100 XP

Explore authentication and authorization in App Service

3 minutes

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, API, and mobile back end, and also Azure Functions.

Why use the built-in authentication?

You're not required to use App Service for authentication and authorization. Many web frameworks are bundled with security features, and you can use them if you like. If you need more flexibility than App Service provides, you can also write your own utilities.

The built-in authentication feature for App Service and Azure Functions can save you time and effort by providing out-of-the-box authentication with federated identity providers, allowing you to focus on the rest of your application.

- Azure App Service allows you to integrate a variety of auth capabilities into your web app or API without implementing them yourself.
- It's built directly into the platform and doesn't require any particular language, SDK, security expertise, or even any code to utilize.
- You can integrate with multiple login providers. For example, Azure AD, Facebook, Google, Twitter.

Identity providers

App Service uses federated identity, in which a third-party identity provider manages the user identities and authentication flow for you. The following identity providers are available by default:

Provider	Sign-in endpoint	How-To guidance
Microsoft Identity Platform	`/.auth/login/aad`	App Service Microsoft Identity Platform login
Facebook	`/.auth/login/facebook`	App Service Facebook login
Google	`/.auth/login/google`	App Service Google login
Twitter	`/.auth/login/twitter`	App Service Twitter login
Any OpenID Connect provider	`/.auth/login/<providerName>`	App Service OpenID Connect login

When you enable authentication and authorization with one of these providers, its sign-in endpoint is available for user authentication and for validation of authentication tokens from the provider. You can provide your users with any number of these sign-in options.

How it works

The authentication and authorization module runs in the same sandbox as your application code. When it's enabled, every incoming HTTP request passes through it before being handled by your application code. This module handles several things for your app:

- Authenticates users with the specified provider
- Validates, stores, and refreshes tokens
- Manages the authenticated session
- Injects identity information into request headers

The module runs separately from your application code and is configured using app settings. No SDKs, specific languages, or changes to your application code are required.

> **Note**
>
> In Linux and containers the authentication and authorization module runs in a separate container, isolated from your application code. Because it does not run in-process, no direct integration with specific language frameworks is possible.

Authentication flow

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK.

- Without provider SDK: The application delegates federated sign-in to App Service. This is typically the case with browser apps, which can present the provider's login page to the user. The server code manages the sign-in process, so it is also called *server-directed flow* or *server flow*.
- With provider SDK: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with browser-less apps, which can't present the provider's sign-in page to the user. The application code manages the sign-in process, so it is also called *client-directed flow* or *client flow*. This applies to REST APIs, Azure Functions, JavaScript browser clients, and native mobile apps that sign users in using the provider's SDK.

The table below shows the steps of the authentication flow.

Step	Without provider SDK	With provider SDK
Sign user in	Redirects client to `/.auth/login/<provider>`.	Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation.
Post-authentication	Provider redirects client to `/.auth/login/<provider>/callback`.	Client code posts token from provider to `/.auth/login/<provider>` for validation.
Establish authenticated session	App Service adds authenticated cookie to response.	App Service returns its own authentication token to client code.
Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in `X-ZUMO-AUTH` header (automatically handled by Mobile Apps client SDKs).

For client browsers, App Service can automatically direct all unauthenticated users to `/.auth/login/<provider>`. You can also present users with one or more `/.auth/login/<provider>`

links to sign in to your app using their provider of choice.

Authorization behavior

In the Azure portal, you can configure App Service with a number of behaviors when an incoming request is not authenticated.

- **Allow unauthenticated requests:** This option defers authorization of unauthenticated traffic to your application code. For authenticated requests, App Service also passes along authentication information in the HTTP headers.This option provides more flexibility in handling anonymous requests. It lets you present multiple sign-in providers to your users.
- **Require authentication:** This option will reject any unauthenticated traffic to your application. This rejection can be a redirect action to one of the configured identity providers. In these cases, a browser client is redirected to `/.auth/login/<provider>` for the provider you choose. If the anonymous request comes from a native mobile app, the returned response is an `HTTP 401 Unauthorized`. You can also configure the rejection to be an `HTTP 401 Unauthorized` or `HTTP 403 Forbidden` for all requests.

> ⊗ **Caution**
>
> Restricting access in this way applies to all calls to your app, which may not be desirable for apps wanting a publicly available home page, as in many single-page applications.

Next unit: Discover App Service networking features

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆