Adding a new 'managed' authentication method to Discourse

- Implementing an authenticator
- ManagedAuthenticator in details
  - Migration to the new system

Continuing from https://meta.discourse.org/t/future-social-authentication-improvements/94691...

We are now in the process of moving all 'associated account' information into a single database table. This will help to significantly reduce duplicated logic, and allow quicker development in the future. For example, migrating our core twitter logic to the new system reduced the number of lines of code from 136 to just 24 :tada:.

This post isn't designed to be a step-by-step instruction manual for adding a new authentication provider, but it will aim to provide an overview, pointing to the relevant source code where necessary.

Implementing an authenticator

Each authenticator **must** implement a subclass of Auth::Authenticator. To use the new shared logic, the authenticator **can** instead extend Auth::ManagedAuthenticator. An example of a bare-bones implementation can be found in the core Facebook authenticator:

https://github.com/discourse/discourse/blob/main/lib/auth/facebook_authenticator.rb

`name` and `register_middleware` must be overridden by implementing classes, along with `enable_setting` — the boolean site setting an admin uses to turn the provider on.

An authenticator **should** also declare `required_settings`: the site settings that must have a value before an authentication can succeed. The base class uses them for `configured?`, and `enabled?` is `enable_setting && configured?`, so a provider with missing credentials is never advertised on the login page and its `/auth/<name>` route stays closed — otherwise clicking the button strands the user on the provider's own error page with no way back. Declaring `required_settings` also lets `AuthProviderCredentialsValidator` refuse to enable the provider in the first place; wire it up with `validator: "AuthProviderCredentialsValidator"` on the enable setting.

```
def enable_setting
  :enable_google_oauth2_logins
end

def required_settings
  %i[google_oauth2_client_id google_oauth2_client_secret]
end
```

An authenticator that overrides `enabled?` directly opts out of both gates.

> :information_source: **Aside:** for multisite compatibility, it is important that any site-specific information is supplied to omniauth in a `setup` lambda, rather than being fixed at the time of definition. See all core authenticators for examples of this.

All logic to link external accounts to Discourse accounts is handled by `Auth::ManagedAuthenticator`. This relies on the omniauth provider returning data in the format defined in their documentation. If any manipulation of this data is required, Authenticators can override the `after_authenticate` method, and manipulate the auth_token as required. For example, the core Twitter authenticator removes all the `extra` information from the token:

https://github.com/discourse/discourse/blob/b46b6e72d1906ca31e29855bda71f3498c8e203f/lib/auth/twitter_authenticator.rb#L10-L14

Data is stored in the `user_associated_accounts` database table. `provider_uid`, `info`, `credentials` and `extra` are all taken directly from the data returned by omniauth.

Provider images are downloaded from `info.image` on authentication and retained in each account's `avatar_upload_id`. They remain separate from the user's uploaded picture and selectable presets. The avatar picker offers cached images from enabled providers under the existing avatar permissions.

`UserAvatar` owns avatar imports, selection, refreshes, and cleanup. Authentication schedules provider retrieval through `UserAvatar.retrieve_for_associated_account`; download jobs and importers use `UserAvatar.import_url_for_user`, passing `associated_account_id` for provider images.

User convenience methods and account/upload lifecycle callbacks delegate avatar changes to `UserAvatar`; permissions remain in Guardian.

`user_avatars.selected_user_associated_account_id` records an explicitly selected provider. Subsequent downloads update the displayed avatar only while that provider remains selected. New accounts initially select their provider when no avatar was assigned; `auth_overrides_avatar` continues to enforce provider selection. Existing avatars retain their appearance and are not assigned a provider automatically. Existing linked accounts populate their provider choices on their next login.

Choosing another avatar clears provider selection. Disconnecting an account preserves its currently displayed image as a local snapshot and preserves any separate uploaded picture. Failed downloads retain the previous image. Queued downloads are discarded if the account was disconnected, moved to another user, or now supplies a different image URL.

Gravatar selection still uses upload-ID matching. A custom or preset picture identical to the cached Gravatar can therefore follow subsequent Gravatar updates.

https://github.com/discourse/discourse/blob/b46b6e72d1906ca31e29855bda71f3498c8e203f/app/models/user_associated_account.rb#L13-L24

Once an `Authenticator` class has been defined, it needs to be registered. This must happen early in the application's lifecycle, and can **not** happen within a plugin's `after_initialize` method. The minimum registration can simply contain a reference to the authenticator. In a plugin, registration can be done using the `auth_provider` function. For example:

```
auth_provider authenticator: OpenIDConnectAuthenticator.new()
```

In core, registration takes place in `discourse.rb`. A full list of possible `AuthProvider` options can be found here. Text content **can** be defined using these options, but it is better to provide localisable strings in `client.en.yml` following the standard keys. For example:

https://github.com/discourse/discourse-openid-connect/blob/88fdf7b5ab624aba7c207e403665e0393334794a/config/locales/client.en.yml#L1-L7

[details=Additional ManagedAuthenticator notes by @fantasticfears]

`ManagedAuthenticator` in details

You might need to work on something special for authentication. And you would like to know more about `ManagedAuthenticator`. Basically, it has several operations, options, and controls how the data will be used.

Discourse manages user information with two controllers. `Users::OmniauthCallbacksController` manages the payload once OAuth2 authentication is done. `after_authenticate` is called here. `can_connect_existing_user?` is also used here. There are some private methods you can read to understand how different data fields work.

```
if authenticator.can_connect_existing_user? && current_user
  @auth_result = authenticator.after_authenticate(auth, existing_account: current_user)
else
  @auth_result = authenticator.after_authenticate(auth)
end
```

`UsersController` has `revoke_account` which uses `can_revoke?` and `revoke`. But for the `revoke` method to work remotely, you need to build your own implementation.

`UserAuthenticator` is a service class helping authenticate (verifying email confirmation or OAuth2 path) users. `after_create_account` is called here.

The core logic remains at `after_authenticate` with `Auth::Result` data class. We follow data structure here. `extra_data` will be passed to `after_create_account` for creating related records.

```
result.extra_data = {
  provider: auth_token[:provider],
  uid: auth_token[:uid],
  info: auth_token[:info],
  extra: auth_token[:extra],
  credentials: auth_token[:credentials]
}
```

It will try to match and connects to an existing account.

You might wonder why automatic account creation is possible but there is no `User.create`. This is done in `UsersController#create`.

```
authentication = UserAuthenticator.new(user, session)
```

The user is a fresh instance will be populated by session data which is prepared by the auth provider. Trust me, it's just magic. [/details]

Migration to the new system

To provide a seamless switch to the new system, data should be migrated from the old storage location. For core authentication providers, this may be dedicated tables. For plugins, this may be

`plugin_store_rows`, or `oauth2_user_infos`. The minimum data required in a `user_associated_accounts` row is `provider_name`, `provider_uid` and `user_id`. For an example migration see:

https://github.com/discourse/discourse/blob/main/db/migrate/20181207141900_migrate_twitter_user_info.rb

Once the `ManagedAuthenticator` system has been released to the stable branch with v2.2.0, we will begin migrating official authentication **plugins**. At this point, a `plugin_store_row` migration example will be added here.

□□□ #2
□ □□□ □□□ 2026-10-31 13:28:14 UTC
□ □□□ □□□ 2026-10-31 13:28:15 UTC