accounts-2fa

This package allows you to provide a way for your users to enable 2FA on their accounts, using an authenticator app such as Google Authenticator, or 1Password. When the user is logged in on your app, they will be able to generate a new QR code and read this code on the app they prefer. After that, they'll start receiving their codes. Then, they can finish enabling 2FA on your app, and every time they try to log in to your app, you can redirect them to a place where they can provide a code they received from the authenticator.

To provide codes that are exactly compatible with all other Authenticator apps and services that implements TOTP, this package uses node-2fa which works on top of notp, **that** implements TOTP (RFC 6238) (the Authenticator standard), which is based on HOTP (RFC 4226).

> This package is meant to be used with `accounts-password` or `accounts-passwordless`, so if you don't have either of those in your project, you'll need to add one of them. In the future, we want to enable the use of this package with other login methods, our oauth methods (Google, GitHub, etc…).

Activating 2FA

The first step, in order to enable 2FA, is to generate a QR code so that the user can scan it in an authenticator app and start receiving codes.

{% apibox "Accounts.generate2faActivationQrCode" "module":"accounts-base" %}

Receives an `appName` which is the name of your app that will show up when the user scans the QR code. Also, a callback called, on success, with a QR code in SVG format, a QR secret, and the URI that can be used to activate the 2FA in an authenticator app, or a single `Error` argument on failure.

On success, this function will also add an object to the logged user's services object containing the QR secret:

```
services: {
  ...
  twoFactorAuthentication: {
    secret: "***"
  }
}
```

Here it's an example on how to call this function:

```
import { Buffer } from "buffer";
import { Accounts } from 'meteor/accounts-base';

--

const [qrCode, setQrCode] = useState(null);

--

<button
  onClick={() => {
    Accounts.generate2faActivationQrCode("My app name", (err, result) => {
      if (err) {console.error("...", err);return;}
      const { svg, secret, uri } = result;
      /*
        the svg can be converted to base64, then be used like:
         <img
            width="200"
            src={`data:image/svg+xml;base64,${qrCode}`}
         />
      */
      setQrCode(Buffer.from(svg).toString('base64'));
    })
  }}
>
  Generate a new code
</button>
```

This method can fail throwing the following error: * "The 2FA is activated. You need to disable the 2FA first before trying to generate a new activation code [2fa-activated]" if trying to generate an activation when the user already have 2FA enabled.

At this point, the 2FA won't be activated just yet. Now that the user has access to the codes generated by their authenticator app, you can call the function `Accounts.enableUser2fa`:

{% apibox "Accounts.enableUser2fa" "module":"accounts-base" %}

It should be called with a code that the users will receive from the authenticator app once they read the QR code. The callback is called with a single `Error` argument on failure. If the code provided is correct, a `type` will be added to the user's `twoFactorAuthentication` object and now 2FA is considered enabled:

```
services: {
  ...
  twoFactorAuthentication: {
    type: "otp",
    secret: "***",
  }
}
```

To verify whether or not a user has 2FA enabled, you can call the function `Accounts.has2faEnabled`:

{% apibox “Accounts.has2faEnabled” “module”:“accounts-base” %}

This function must be called when the user is logged in.

Disabling 2FA

To disable 2FA for a user use this method:

{% apibox “Accounts.disableUser2fa” “module”:“accounts-base” %}

To call this function the user must be already logged in.

Log in with 2FA

Now that you have a way to allow your users to enable 2FA on their accounts, you can create a login flow based on that.

As said at the beginning of this guide, this package is currently working with two other packages: `accounts-password` and `accounts-passwordless`. Below there is an explanation on how to use this package with them.

Working with accounts-password

When calling the function `Meteor.loginWithPassword`, if the 2FA is enabled for the user, an error will be returned to the callback, so you can redirect the user to a place where they can provide a code.

As an example:

```
<button
  onClick={() => {
    Meteor.loginWithPassword(username, password, error => {
      if (error) {
        if (error.error === 'no-2fa-code') {
            // send user to a page or show a component
            // where they can provide a 2FA code
            setShouldAskCode(true);
            return;
        }
        console.error("Error trying to log in (user without 2fa)", error);
      }
    });
  }
}>
  Login
</button>
```

If the 2FA is not enabled, the user will be logged in normally.

The function you will need to call now to allow the user to login is `Meteor.loginWithPasswordAnd2faCode`:

{% apibox “Meteor.loginWithPasswordAnd2faCode” %}

Now you will be able to receive a code from the user and this function will verify if the code is valid. If it is, the user will be logged in.

So the call of this function should look something like this:

```
<button onClick={() => {
  Meteor.loginWithPasswordAnd2faCode(username, password, code, error => {
    if (error) {
      console.error("Error trying to log in (user with 2fa)", error);
    }
  })}
}>
  Validate and log in
</button>
```

This method can fail throwing one of the following errors: * “2FA code must be informed [no-2fa-code]” if a 2FA code was not provided. * “Invalid 2FA code [invalid-2fa-code]” if the provided 2FA code is invalid.

Working with accounts-passwordless

Following the same logic from the previous package, if the 2FA is enabled, an error will be returned to the callback of the function `Meteor.passwordlessLoginWithToken`, then you can redirect the user to a place where they can provide a code.

Here is an example:

```
<button
  onClick={() => {
    // logging in just with token
    Meteor.passwordlessLoginWithToken(
      email,
      token,
      error => {
        if (error) {
          if (error.error === 'no-2fa-code') {
            // send user to a page or show a component
            // where they can provide a 2FA code
            setShouldAskCode(true);
            return;
          }
          console.error('Error verifying token', error);
        }
      }
    );
  }}
>
  Validate token
</button>;
```

Now you can call the function `Meteor.passwordlessLoginWithTokenAnd2faCode` that will allow you to provide a selector, token, and 2FA code:

{% apibox “Meteor.passwordlessLoginWithTokenAnd2faCode” %}

This method can fail throwing one of the following errors: * “2FA code must be informed [no-2fa-code]” if a 2FA code was not provided. * “Invalid 2FA code [invalid-2fa-code]” if the provided 2FA code is invalid.

How to integrate an Authentication Package with accounts-2fa

To integrate this package with any other existing Login method, it’s necessary following two steps:

1 - For the client, create a new method from your current login method. So for example, from the method `Meteor.loginWithPassword` we created a new one called `Meteor.loginWithPasswordAnd2faCode`, and the only difference between them is that the latest one receives one additional parameter, the 2FA code, but we call the same function on the server side.

2 - For the server, inside the function that will log the user in, you verify if the function `Accounts._check2faEnabled` exists, and if yes, you call it providing the user object you want to check if the 2FA is enabled, and if either of these statements are false, you proceed with the login flow. This function exists only when the package `accounts-2fa` is added to the project.

If both statements are true, and the login validation succeeds, you verify if a code was provided: if not, throw an error; if it was provided, verify if the code is valid by calling the function `Accounts._isTokenValid`. if `Accounts._isTokenValid` returns false, throw an error.

Here it’s an example:

```
  const result = validateLogin();
  if (
    !result.error &&
    Accounts._check2faEnabled?.(user)
  ) {
    if (!code) {
```

```
    Accounts._handleError('2FA code must be informed.');
  }
  if (
    !Accounts._isTokenValid(user.services.twoFactorAuthentication.secret, code)
  ) {
    Accounts._handleError('Invalid 2FA code.');
  }
}

return result;
```