Social sign in with single-page app and JWT server validation

Article by Ole Michelsen *posted on* January 24, 2016

Social sign in is ubiquitous nowadays, and if you are running a Single-Page App (SPA), you can sign in without ever reloading the page. This will allow your app to talk to all the social networks like Facebook and Twitter, and you can access profile info, friends/contacts, photos and more, all without handling anything on your server.

However if you *do* need to send some data to your server at some point, you need to ensure that the users posting to your server, are actually who they say they are. This tutorial will go through the sign in process in a SPA, and validating the access tokens we recieve on our own server. First let's look at how to create a simple sign in page.

Client app and hello.js

To sign in with a social network provider actually turns out to be the easiest part. Using the great library hello.js you just have to sign up for a client key with the providers you wish to support (e.g. Facebook) and call `login()`.

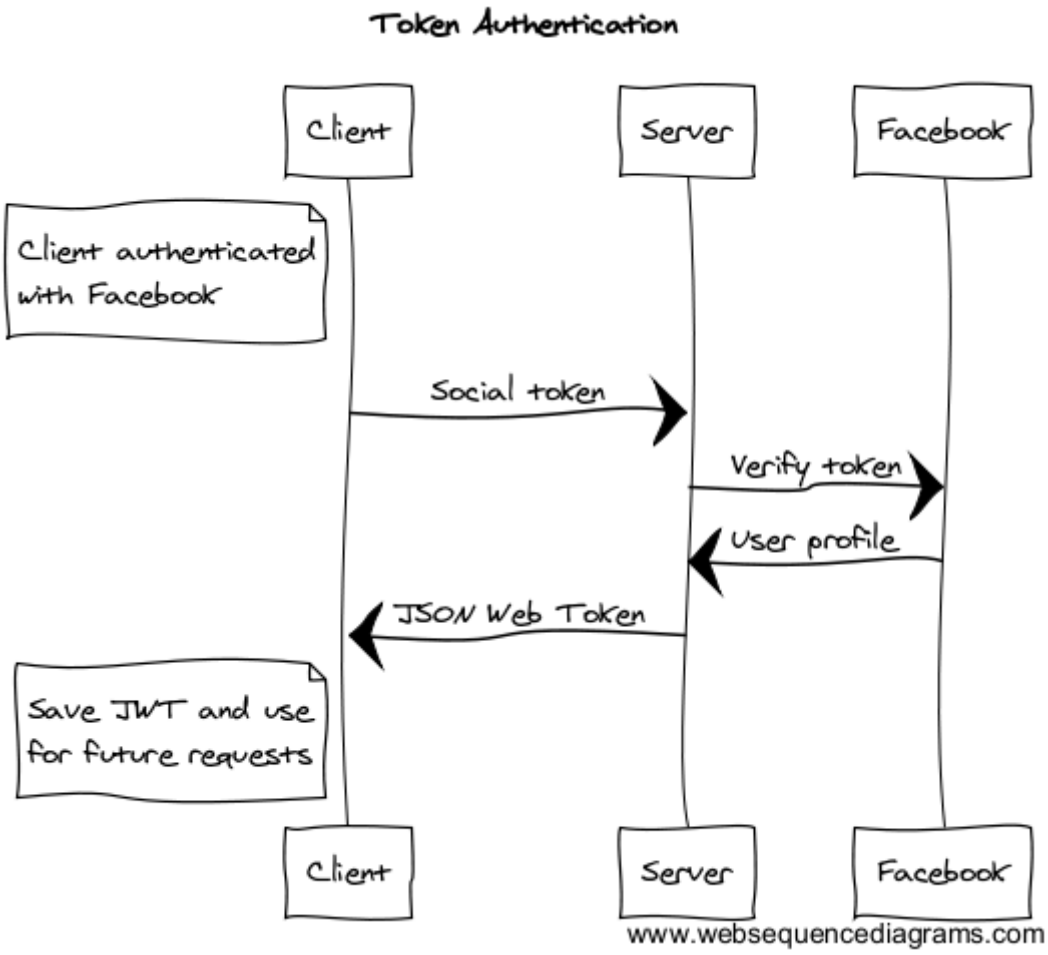
```
hello.init({
    facebook: FACEBOOK_CLIENT_ID,
    windows: WINDOWS_CLIENT_ID,
    google: GOOGLE_CLIENT_ID
}, {
    redirect_uri: 'redirect.html'
});
```

```
<button onclick="hello('facebook').login()">Facebook</button>
```

This bit of code allows us to sign in with Facebook, and if you check out the API for hello.js, you can access most data from the providers you want to support.

Authenticating with the server

But now we want to allow the user to save some data on our server. Since we have authenticated client-side with Facebook, we have an authentication token from Facebook. This is used to validate our session when talking to Facebook. We need to establish the same level of trust with our own server. This diagram shows the flow we are trying to achieve:



To secure communications between our client and server, we will first send the token to our server for verification. This snippet will POST the token to our own endpoint `/api/auth` using the small AJAX lib superagent:

```
var socialToken;
var serverToken;

hello.on('auth.login', function (auth) {
    // Save the social token
    socialToken = auth.authResponse.access_token;

    // Auth with our own server using the social token
    authenticate(auth.network, socialToken).then(function (token) {
        serverToken = token;
    });
});

function authenticate(network, socialToken) {
    return new Promise(function (resolve, reject) {
        request
            .post('/api/auth')
            .send({
                network: network,
                socialToken: socialToken
            })
            .set('Accept', 'application/json')
            .end(function(err, res){
                if (err) {
                    reject(err);
                } else {
                    resolve(res);
```

```
            }
        });
    });
}
```

Validate social access tokens on the server

Now let's see how to handle it on the server side. I show the example as a simple node.js/Express app, but you can implement it in any server language you like (PHP, Ruby etc.). You can find this example on Github.

We want to verify with Facebook, that the token is valid and who the user is. To do this we just have to send the token to the Facebook API like this:

```
app.post('/api/auth', function (req, res) {
    // Grab the social network and token
    var network = req.body.network;
    var socialToken = req.body.socialToken;

    // Validate the social token with Facebook
    validateWithProvider(network, socialToken).then(function (profile) {
        // Return the user data we got from Facebook
        res.send('Authenticated as: ' + profile.id);
    }).catch(function (err) {
        res.send('Failed!' + err.message);
    });
});


var providers = {
    facebook: {
        url: 'https://graph.facebook.com/me'
    }
};

function validateWithProvider(network, socialToken) {
    return new Promise(function (resolve, reject) {
        // Send a GET request to Facebook with the token as query string
        request({
                url: providers[network].url,
                qs: {access_token: socialToken}
            },
            function (error, response, body) {
                if (!error && response.statusCode == 200) {
                    resolve(JSON.parse(body));
                } else {
                    reject(err);
                }
            }
        );
    });
}
```

JSON Web Token

Now we have validated the authenticity of the user, we can give the client our own signed JSON Web Token (JWT) that contains the info of the authenticated user. This token needs

to be sent along with all future requests, and our server can then validate the request simply by verifying the token signature. This way we don't have to validate the social network token with e.g. Facebook on every request.

So we will change our `/api/auth` endpoint to return a JWT instead. Here I'm using the jsonwebtoken lib, but check out jwt.io for a library for your chosen server language.

```
...
    // Validate the social token with Facebook
    validateWithProvider(network, socialToken).then(function (profile) {
        // Return a server signed JWT
        res.send(createJwt(profile));
    }).catch(function (err) {
        res.send('Failed!' + err.message);
    });
...


function createJwt(profile) {
    return jwt.sign(profile, 'MY_PRIVATE_KEY', {
        expiresIn: '2h',
        issuer: 'MY_APP'
    });
}
```

We are returning a JWT signed with our own private key (you should never leak your private key to the client!), and containing the user profile data. It is important to implement the `expiresIn` value, as anybody with this JWT can make requests, so you will reduce the risk by forcing the user to do a new `/api/auth` request once in a while.

Verify the JWT

Now we just need to send this JWT along with any other requests we do to the server, verify that the signature is valid, and we are good to go. This example shows another endpoint on the server, which will verify that the JWT is good before returning data:

```
app.get('/secure', function (req, res) {
    var jwtString = req.query.jwt;

    try {
        var profile = verifyJwt(jwtString);
        res.send('You are good people: ' + profile.id);
    } catch (err) {
        res.send('Hey, you are not supposed to be here');
    }
});


function verifyJwt(jwtString) {
    return jwt.verify(jwtString, 'MY_PRIVATE_KEY', {
        issuer: 'MY_APP'
    });
}
```

You can send the JWT to your server any way you like, normally you would put it in a header, but query string, POST data or cookie is also fine.

It is possible to append any data to the JWT when you generate it on your server, so this is also a very convenient way to send some additional user data to your client-side app. Just remember that a JWT can be decoded to clear text by anyone, so don't put confidential info in there. The safety of a JWT comes from the fact that it is signed, so it can't be modified. It is not encrypted.

In conclusion this is probably the easiest way to handle social sign in with a single-page app, and still securing your server/API with just a few lines of code. All the network providers I have been using supports this form of auth token verification, and from their documentation, these are the URL's you can send the token to:

- Facebook: https://graph.facebook.com/me
- Github: https://api.github.com/user
- Google: https://www.googleapis.com/oauth2/v3/tokeninfo
- Windows: https://apis.live.net/v5.0/me

All of them accept the auth token as a query string in the form of `access_token=TOKEN`.

The code I have shown here is packaged as an example project on Github.

← Making an offline web app with Service Workers

Testing ReactJS with coverage using mocha+babel+nyc →

2 • Reply • Share ›

Ole Bjørn Michelsen Mod ➦ yaron levi • 2 years ago

Sure, that should work as well.

• Reply • Share ›

Yoni Goldberg • 23 days ago

Great article!

May I ask - which tool did you use to craft these nice diagrams?

• Reply • Share ›

Ole Bjørn Michelsen Mod ➦ Yoni Goldberg • 22 days ago

Sure, I used https://www.websequencediag...

• Reply • Share ›

Bob the Developer • 3 months ago

This is great. Thank you for your post. :)

• Reply • Share ›

Hernán • 6 months ago

I suggest a little change.

Instead of:

```
hello.on('auth.login', function (auth) {
...
});
```

It's better this code:

```
hello(social).login().then((res) => {
...
}, err => {
...
})
```

Because hello.on is executed a lot of times. Instead, If you click a button (In react por example) is better something like this.

<button onclick="{" (e)=""> this.handleSocial(e, 'facebook')}>

But is only a sugerence for React and Angular People

• Reply • Share ›

Hernán • 6 months ago

How can i obtain the profile from the server (with validateWithProvider)?. Because i don't want to send the profile (first_name, last_name, etc) from the client to the server (Risk of manipulation)

• Reply • Share ›

Ole Bjørn Michelsen Mod ➚ Hernán • 6 months ago

We are just calling the /me endpoint on the Facebook Graph API, so you can ask for any field. Try and play with it here: https://developers.facebook...

^ v • Reply • Share ›

Hernán ➚ Ole Bjørn Michelsen • 6 months ago

Thanks. That works very well with Facebook. ¿What about Google?

^ v • Reply • Share ›

Ole Bjørn Michelsen Mod ➚ Hernán • 6 months ago

You will have to look at the docs for their endpoints.

^ v • Reply • Share ›

Hernán ➚ Ole Bjørn Michelsen • 6 months ago

passport-facebook-token and passport-google-plus-token did the trick. Awesome packages!!

Thank you and congratulations for your post. is really very useful.

^ v • Reply • Share ›

helioalves ➚ Hernán • 4 months ago

Hi **@Hernán** ,

I was wondering if you could share your code on how you've implemented authentication with passport-facebook-token. I've been battling with it for a moment now and I can't make it to work. Any help would be appreciated.

Thanks

^ v • Reply • Share ›

Hernán ➚ helioalves • 4 months ago

Hi **@helioalves** . Sure!

First. You need to get the token in the client and then you need to send it to your backend.

if you use an api rest, you can send that token in two ways. In the header or in the body request.

Example with an Api Rest.

```
app.post('/auth/facebook/token',
passport.authenticate('facebook-token'),
function (req, res) {
// do something with req.user
res.send(req.user? 200 : 401);
}
);
```

So, with Postman or something send to your server (example:

localhost:3000/auth/facebook/token) an post request and in the body put: access_token and in the value put the social token obtained in the client.

access_token: YourSocialToken

and don't forget configure passport strategy in another file and them import to your route file (or main file).

Reply • Share ›

helioalves ➚ Hernán • 4 months ago

@Herma,

Thank you very much for the quick reply.

How do I get the token in the client? Are we talking about the accessToken in the Facebook Strategy?
I'm very confused about how to get the token in the client.

This is my Facebook Strategy, I've removed the create user part:

```
passport.use(new FacebookStrategy({
clientID: facebookClientID,
clientSecret: facebookClientSecret
},
async (accessToken, refreshToken, profile, done) => {
console.log('accessToken', accessToken);
}));
```

The token you are talking about here, is it accessToken? If so, how do I pass it to the client?

Reply • Share ›

Hernán ➚ helioalves • 4 months ago

Okey...I think i know about your confusion.

Are you using a Client Framework or library for the frontend? (Angular, React, vue)

Reply • Share ›

helioalves ➚ Hernán • 4 months ago

Hi **@Hernán**,

I'm sorry to come back to you again, but I was just wondering if you know how I could get the initial token that will be passed from the client to the server API. Please not that I'm using React as the client.

Thank you again for your time, really appreciated.

3 Reply • Share ›

helioalves ➚ Hernán • 4 months ago

I was doing some tests with Postman but I'll be using react.

Reply • Share ›

Pavel Strashnov • 8 months ago

Thank you so much Ole!
Was looking for the solution for a while and found everything so clearly explained here.

Reply • Share ›

t3__rrY • 9 months ago

Great article: precisely what I was looking for thanks

Reply • Share ›

Cameron Strandberg • 10 months ago

What about with twitter? Is their approach different?

Reply • Share ›

Ole Bjørn Michelsen Mod ➚ Cameron Strandberg • 10 months ago

No, you can also do OAuth with them.

Reply • Share ›

Mohankumar Anna • 10 months ago

Thanks for sharing this article. I find it useful to implement social logins with already existing jwt authentication

Reply • Share ›

Lorent • a year ago

Thank you for this article. I was looking for a similar solution to replace Auth0 in my projects.
I have started creating a Docker container for the server part (mostly based on your example). Do you allow me to publish it on Docker Hub?

Reply • Share ›

Ole Bjørn Michelsen Mod ➚ Lorent • a year ago

Sure, that sounds cool.

Reply • Share ›

Lorent ➚ Ole Bjørn Michelsen • a year ago

Thanks! Here is a first draft. I will test it more seriously when using it in real use cases.
https://hub.docker.com/r/lo...

Reply • Share ›

Phom Phom • 2 years ago

Could explain me what's facebook secret key for? It seems that it's not used anywhere since access token can be validated with https://graph.facebook.com/me.

Reply • Share ›

Ole Bjørn Michelsen Mod ➚ Phom Phom • 2 years ago

Auth secrets are used for refreshing and authenticating from server-side code. It is not needed (and shouldn't be used) in client-side apps.

Reply • Share ›

Phom Phom ➔ Ole Bjørn Michelsen • 2 years ago

Sure but if i can check if token is valid just by querying graph.facebook.com/me on server side, whats the point of having secret key?

Reply • Share ›

Ole Bjørn Michelsen Mod ➔ Phom Phom • 2 years ago

You don't need it in this use case, or for just doing validation. The secret is used for applications where you might need to do actions on behalf of the user, or update refresh tokens etc.

1 Reply • Share ›

CristianCortez • 2 years ago

Fantastic article! I'm doing exactly the same but using passport-facebook-token and jwt-simple to help me out a little bit!

Reply • Share ›

Ole Bjørn Michelsen Mod ➔ CristianCortez • 2 years ago
