Django-Web3-Auth Documentation

Release 0.1.6

Denis Bobrov

Aug 11, 2023

Contents

Contents:

CHAPTER 1

Django-Web3-Auth

django-web3-auth is a pluggable Django app that enables login/signup via an Ethereum wallet (a la CryptoKitties). The user authenticates themselves by digitally signing the session key with their wallet's private key.

1.1 Documentation

The full documentation is at https://django-web3-auth.readthedocs.io.

1.2 Example project

https://github.com/Bearle/django-web3-auth/tree/master/example

You can check out our example project by cloning the repo and heading into example/ directory. There is a README file for you to check, also.

1.3 Features

- Web3 API login, signup
- Web3 Django forms for signup, login
- Checks ethereum address validity
- Uses random token signing as proof of private key posession
- Easy to set up and use (just one click)
- Custom auth backend
- VERY customizable - uses Django settings, allows for custom User model

- Vanilla Javascript helpers included

1.4 Quickstart

Install Django-Web3-Auth with pip:

```
pip install django-web3-auth
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'web3auth.apps.Web3AuthConfig',
    ...
)
```

Set *'web3auth.backend.Web3Backend'* as your authentication backend:

```
AUTHENTICATION_BACKENDS = [
'django.contrib.auth.backends.ModelBackend',
'web3auth.backend.Web3Backend'
]
```

Set your User model's field to use as ETH address provider:

```
WEB3AUTH_USER_ADDRESS_FIELD = 'username'
```

And if you have some other fields you want to be in the SignupForm, add them too:

```
WEB3AUTH_USER_SIGNUP_FIELDS = ['email',]
```

Add Django-Web3-Auth's URL patterns:

```
from web3auth import urls as web3auth_urls


urlpatterns = [
    ...
    url(r'^', include(web3auth_urls)),
    ...
]
```

Add some javascript to handle login:

```
<script src="{% static 'web3auth/js/web3auth.js' %}"></script>
```

```
function startLogin() {
  if (typeof web3 !== 'undefined') {
    checkWeb3(function (loggedIn) {
      if (!loggedIn) {
        alert("Please unlock your web3 provider (probably, Metamask)")
      } else {
        var login_url = '{% url 'web3auth:web3auth_login_api' %}';
        web3Login(login_url, console.log, console.log, console.log, console.log,␣
↪console.log, function (resp) {
```

(continues on next page)

(continued from previous page)

```
          console.log(resp);
          window.location.replace(resp.redirect_url);
        });
      }
    });

  } else {
    alert('web3 missing');
  }
}
```

You can access signup using {% url 'web3auth:web3auth_signup' %}.

If you have any questions left, head to the example app https://github.com/Bearle/django-web3-auth/tree/master/example

1.5 Important details and FAQ

1. *If you set a custom address field (WEB3AUTH_USER_ADDRESS_FIELD), it MUST be unique (unique=True).*

This is needed because if it's not, the user can register a new account with the same address as the other one, meaning that the user can now login as any of those accounts (sometimes being the wrong one).

2. *How do i deal with user passwords or Password is not set* There should be some code in your project that generates a password using `User.objects.make_random_password` and sends it to a user email. Or, even better, sends them a 'restore password' link. Also, it's possible to copy signup_view to your project, assign it a url, and add the corresponding lines to set some password for a user.

3. *Why do i have to sign a message? It's not needed in MyEtherWallet or other DApps!*

The main reason is that when using a DApp, you most likely don't have an account on the website, it's accessible only with web3 (Metamask). When using web3 only to sign into user account, it is necessary to prove your identity with a private key (e.g. sign a random message), because when we have backend we can't trust any user just by his knowledge of the public address. Signed message proves that user possesses the private key, associated with the address.

1.6 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.7 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Overview

Django-web3-auth features 1 view for login (with JSON responses) and 2 views for Signup (one with JSON responses, and the other - using Django Forms and rendered templates).

It also has 2 forms, SignupForm (rendered) and LoginForm (uses hidden inputs, used to validate data only).

Possible configuration includes customizable address field (`WEB3AUTH_USER_ADDRESS_FIELD`), additional fields for User model (`WEB3AUTH_USER_SIGNUP_FIELDS`) and on/off switch for registration (`WEB3AUTH_SIGNUP_ENABLED`). You can read more on that in the Configuration section.

2.1 Sign up

The signup process is as follows (signup_view example, signup_api is similar):

1. User heads to the signup URL (`{% url 'web3auth:web3auth_signup' %}`)
2. The signup view is rendered with a `SignupForm` which includes `WEB3AUTH_USER_SIGNUP_FIELDS` and `WEB3AUTH_USER_ADDRESS_FIELD`
3. The user enters required data and clicks the submit button and the POST request fires to the same URL with `signup_view`
4. **Signup view does the following:** 4.1. Creates an instance of a `SignupForm`. 4.2. Checks if the registration is enabled. 4.3. If the registration is closed or form has errors, returns form with errors 4.4 If the form is valid, saves the user without saving to DB 4.5. Sets the user address from the form, saves it to DB 4.6. Logins the user using `web3auth.backend.Web3Backend` 4.7. Redirects the user to `LOGIN_REDIRECT_URL` or 'next' in get or post params
5. The user is signed up and logged in

2.2 Login

The login process is as follows (login_api example):

1. On some page of the website, there is Javascript which fires a GET request to the `{% url 'web3auth:web3auth_login_api' %}`
2. The `login_api` view returns 32-char length login token
3. Javascript on the page invites user to sign the token using web3 instance (probably Metamask)
4. If the token is signed, the signature and address are sent ot he same `login_api` view
5. The view validates signature & address against `LoginForm` to check that the token is signed correctly
6. If the form is valid, the view tries to `authenticate` the user with given token,address and signature
7. If the user is found, the user is signed in and the view responds with a `redirect_url` for Javascript to handle
8. If the user is not found, the corresponding error is returned

The Javascript is included in the app, also you can check out example app if you are struggling with logging in the user.

CHAPTER 3

Settings

You should specify settings in your settings.py like this:

```
WEB3AUTH_USER_ADDRESS_FIELD = 'address'
WEB3AUTH_USER_SIGNUP_FIELDS = ['email', 'username']
```

In the above example the following User model is used:

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import ugettext_lazy as _
from web3auth.utils import validate_eth_address

class User(AbstractUser):
    address = models.CharField(max_length=42, verbose_name=_("Ethereum wallet address
↪"), unique=True,
                               validators=[validate_eth_address], null=True, blank=True)

    def __str__(self):
        return self.username
```

Here's a list of available settings:

Setting	Default	Description
WEB3AUTH_SIGNUP_ENABLED	True	If False, new users won't be able to sign up (used in `signup_view`)
WEB3AUTH_USER_SIGNUP_FIELDS	['email']	Specifies field to be used in signup form for a new User model
WEB3AUTH_USER_ADDRESS_FIELD	'username'	Field on the User model, which has ethereum address to check against.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Bearle/django-web3-auth/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

Django-Web3-Auth could always use more documentation, whether as part of the official Django-Web3-Auth docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Bearle/django-web3-auth/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-web3-auth* for local development.

1. Fork the *django-web3-auth* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-web3-auth.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv django-web3-auth
   $ cd django-web3-auth/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 web3auth tests
   $ python setup.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/Bearle/django-web3-auth/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_web3auth
```

CHAPTER 5

Credits

5.1 Development Lead

- Denis Bobrov <tech@bearle.ru>

5.2 Contributors

- Alexander Tereshkin <atereshkin@y-node.com>

CHAPTER 6

0.1.6 (2021-09-06)

- Update from PR#6 by @hiporox
- Resolve's issue #4 - Example app has missing url configuration

CHAPTER 7

0.1.5 (2021-09-06)

- Update from PR#5 by @hiporox
- Updated .gitignore to include missing file types
- Added script tag to the base.html that imports web3 since MetaMask no longer auto imports (https://docs.metamask.io/guide/provider-migration.html#replacing-window-web3)

7.1 History

CHAPTER 8

0.1.4 (2021-05-06)

- Try fix rlp

CHAPTER 9

0.1.3 (2021-03-23)

- Try fix ethereum

CHAPTER 10

0.1.2 (2021-03-16)

- Flake8, tox fixes in PR#2 by SukiCZ

CHAPTER 11

0.1.1 (2021-03-16)

- Bump 'rlp' - PR#1 by SukiCZ

CHAPTER 12

0.1.0 (2018-06-29)

- First release on PyPi