The `accounts-password` package contains a full system for password-based authentication. In addition to the basic username and password-based sign-in process, it also supports email-based sign-in including address verification and password recovery emails.

The Meteor server stores passwords using the bcrypt algorithm. This helps protect against embarrassing password leaks if the server's database is compromised.

To add password support to your application, run this command in your terminal:

```
meteor add accounts-password
```

> *In addition to configuring the `email` package's `MAIL_URL`, it is critical that you set proper values (specifically the `from` address) in `Accounts.emailTemplates` to ensure proper delivery of e-mails!*

You can construct your own user interface using the functions below, or use the `accounts-ui` package to include a turn-key user interface for password-based sign-in.

{% apibox "Accounts.createUser" %}

On the client, this function logs in as the newly created user on successful completion. On the server, it returns the newly created user id.

On the client, you must pass `password` and at least one of `username` or `email` — enough information for the user to be able to log in again later. If there are existing users with a username or email only differing in case, `createUser` will fail. The callback's `error.reason` will be `'Username already exists.'` or `'Email already exists.'` In the latter case, the user can then either login or reset their password.

On the server, you do not need to specify `password`, but the user will not be able to log in until it has a password (eg, set with `Accounts.setPassword`). To create an account without a password on the server and still let the user pick their own password, call `createUser` with the `email` option and then call `Accounts.sendEnrollmentEmail`. This will send the user an email with a link to set their initial password.

By default the `profile` option is added directly to the new user document. To override this behavior, use `Accounts.onCreateUser`.

This function is only used for creating users with passwords. The external service login flows do not use this function.

Instead of modifying documents in the `Meteor.users` collection directly, use these convenience functions which correctly check for case insensitive duplicates before updates.

{% apibox "Accounts.createUserVerifyingEmail" %}

{% apibox "Accounts.setUsername" %}

{% apibox "Accounts.addEmail" %}

By default, an email address is added with `{ verified: false }`. Use `Accounts.sendVerificationEmail` to send an email with a link the user can use to verify their email address.

{% apibox "Accounts.removeEmail" %}

{% apibox "Accounts.verifyEmail" %}

This function accepts tokens passed into the callback registered with `Accounts.onEmailVerificationLink`.

{% apibox "Accounts.findUserByUsername" %}

{% apibox "Accounts.findUserByEmail" %}

Use the below functions to initiate password changes or resets from the server or the client.

{% apibox "Accounts.changePassword" %}

{% apibox "Accounts.forgotPassword" %}

This triggers a call to `Accounts.sendResetPasswordEmail` on the server. When the user visits the link in this email, the callback registered with `Accounts.onResetPasswordLink` will be called.

If you are using the `accounts-ui` package, this is handled automatically. Otherwise, it is your responsibility to prompt the user for the new password and call `resetPassword`.

{% apibox "Accounts.resetPassword" %}

This function accepts tokens passed into the callbacks registered with `AccountsClient#onResetPasswordLink` and `Accounts.onEnrollmentLink`.

{% apibox "Accounts.setPassword" %}

{% apibox "Accounts.sendResetPasswordEmail" %}

When the user visits the link in this email, the callback registered with `AccountsClient#onResetPasswordLink` will be called.

To customize the contents of the email, see `Accounts.emailTemplates`.

{% apibox "Accounts.sendEnrollmentEmail" %}

When the user visits the link in this email, the callback registered with `Accounts.onEnrollmentLink` will be called.

To customize the contents of the email, see `Accounts.emailTemplates`.

{% apibox "Accounts.sendVerificationEmail" %}

When the user visits the link in this email, the callback registered with `Accounts.onEmailVerificationLink` will be called.

To customize the contents of the email, see `Accounts.emailTemplates`.

{% apibox "Accounts.onResetPasswordLink" %}

{% apibox "Accounts.onEnrollmentLink" %}

{% apibox "Accounts.onEmailVerificationLink" %}

{% apibox "Accounts.emailTemplates" %}

This is an `Object` with several fields that are used to generate text/html for the emails sent by `sendResetPasswordEmail`, `sendEnrollmentEmail`, and `sendVerificationEmail`.

Set the fields of the object by assigning to them:

- `from` : (**required**) A `String` with an RFC5322 From address. By default, the email is sent from `no-reply@example.com` . **If you want e-mails to send correctly, this should be changed to your own domain as most e-mail providers will reject mail sent from `example.com` .**
- `siteName` : The public name of your application. Defaults to the DNS name of the application (eg: `awesome.meteor.com`).
- `headers` : An `Object` for custom email headers as described in `Email.send` .
- `resetPassword` : An `Object` with the fields:
- `from` : A `Function` used to override the `from` address defined by the `emailTemplates.from` field.
- `subject` : A `Function` that takes a user object and returns a `String` for the subject line of a reset password email.
- `text` : An optional `Function` that takes a user object and a url, and returns the body text for a reset password email.
- `html` : An optional `Function` that takes a user object and a url, and returns the body html for a reset password email.
- `enrollAccount` : Same as `resetPassword` , but for initial password setup for new accounts.
- `verifyEmail` : Same as `resetPassword` , but for verifying the users email address.

Example:

```
Accounts.emailTemplates.siteName = 'AwesomeSite';
Accounts.emailTemplates.from = 'AwesomeSite Admin <accounts@example.com>';

Accounts.emailTemplates.enrollAccount.subject = (user) => {
  return `Welcome to Awesome Town, ${user.profile.name}`;
};

Accounts.emailTemplates.enrollAccount.text = (user, url) => {
  return 'You have been selected to participate in building a better future!'
    + ' To activate your account, simply click the link below:\n\n'
    + url;
};

Accounts.emailTemplates.resetPassword.from = () => {
  // Overrides the value set in `Accounts.emailTemplates.from` when resetting
  // passwords.
  return 'AwesomeSite Password Reset <no-reply@example.com>';
};
Accounts.emailTemplates.verifyEmail = {
   subject() {
      return "Activate your account now!";
   },
   text(user, url) {
      return `Hey ${user}! Verify your e-mail by following this link: ${url}`;
   }
};
```

Enable 2FA for this package

You can add 2FA to your login flow by using the package accounts-2fa. You can find an example showing how this would look like here.