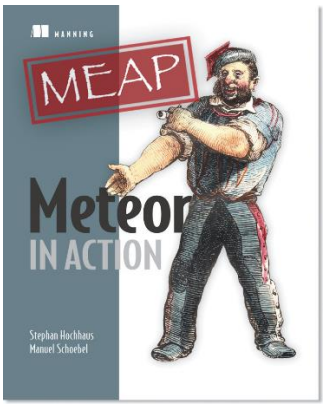
Meteor in Action: Authenticating users with 0Auth

By Stephan Hochhaus and Manuel Schoebel



Meteor ships with multiple authentication providers. All of them are based on oAuth, a complex way to pass authentication data from one site to another. In this article, excerpted from Meteor in Action, we show you how oAuth works.

Oftentimes usernames and passwords are not the only option you want to give your users to log into an application. Being able to use an existing account to log into a site lowers the barrier of signing up by not having to type in a single bit of information. Additionally it simplifies using an application by not having to remember additional usernames or passwords.

Meteor ships with multiple authentication providers that allow users to use a social network instead of a local username. These networks include:

- Facebook
- Github
- Google
- Meetup
- Meteor Developer Account
- Twitter
- Weibo

All of the above are based on oAuth, a complex way to pass authentication data from one site to another. There are also many community packages available that enable other authentication providers such as LinkedIn or Dropbox. As the fundamentals of working with Auth providers are the same for each provider, we will not need to discuss each.

An introduction to oAuth

The Open Authentication – or short: *oAuth* – mechanism has become very popular among web applications since its beginnings in 2007. The main idea behind it is to use a service provider, e.g., Facebook, to authenticate a user and allow a third-party application to access specific information from the authenticated user (access authorization). This could be simply the username or more sensitive information such as friends or a permission to post to the user's wall.

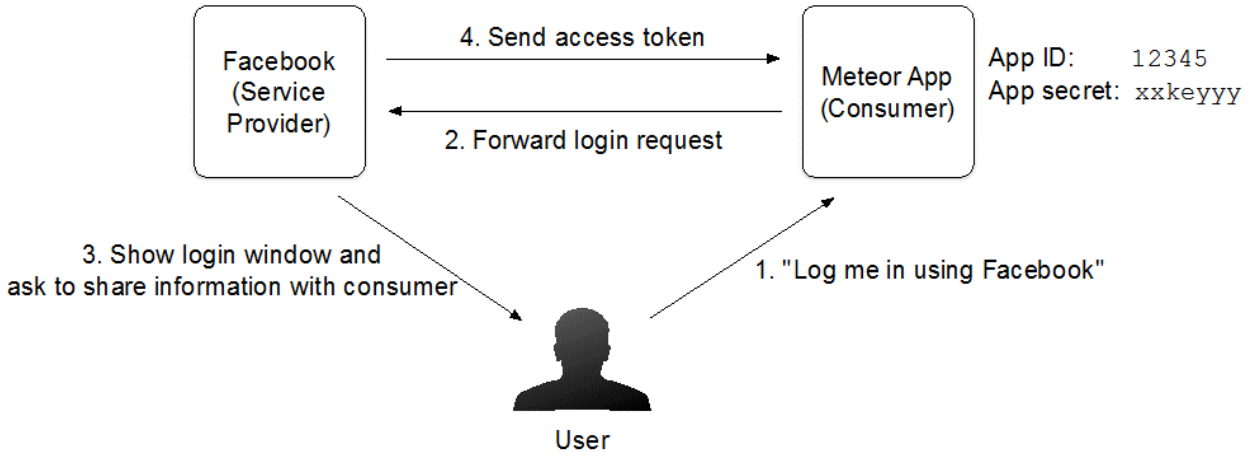


Figure 1: oAuth flow using Facebook as the Service Provider

As Figure 1 shows, there are three main ingredients to every oAuth scenario:

1. A Service Provider – e.g. Facebook or Twitter
2. A Consumer – e.g. your Meteor application
3. The user – e.g. an existing Facebook user wanting to log into your Meteor application

Many sites on the web can act as a service provider for oAuth. We are going to use Facebook as an example to illustrate the process. Our Meteor application must be connected to Facebook. This is done by creating a new application on Facebook's developer site. To verify our application is not a malicious script, it will be able to identify itself using the corresponding *application ID* (App ID) and a *secret key (App Secret)*. These are basically the username and password for our Meteor server process. Once both are connected we can allow users to sign in with their Facebook account.

Instead of entering any credentials in our Meteor app, users can now click a button to log them in via Facebook. Assuming they are already logged in on Facebook they will now see a dialog asking them whether they want to share information with the Meteor application.

Behind the scenes the Meteor application forwarded the login request to Facebook as the service provider. If the user agrees to share their login information with the Meteor application, then Facebook generates an access token. This token lets the Meteor app know that the user has been authenticated properly and it grants the permissions provided by the user. In the simplest case Meteor may only have read access to the user's e-mail address. Depending on the configuration settings we could also request more permissions such as posting to the user's wall.

Not all oAuth providers support the same set of permissions, so they must all be individually configured. The advantage of using oAuth is that the consumer application can talk directly with the service provider to exchange data, if permissions exist. That way all Facebook friends, recent Tweets or number of private repos on Github can easily be accessed and added to the user's profile.

Integrating Facebook authentication

To integrate oAuth authentication via Facebook in a Meteor app we need to perform the following steps:

1. Add the `accounts-facebook` package
2. Create a new Facebook application
3. Configure the Facebook integration

ADDING ACCOUNTS-FACEBOOK TO AN APPLICATION

The first step is to add Facebook as the authentication provider for our application. If the application already support username/password authentication as in section 6.1, it is sufficient to add a single package

```
$ meteor add accounts-facebook
```

This package will not add any templates to the application. Therefore, if `accountsfacebook` is the only package available in the project you would need to manually call all functionality within your templates. Or you can add the

`accounts-ui` package, which not only provides a login box for use with password authentication, but also for many oAuth services.

All oAuth packages require configuration. Just like users, this configuration is stored inside a MongoDB collection. A collection named `meteor_accounts_loginServiceConfiguration` will be created as soon as the service is configured. Additionally, pending credentials will be stored temporarily as well, which is done in a dedicated collection. This collection is created at server startup already and is called `meteor_oauth_pendingCredentials`.

There is no need to manually access either of these two collections. Meteor will use these internally only and there is no benefit in querying data from them directly.

CREATING A FACEBOOK APPLICATION

If Meteor cannot find a configuration for integration with Facebook, the UI will show a red button labeled "Configure Facebook Login" instead of the regular log in button. Clicking on it will bring up a short configuration guide as well as two fields to provide the application ID and secret.

You will need to register yourself as a Facebook developer, which is free of charge but requires you to have a Facebook account. A new Facebook application can be created at https://developers.facebook.com. Under the Apps tab you can add a new application of type Web/WWW. Next you need to assign an application ID. This can be any name that helps you and your users to identify the application. As your users may eventually see the application name it is good practice to use the site name or something that closely describes your application. The actual category for your application and whether it is a test version of another Facebook application do not have any influence on the functionality and can be set to the values that best describe your project.

The site URL for a Facebook application that is used with a local development environment should typically be set to http://localhost:3000. The correct URL setting can be taken from the configuration dialog shown by Meteor.

Once you have made these settings, Facebook requires a contact e-mail to be set up for the application before you can activate the application. Navigate to the application dashboard on the Facebook Developer site and enter a contact e-mail in the settings section (see Figure 2). Finally you need to activate the application in the Status & Review tab.

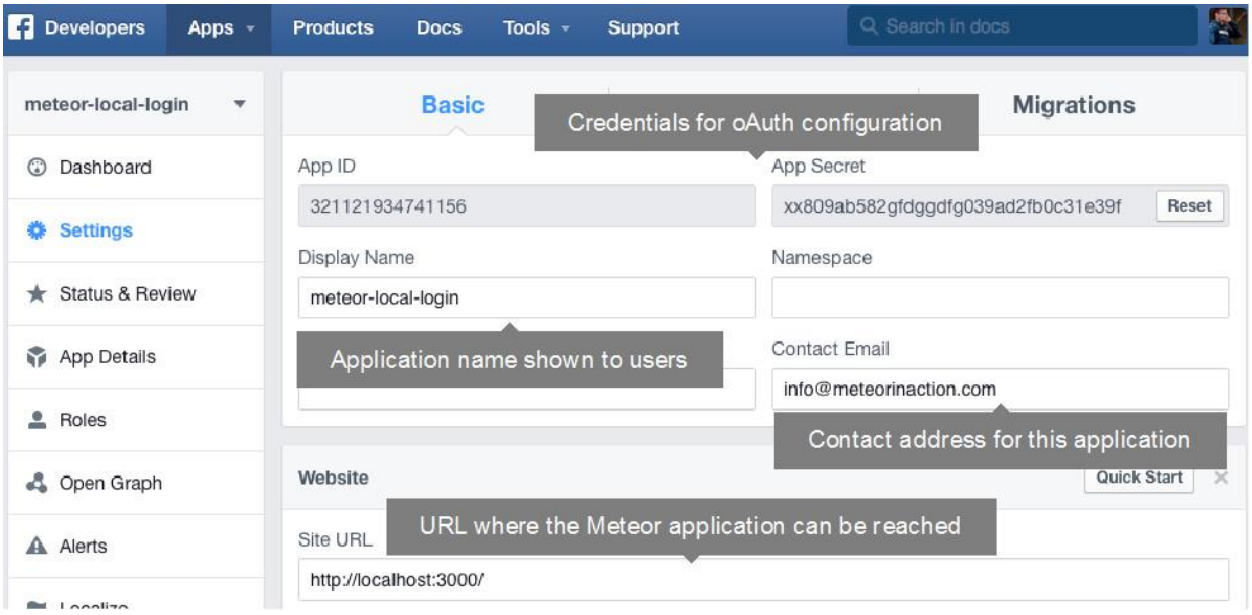


Figure 2: Settings for a Facebook application to integrate with Meteor

An activated Facebook application can be used to authenticate users. The last step to implement logging in via Facebook is to configure the Meteor application.

CONFIGURING

Open the Meteor application in the browser and click on the Facebook button to bring up the configuration dialog as shown in Figure 3.

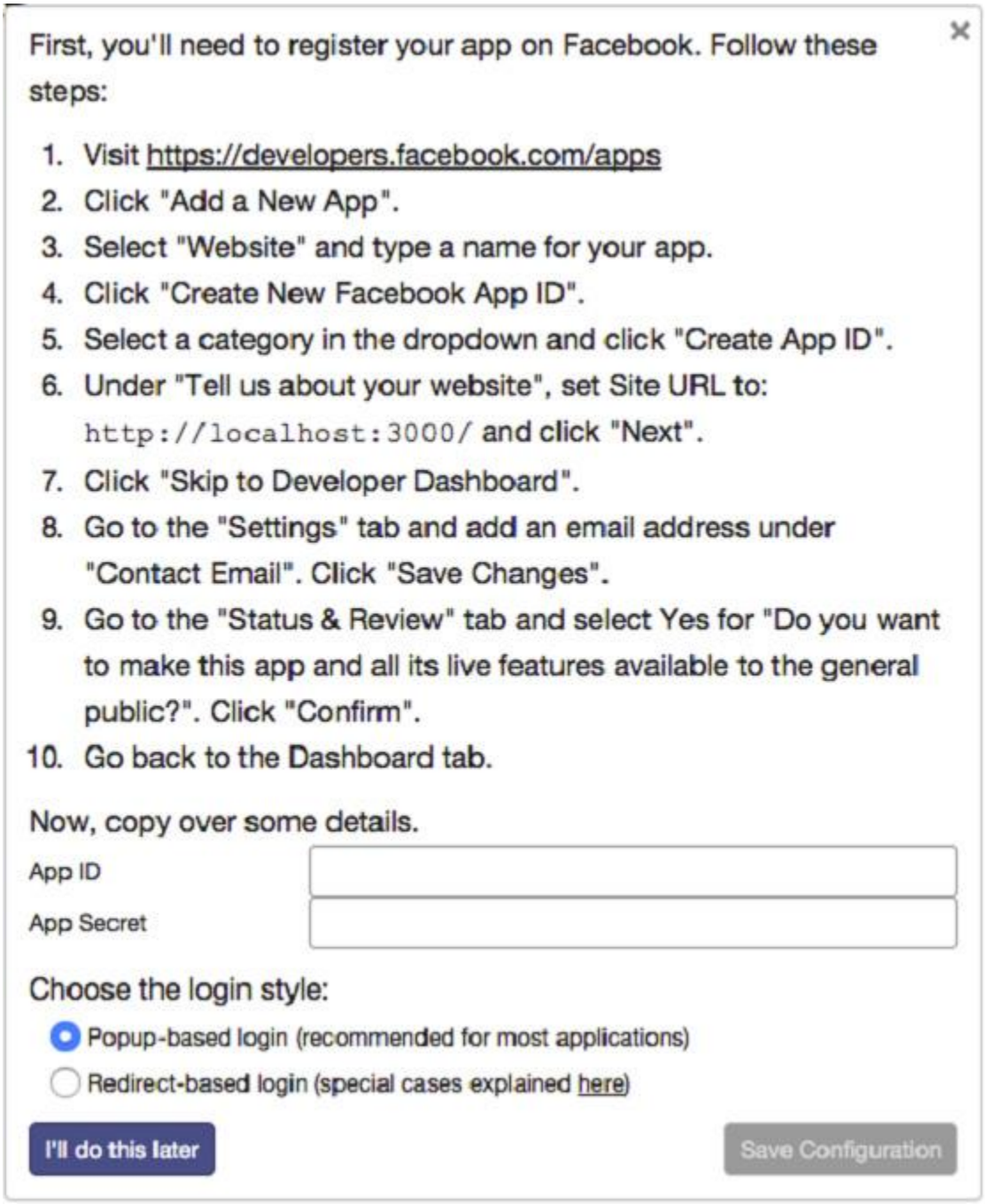


Figure 3: Configuration dialog for Facebook integration

Besides the basic instructions on how to create a Facebook application the configuration dialog lets you add the application credentials (App ID and App Secret) as well as a login style. The default is to use a *popup-based* dialog, which means the application window will remain when the user logs into Facebook and a new window with the Facebook dialog is opened. In contrast, the *redirect-based* login style will leave the application, redirect the current browser window to Facebook, and reload the entire application once the authentication was successful. Unless you plan on running your application inside a Cordova container on mobile devices[13] it is preferable to use the popup-based login.

Save the configuration and users can start logging in via Facebook. Should you have misconfigured the application you can either manually delete the configuration information from the

`meteor_accounts_loginServiceConfiguration` collection inside the MongoDB or simply issue `meteor reset` to empty all collections.

REMEMBER All oAuth configuration is stored inside the application database. Whenever you issue the `meteor reset` command to empty the database it will also remove all oAuth configuration data from the database.