Article

Authenticating a User Through a Web Service

Use a web authentication session to authenticate a user in your app.

Overview

Some websites provide, as a service, a secure mechanism for authenticating users. When the user navigates to the site's authentication URL, the site presents the user with a form to collect credentials. After validating the credentials, the site redirects the user's browser, typically using a custom scheme, to a URL that indicates the outcome of the authentication attempt.

Create a Web Authentication Session

You can make use of a web authentication service in your app by initializing an `ASWebAuthenticationSession` instance with a URL that points to the authentication webpage. The page can be one that you maintain, or one operated by a third party. During initialization, indicate the callback scheme that the page uses to return the authentication outcome:

```
// Use the URL and callback scheme specified by the authorization provider.
guard let authURL = URL(string: "https://example.com/auth") else { return }
let scheme = "exampleauth"

// Initialize the session.
let session = ASWebAuthenticationSession(url: authURL, callbackURLScheme: scheme)
{ callbackURL, error in
    // Handle the callback.
}
```

Use the initializer's trailing closure to tell the session how to handle the callback after authentication completes, as described below in Handle the Callback.

In macOS, or if you have a deployment target of iOS 13 or later, the session keeps a strong reference to itself until the authentication process completes to prevent the system from deallocating the closure. For earlier iOS deployment targets, your app needs to keep a strong reference to the session until authentication completes.

Provide a Presentation Context

Your app indicates the window that should act as a presentation anchor for the session by adopting the `ASWebAuthenticationPresentationContextProviding` protocol. From the `presentationAnchor(for:)` method, which is the protocol's one required method, return the window that should act as the anchor:

```
extension ViewController: ASWebAuthenticationPresentationContextProviding {
    func presentationAnchor(for session: ASWebAuthenticationSession) -> ASPresentati
        return view.window!
    }
}
```

After creating the session, set an appropriate context provider instance as the session's `presentationContextProvider` delegate:

```
session.presentationContextProvider = viewController
```

Optionally Request Ephemeral Browsing

You can configure the session to request ephemeral browsing by setting the session's `prefersEphemeralWebBrowserSession` property to `true`:

```
session.prefersEphemeralWebBrowserSession = true
```

This setting asks the browser to avoid using any existing browsing data, like cookies, during the authentication process. It also asks that the browser avoid retaining any data collected during the authentication attempt beyond the lifetime of the attempt, or sharing it with any other session. An ephemeral session may improve security but prevents reusing the result of a previously successful authentication, potentially forcing the user to reenter credentials. As a result, it's typically best to let the user choose whether to request ephemeral browsing.

Safari always respects the request. In macOS, the user can choose a different default browser that might or might not respect the request.

> **Note**
>
> When not using an ephemeral session, all cookies except session cookies are available to the browser.

Start the Authentication Flow

After configuring the session, call its `start()` method:

```
session.start()
```

In iOS, the session loads the authentication web page that you indicated during initialization in an embedded browser view. In macOS, the session sends the page load request to the user's default browser if it handles authentication sessions, or to Safari otherwise. In any case, the browser presents the user with the authentication page, which is typically a form for entering a username and password.

You can cancel the authentication attempt from your app before the user finishes by calling the session's `cancel()` method:

```
session.cancel()
```

When you cancel, the session automatically dismisses the corresponding browser view.

Handle the Callback

After the user authenticates, the authentication provider redirects the browser to a URL that uses the callback scheme. The browser detects the redirect, dismisses itself, and passes the complete URL to your app by calling the closure you specified during initialization.

When you receive this callback, first check for errors. For example, you receive the `canceledLogin` error if the user aborts the flow by dismissing the browser window. If the error is `nil`, inspect the callback URL to determine the outcome of the authentication:

```
guard error == nil, let callbackURL = callbackURL else { return }

// The callback URL format depends on the provider. For this example:
```

```
//   exampleauth://auth?token=1234
let queryItems = URLComponents(string: callbackURL.absoluteString)?.queryItems
let token = queryItems?.filter({ $0.name == "token" }).first?.value
```

The above example looks for a token stored as a query parameter. The specific parsing that you have to do depends on how the authentication provider structures the callback URL.

See Also

Web authentication sessions

Securing Logins with iCloud Keychain Verification Codes

Use time-based codes generated on-device for a secure authentication experience.

`class ASWebAuthenticationSession`

A session that an app uses to authenticate a user through a web service.

`struct WebAuthenticationSession`

A SwiftUI environment value that views use to authenticate someone using a web service.

Supporting Single Sign-On in a Web Browser App

Extend your web browser app to handle web authentication requests from other apps.

`class ASWebAuthenticationSessionWebBrowserSessionManager`

A session manager that mediates sharing data between an app and a web browser.

`ASWebAuthenticationSessionWebBrowserSupportCapabilities`

A collection of keys that a browser app uses to declare its ability to handle authentication requests from other apps.