Sample Code

Implementing User Authentication with Sign in with Apple

Provide a way for users of your app to set up an account and start using your services.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 11.3+

Overview

This sample app, Juice, uses the AuthenticationServices framework to provide users an interface to set up accounts and sign in with their Apple ID. The app presents a form in which the user can create and set up an account for the app, then authenticates the user's Apple ID with Sign in with Apple, and displays the user's account data.

For more information about implementing Sign in with Apple on iOS 12 and earlier, see Incorporating Sign in with Apple into Other Platforms.

Configure the Sample Code Project

To configure the sample code project, perform the following steps in Xcode:

1. On the Signing & Capabilities pane, set the bundle ID to a unique identifier (you must change the bundle ID to proceed).
2. Add your Apple ID account and assign the target to a team so Xcode can enable the Sign in with Apple capability with your provisioning profile.
3. Choose a run destination from the scheme pop-up menu that you're signed into with an Apple ID and that uses Two-Factor Authentication.

4. If necessary, click Register Device in the Signing & Capabilities pane to create the provisioning profile.

5. In the toolbar, click Run, or choose Product > Run (⌘R).

Add a Sign in with Apple Button

In the sample app, `LoginViewController` displays a login form and a Sign in with Apple button (`ASAuthorizationAppleIDButton`) in its view hierarchy. The view controller also adds itself as the button's target, and passes an action to be invoked when the button receives a touch-up event.

```
func setupProviderLoginView() {
    let authorizationButton = ASAuthorizationAppleIDButton()
    authorizationButton.addTarget(self, action: #selector(handleAuthorizationAppleID
    self.loginProviderStackView.addArrangedSubview(authorizationButton)
}
```

For more information about which Sign in with Apple buttons are available on different Apple platforms, see Displaying Sign in with Apple buttons in your app.

> **Important**
>
> When adding the Sign in with Apple button to your storyboard, you must also set the control's class value to `ASAuthorizationAppleIDButton` in Xcode's Identity Inspector.

Request Authorization with Apple ID

When the user taps the Sign in with Apple button, the view controller invokes the `handleAuthorizationAppleIDButtonPress()` function, which starts the authentication flow by performing an authorization request for the users's full name and email address. The system then checks whether the user is signed in with their Apple ID on the device. If the user is not signed in at the system-level, the app presents an alert directing the user to sign in with their Apple ID in Settings.

```
@objc
func handleAuthorizationAppleIDButtonPress() {
    let appleIDProvider = ASAuthorizationAppleIDProvider()
    let request = appleIDProvider.createRequest()
    request.requestedScopes = [.fullName, .email]
```

```
    let authorizationController = ASAuthorizationController(authorizationRequests: [
    authorizationController.delegate = self
    authorizationController.presentationContextProvider = self
    authorizationController.performRequests()
}
```

> **Important**
>
> The user must enable Two-Factor Authentication to use Sign in with Apple so that access to the account is secure.

The authorization controller calls the `ASAuthorizationControllerPresentationContextProviding.presentationAnchor(for:)` function to get the window from the app where it presents the Sign in with Apple content to the user in a modal sheet.

```
func presentationAnchor(for controller: ASAuthorizationController) -> ASPresentatior
    return self.view.window!
}
```

If the user is signed in at the system-level with their Apple ID, the sheet appears describing the Sign in with Apple feature, followed by another sheet allowing the user to edit the information in their account. The user can edit their first and last name, choose another email address as their contact information, and hide their email address from the app. If the user chooses to hide their email address from the app, Apple generates a proxy email address to forward email to the user's private email address. Lastly, the user enters the password for the Apple ID, then clicks Continue to create the account.

Handle User Credentials

If the authentication succeeds, the authorization controller invokes the `ASAuthorizationControllerDelegate.authorizationController(controller:didCompleteWithAuthorization:)` delegate function, which the app uses to store the user's data in the keychain.

```
func authorizationController(controller: ASAuthorizationController, didCompleteWithA
    switch authorization.credential {
    case let appleIDCredential as ASAuthorizationAppleIDCredential:

        // Create an account in your system.
```

```
        let userIdentifier = appleIDCredential.user
        let fullName = appleIDCredential.fullName
        let email = appleIDCredential.email

        // For the purpose of this demo app, store the `userIdentifier` in the keych
        self.saveUserInKeychain(userIdentifier)

        // For the purpose of this demo app, show the Apple ID credential informatic
        self.showResultViewController(userIdentifier: userIdentifier, fullName: full

    case let passwordCredential as ASPasswordCredential:

        // Sign in using an existing iCloud Keychain credential.
        let username = passwordCredential.user
        let password = passwordCredential.password

        // For the purpose of this demo app, show the password credential as an aler
        DispatchQueue.main.async {
            self.showPasswordCredentialAlert(username: username, password: password)
        }

    default:
        break
    }
}
```

> **Note**
>
> In your implementation, the `ASAuthorizationControllerDelegate.authorizationController(controller:didCompleteWithAuthorization:)` delegate function should create an account in your system using the data contained in the user identifier.

If the authentication fails, the authorization controller invokes the `ASAuthorizationControllerDelegate.authorizationController(controller:didCompleteWithError:)` delegate function to handle the error.

```
func authorizationController(controller: ASAuthorizationController, didCompleteWithE
    // Handle error.
}
```

Once the system authenticates the user, the app displays the `ResultViewController` which shows the user information requested from the framework, including the user-provided full name

and email address. The view controller also displays a Sign Out button and stores the user data in the keychain. When the user taps the Sign Out button, the app deletes the user information from the view controller and the keychain, and presents the `LoginViewController` to the user.

Request Existing Credentials

The `LoginViewController.performExistingAccountSetupFlows()` function checks if the user has an existing account by requesting both an Apple ID and an iCloud keychain password. Similar to `handleAuthorizationAppleIDButtonPress()`, the authorization controller sets its presentation content provider and delegate to the `LoginViewController` object.

```
func performExistingAccountSetupFlows() {
    // Prepare requests for both Apple ID and password providers.
    let requests = [ASAuthorizationAppleIDProvider().createRequest(),
                    ASAuthorizationPasswordProvider().createRequest()]

    // Create an authorization controller with the given requests.
    let authorizationController = ASAuthorizationController(authorizationRequests: r
    authorizationController.delegate = self
    authorizationController.presentationContextProvider = self
    authorizationController.performRequests()
}
```

The `authorizationController(controller:didCompleteWithAuthorization:)` delegate function checks whether the credential is an Apple ID (`ASAuthorizationAppleIDCredential`) or a password credential (`ASPasswordCredential`). If the credential is a password credential, the system displays an alert allowing the user to authenticate with the existing account.

Check User Credentials at Launch

The sample app only shows the Sign in with Apple user interface when necessary. The app delegate checks the status of the saved user credentials immediately after launch in the App `Delegate.application(_:didFinishLaunchingWithOptions:)` function.

The `ASAuthorizationAppleIDProvider.getCredentialState()` function retrieves the state of the user identifier saved in the keychain. If the user granted authorization for the app (for example, the user is signed into the app with their Apple ID on the device), then the app continues executing. If the user revoked authorization for the app, or the user's credential state not found, the app displays the log in form by invoking the `showLoginViewController()` function.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launch
    let appleIDProvider = ASAuthorizationAppleIDProvider()
    appleIDProvider.getCredentialState(forUserID: KeychainItem.currentUserIdentifie
        switch credentialState {
        case .authorized:
            break // The Apple ID credential is valid.
        case .revoked, .notFound:
            // The Apple ID credential is either revoked or was not found, so show t
            DispatchQueue.main.async {
                self.window?.rootViewController?.showLoginViewController()
            }
        default:
            break
        }
    }
    return true
}
```

See Also

Sign In with Apple

{} Simplifying User Authentication in a tvOS App

Build a fluid sign-in experience for your tvOS apps using AuthenticationServices.

`struct SignInWithAppleButton`

A SwiftUI view that creates the Sign in with Apple button for display.

`Sign in with Apple Entitlement`

An entitlement that lets your app use Sign in with Apple.

`class ASAuthorizationAppleIDProvider`

A mechanism for generating requests to authenticate users based on their Apple ID.

`class ASAuthorizationAppleIDCredential`

A credential that results from a successful Apple ID authentication.