Auth Integration Research & Considerations

Content

Introduction

In modern web applications, managing user authentication securely and efficiently is crucial. Two widely adopted approaches that simplify this task are **Federated Identity Services** and **Single Sign-On (SSO)**.

When building websites or applications that users need to log in to, one of the first questions developers face is:
"How do we handle authentication?"
In other words, how do we know who the user is, and how do we keep their account secure?

Traditionally, this meant building a system where users create a unique account with a username and password just for your app. But as the internet grew, this became increasingly frustrating — both for developers and users.

Imagine having to create a new key for every single building you visit. Your school has one, your gym has one, your job has one, your bank has one — and you have to carry them all and remember which key fits where.

That's where **Federated Identity** and **Single Sign-On (SSO)** come in.
These systems let users log in using a **key they already have**, like a Google, Facebook, or GitHub account — instead of creating a new one just for your app.

So instead of making users go through "create account → verify email → remember another password," you just say:
"Hey, want to log in with Google instead?"

This is **federated identity**: your application trusts another well-known service to vouch for the user's identity.

And if that same login works across multiple apps without asking the user to log in again, that's called **Single Sign-On (SSO)**.

What Is a Federated Identity Service?

As mentioned in the introduction, a **federated identity service** allows users to log in to your application using an identity they already have — such as a **Google**, **GitHub**, or **Microsoft** account — rather than creating a new one just for your app.

The term **federated** comes from the idea of **federation** — a group of independent systems agreeing to work together under a shared framework of trust. In this case, your application (the "relying party") **federates** or forms a trust relationship with an external **identity provider** like Google or Microsoft.

This works because your app **trusts** an external identity provider (commonly referred to as an **IdP** or Identity Provider) to verify the user's identity. When a user chooses to log in with, for example, Google, the flow looks roughly like this:

1. Your app sends the user to Google to log in
2. Google asks the user to authenticate (e.g., enter their password, use 2FA)
3. If successful, Google sends a secure message back to your app saying:
   → "This is Jane, and I vouch for her."

Your app never sees the user's Google password — it just receives **proof** that Google authenticated the user.

Federated identity systems rely on **open and standardized protocols** that ensure all of this happens securely. The most common protocols used are:

- **OAuth 2.0**
  → Think of this as a **delegation** protocol. It lets users grant limited access to their data or actions (like: "this app can view your email address") without sharing their password.
- **OpenID Connect (OIDC)**
  → Built on top of OAuth 2.0, OIDC adds **authentication**. It allows your app to receive information about who the user is — such as their name or email — in a secure, verifiable way.
- **SAML (Security Assertion Markup Language)**
  → Used more often in enterprise and older systems. It serves a similar purpose but is more complex and XML-based.

These protocols work by using encrypted tokens and signatures. They make sure that:

- Messages can't be intercepted or altered
- Only authorized apps can ask for login
- The identity of both the app and the user can be verified

Think of your app like an airport, and the user is a traveler trying to board a plane. Your app doesn't need to verify who the traveler is from scratch — it just checks their **passport** (token) issued by a **trusted government** (Google, Microsoft, etc.).

Instead of making users create a new identity just for your app, they arrive with one they already have — like showing a passport they've used many times before.
Behind the scenes, security protocols like **OAuth 2.0**, **OpenID Connect**, or **SAML** act like:

- **Scanners** checking for forgery (verifying the token is real and unaltered)
- **Secure border systems** (encrypted communication)
- **Digital stamps** (signatures proving the identity came from a trusted source)

Your app (the airport) never sees the user's password — it just gets a secure message from the identity provider (like Google) saying: **"This is Jane, and I vouch for her."**

What Is Single Sign-On (SSO)?

Single Sign-On (SSO) is a system that lets users log in once — and then access multiple applications without needing to log in again for each one.

Instead of remembering and typing separate usernames and passwords for every app, users authenticate once with a central system. That system then tells each app, **"This user is already logged in — you can trust them."**

SSO is commonly used in organizations where employees need access to many internal tools. But it's also widely adopted on the web — for example, when logging in with your Google account lets you access Gmail, YouTube, Drive, and more without extra login prompts.

SSO often works hand-in-hand with **federated identity services**, especially when logging into apps from **external providers** (like Google or Microsoft) rather than your own organization's login system.

SSO relies on the same secure protocols (like **OIDC**, **OAuth2**, or **SAML**) to ensure the login is **secure**, **verifiable**, and **private**.

Imagine you're staying at a large resort with several different buildings — spa, restaurant, pool, your hotel room — and each has its own security desk.

Without SSO, you'd need to **show ID and register separately** at every entrance (logging in to each app).

With SSO, you check in **once at the main reception** (central login). They give you a **wristband** (token) that you can use everywhere in the resort — no need to verify your identity again at each building.

The wristband is:

- **Proof you're a verified guest** (authenticated user)
- **Encrypted and hard to forge** (secure token)
- **Recognized by every building** (trusted across apps)

So, just like in a digital system with SSO, you log in once and get smooth access to everything — securely and without repeating yourself.

What Is the Difference Between Federated Identity & SSO?

While often used together, **Federated Identity** and **Single Sign-On (SSO)** solve different problems:

- **Federated Identity** lets users log in to your app using an existing account from a trusted third party — like Google, Microsoft, or GitHub.
  Instead of creating a new username and password, users "bring their identity" from another service.
    - Use **Federated Identity** when you want to simplify signup and login by letting users authenticate through external identity providers.
- **Single Sign-On (SSO)** allows users to log in **once** and access **multiple systems** without being prompted to log in again.
  It provides a seamless experience across multiple apps or services.
    - Use **SSO** when you have multiple apps or services and want to avoid repeated logins between them.

Federated Identity and SSO often go hand-in-hand:

- **Federated Identity** determines **who** is verifying the user (e.g., Google or Microsoft)
- **SSO** ensures that once verified, the user can move between systems without logging in again

Example:
A user logs in to your app using their **Google account** (Federated Identity). Then they can also access other internal apps or tools in your system **without re-entering their credentials** (SSO).

Why Use Federated Identity?

For this assignment, the goal is to integrate your app with a system external to your own — and a **federated identity** service is a straightforward and effective way to do just that.

Unlike traditional **SSO**, which is often used within large organizations to connect internal systems under a single login, **federated identity** is designed for broader scenarios. It allows users to sign in using accounts from well-known third parties like Google or GitHub, without requiring you to manage user credentials at all.

This makes **federated identity** especially suitable for a small educational app, where building a complex authentication system isn't the focus. It's simpler to implement, teaches you how modern login systems work, and clearly demonstrates external integration — which is exactly what the assignment is asking for.

1. Google OAuth 2.0 

Description: Google OAuth 2.0 is Google's implementation of the OAuth 2.0 protocol, used to enable secure and standardized login for web and mobile applications. It allows users to sign in with their Google account and grants applications limited access to profile information through access-tokens. It supports federated identity using protocols like OAuth 2.0 and OpenID Connect (OIDC) and is widely used to integrate Google login into backend systems, single-page apps, and mobile platforms. Google OAuth 2.0 works well with popular frameworks such as FastAPI, Flask, and Express.

Pros and Cons of Picking Google OAuth 2.0:

- **Direct integration with a major identity provider** – Enables login with Google without third-party intermediaries.
- **No vendor lock-in** – Full control of authentication flow, with no dependency on external platforms.
- **Strong educational value** – Helps developers understand token exchange, scopes, and authorization flows in detail.

- **Requires more setup** – Token handling, redirects, and user data extraction must be implemented manually.
- **Single-provider limitation** – If additional identity providers are needed (e.g., Microsoft, GitHub), similar flows must be built separately.
- **More error handling needed** – Token expiration, refresh logic, and edge cases must be handled in the application code.

Google OAuth 2.0 Conclusion:

Google OAuth 2.0 is a reliable, standards-based choice for implementing federated login. It offers a secure and flexible way to authenticate users via their Google account and provides an opportunity to learn the details of modern authentication protocols. Although it requires more manual implementation, it provides transparency and avoids external dependencies — making it a strong candidate for educational or backend-focused projects.

2. Firebase Authentication 

Description: Firebase Authentication is part of Google's Firebase platform, offering authentication services for web and mobile apps. It supports multiple identity providers like Google, Facebook, Apple, GitHub, as well as traditional email/password and phone number logins. Firebase is designed with a frontend-first mindset, which makes it especially appealing for mobile and single-page applications. It integrates naturally with other Firebase services such as Firestore and Cloud Functions.

Pros and Cons of Picking Firebase Authentication:

- **Smooth Integration with Google Services** – Works well if you're using other parts of Firebase or Google Cloud.
- **Frontend SDKs & UI Libraries** – Simplifies the implementation of authentication flows in frontend code.
- **Supports Major Federated Providers** – Built-in support for social and enterprise providers.
- **Generous Free Tier** – Thousands of monthly active users are supported at no cost.
- **Mobile App Friendly** – Especially robust support for Android and iOS app development.

- **Backend Integration Can Be Challenging** – Firebase Authentication is primarily optimized for frontend use; backend logic requires more effort.
- **Limited Server-Side Control** – Token handling and user data management are mostly abstracted by Firebase.
- **Heavily Tied to Google Ecosystem** – Works best when the entire app is built around Firebase services.

Firebase Authentication Conclusion:

Firebase Authentication is a great option if the focus is on frontend-heavy or mobile applications. However, it can feel limiting for server-side-heavy projects or when trying to deeply understand how authentication works under the hood.

3. Amazon Cognito 

Description: Amazon Cognito is AWS's identity management service that enables developers to manage user sign-up, sign-in, and access control. It supports both user pools (for managing app users) and identity pools (for granting temporary access to AWS services). Cognito supports federated identity through social providers, SAML, and OpenID Connect. It integrates deeply with other AWS services and is highly scalable, making it popular for production environments on AWS.

Pros and Cons of Picking Amazon Cognito:

- **Scalable and Production-Ready** – Trusted in large-scale applications and proven in enterprise environments.
- **Tight AWS Integration** – Ideal if you're using other AWS services such as Lambda, API Gateway, or S3.
- **Advanced Security Features** – Offers multi-factor authentication, device tracking, and token customizations.
- **Federated Identity Support** – Google, Facebook, and SAML providers can all be integrated.
- **Complex to Set Up** – Requires navigating user pools, identity pools, and AWS IAM roles, which can be overwhelming.
- **Steep Learning Curve** – The fragmented documentation and number of configuration options can be hard to manage.
- **Optimized for AWS Stack** – Best suited when your whole project is already within the AWS ecosystem.

Amazon Cognito Conclusion:

Cognito offers great capabilities for managing users and permissions but comes with significant setup complexity. For small-scale or educational projects, especially those not hosted on AWS, it might be more cumbersome than necessary.

4. Okta 

Description: Okta is a comprehensive identity and access management solution often used in enterprise settings. It supports single sign-on (SSO), multi-factor authentication, directory integration, and federated identity protocols like SAML and OIDC. Okta offers extensive administrative tools for managing users, groups, and roles. It also supports API access and SDKs in several languages including Java, Node.js, and Python.

Pros and Cons of Picking Okta:

- **Enterprise-Grade Features** – Designed with strict compliance and access control policies in mind.
- **Supports Major Identity Protocols** – SAML, OIDC, and even SCIM for automated user provisioning.
- **Robust Admin Tools** – Dashboards, group management, and policy controls are extensive and powerful.
- **API-First Architecture** – Easy to automate and integrate via REST APIs.

- **Not Beginner-Friendly** – The feature-rich setup can be overkill for small or learning-focused projects.
- **Limited Free Tier** – The free version is time-limited and lacks many advanced features.
- **Heavier Configuration Overhead** – More upfront setup and configuration compared to simpler solutions like **Google OAuth 2.0** or **Firebase Authentication**.

Okta Conclusion:

Okta is a powerful option for enterprise-grade identity systems, especially when handling complex user roles or internal systems. However, its pricing and complexity make it less ideal for quick prototyping or educational use cases.

5. Microsoft Entra ID 

Description: Microsoft Entra ID (previously Azure AD B2C) is a cloud identity solution designed for customer-facing applications. It supports social logins, enterprise identity providers, and local accounts, and it excels at handling complex user flows and access policies. Built on Microsoft's Azure platform, it's tightly integrated with Microsoft 365, Active Directory, and other Azure services. It is primarily designed for use with the Microsoft ecosystem, such as .NET applications.

Pros and Cons of Picking Microsoft Entra ID:

- **Supports Complex Authentication Flows** – Can handle advanced use cases with custom user journeys and policies.
- **Scalable and Secure** – Backed by Microsoft infrastructure, suitable for large apps or enterprise deployments.
- **Strong Microsoft Integration** – Seamless experience if users need to log in using Microsoft accounts.
- **Claims-Based Access** – Offers fine-grained access control using token claims and conditional access policies.

- **Steep Setup Requirements** – Setting up tenants, identity providers, and custom policies requires a lot of configuration.
- **Best with Microsoft Stack** – Integration examples and SDKs are focused around .NET and Azure-hosted platforms.
- **Overly Complex for Small Projects** – Most features go unused in smaller-scale applications.

Microsoft Entra ID Conclusion:

Microsoft Entra ID is a solid tool for large-scale applications or those built within the Microsoft ecosystem. But for educational purposes or small-scale experimentation, its complexity and focus on enterprise use make it harder to adopt quickly.

Final Conclusion:

After evaluating several identity providers, **Google OAuth 2.0** stands out as the most appropriate choice for this project. It provides a secure and standardized way to enable federated login through a trusted provider without relying on third-party identity platforms. Because it uses the **OAuth 2.0** and **OpenID** Connect protocols, it offers compatibility with modern authentication practices while maintaining full control over the flow.

This approach supports the goal of building a backend-focused educational project using FastAPI, where learning how authentication works — including how access tokens, scopes, redirects, and user information are handled — is a key objective. **Google OAuth 2.0** is also widely supported, well-documented, and suitable for integration into a variety of systems.

Comparison to Other Tools

While each identity provider we considered has its strengths, they all come with trade-offs that make **Google OAuth 2.0** the better fit for this project:

Firebase Authentication is a good choice for frontend-heavy or mobile applications. However, it abstracts away much of the backend logic, which limits learning opportunities for a backend-focused project.

Amazon Cognito provides a scalable identity system with strong AWS integration, but its setup is relatively complex. For educational purposes, **Google OAuth 2.0** offers a simpler and more focused path.

Okta is a powerful identity platform used in enterprise environments. While it supports **OAuth** flows, it introduces more configuration and is better suited for complex systems rather than lightweight educational projects.

Microsoft Entra ID (formerly Azure AD B2C) is highly customizable and well-integrated into the Microsoft ecosystem. However, its complexity and Azure-specific requirements make it less ideal for a small-scale, Python-based project.

Summary

In short, **Google OAuth 2.0** offers the best balance of flexibility, security, and educational value for this project. It enables secure federated login using an existing Google account, follows open standards, and provides insight into how authentication works at a protocol level. While other tools offer additional features or simplified setup, **Google OAuth 2.0** is better suited for gaining a deeper understanding of backend authentication without unnecessary abstraction.