osman
- Austin King
- Ben Adida

7.3.3 Patches and Suggestions

- Thomas Grainger
- Owen Coutts
- Francois Marier
- Andy McKay
- Giorgos Logiotatidis
- Alexis Metaireau
- Rob Hudson
- Ross Bruniges
- Les Orchard

- Charlie DeTar
- Luke Crouch
- shaib
- Kumar McMillan
- Carl Meyer
- ptgolden
- Will Kahn-Greene
- Allen Short
- meehow
- Greg Koberger
- Niran Babalola
- callmekatootie
- Paul Mclanahan
- JR Conlin
- Prasoon Shukla
- Peter Bengtsson
- Javed Khan
- Kalail (Kashif Malik)
- Richard Mansfield
- Francesco Pischedda

PYTHON MODULE INDEX

d