



Authentication Decision Guide

Use this guide to choose how users sign in to your application and how it authenticates requests to Medplum. Compare the options and tradeoffs, then follow the linked technical docs to implement your choices.

Companion to the [Authentication](#) and [User Management](#) docs. Builds on [Identity Management: A Practical Guide](#).

Staff, partner clinicians, patients, and caregivers may need different approaches. For each population:

1. **Choose your identity provider.** Use Medplum-managed authentication or an external IDP.
2. **Choose your token flow** if you use external or existing authentication.
3. **Choose your matching key** if you use an external IDP authorization-code flow or token exchange.

All approaches need [user provisioning and session lifecycle decisions](#). For tenancy and permissions, use the [Access Control Decision Guide](#).

Decision 1: Which identity provider will you use?

An identity provider (IDP) manages credentials and authenticates users. Choose based on your existing identity systems and SSO requirements.

Option A: Medplum-managed authentication

Medplum manages credentials, registration, password resets, and MFA.

- **Advantages:**
 - Built-in authentication and login components without a separate IDP integration.
- **Tradeoffs:**
 - Does not meet a requirement to use an existing corporate IDP.
- **Best fit:** Applications without an existing identity system, including patient-facing applications.

Implementation: [Redirect login](#) or [Embedded Login](#).

Next: Skip Decisions 2 and 3 and continue to [Before You Implement](#).

Option B: External IDP with application-based routing

Configure an external IDP on the Medplum ClientApplication. For example, your patient portal and clinician application can use different providers.

- **Advantages:**
 - Reuses existing identities and advanced features from providers such as Okta or Auth0.
 - Different applications can use different identity providers.
- **Tradeoffs:**
 - Multiple upstream IDPs within one application need additional routing, such as an identity broker.
- **Best fit:** Each application selects a known IDP or broker for its users.

Implementation: [External Identity Providers](#).

Option C: External IDP with domain-based routing

Medplum selects the IDP from the user's email domain, such as routing `user@clinic-a.com` to Clinic A's corporate IDP.

- **Advantages:**
 - An organization's SSO applies across Medplum applications, including the Medplum App.
- **Tradeoffs:**
 - Applies to all users of the domain, including developer accounts.

- Requires Medplum setup support for both hosted and self-hosted deployments. Hosted setup requires Enterprise.
- **Best fit:** Customer organizations bring their own IDPs and their email domains should determine where users sign in.

Implementation: [Domain-level Identity Providers](#).

Next for external IDPs: Choose your token flow below.

Decision 2: What is your token flow architecture?

For applications using external or existing authentication. Choose how the application obtains Medplum credentials. Here, “client” means your browser or native app; “server” means your application backend.

Both the authorization-code flow and token exchange can use the same external IDP. The difference is whether Medplum handles the IDP callback or your application obtains an external access token and exchanges it.

Option A: External IDP authorization-code flow (three-legged OAuth)

Medplum handles the external IDP callback and returns an authorization code to your application, which exchanges it for Medplum tokens.

- **Advantages:**
 - The client can call Medplum directly using the SDK and authenticated components.
 - The client does not need to retain an external access token for this flow.
- **Tradeoffs:**
 - Requires redirect login and coordinated callback settings.
 - Does not provide the client with an external access token for other services.
- **Best fit:** Medplum is the primary client API and redirect login fits your application. Start here unless another flow meets a specific requirement.

Implementation: [External Identity Providers](#).

Option B: Token exchange

Your application obtains an external access token through its own login flow. Medplum validates it with the configured IDP and issues a user-scoped Medplum token.

- **Advantages:**
 - Adds Medplum access to an existing login without a second interactive sign-in.
 - Your application retains control of the external login and can use external credentials with other services that accept them.
- **Tradeoffs:**
 - Requires an external access token that the IDP's UserInfo endpoint can validate.
 - Your application manages both external and Medplum credential lifecycles.
- **Best fit:** Your application already obtains an external access token and needs to add Medplum access, potentially alongside other services.

Other integrations alone do not require token exchange. Integrations running in Bots or your backend with service credentials do not require the client to hold an external token.

If you choose token exchange: where will you keep the Medplum token?

Both options below use **user-scoped Medplum tokens**. Choose based on whether the client should call Medplum directly or through your backend.

B1: In the client

The client retains the exchanged Medplum token and calls Medplum directly.

- **Advantages:**
 - Direct SDK, authenticated React component, and WebSocket subscription access.
 - No application proxy for Medplum requests.
- **Tradeoffs:**

- The client manages both credentials, including expiry and logout, with storage appropriate to its platform.
- **Best fit:** Direct Medplum access is valuable and credentials can reside in the client.

B2: On the server

Your backend retains each user's exchanged Medplum token and uses it to fulfill their requests.

- **Advantages:**
 - Medplum credentials stay on the server.
 - Fits an existing backend that coordinates multiple services, while preserving each user's Medplum access policy.
- **Tradeoffs:**
 - Your backend manages per-user tokens, renewal, and logout.
 - Proxying adds latency and infrastructure to operate.
 - Live updates need a backend-to-client delivery path; components expecting direct Medplum access need adaptation.
- **Best fit:** Your architecture requires backend mediation or keeping Medplum credentials out of the client.

Implementation: [Token Exchange](#).

Option C: On-Behalf-Of (single machine-to-machine user)

Your backend uses a single machine-to-machine application identity (ClientApplication) and makes requests on behalf of individual human users. Medplum applies the specified user's access policy and records their attribution.

- **Advantages:**
 - Supports a backend that owns authentication without requiring external user tokens that Medplum can exchange.
 - Preserves user-level access policies and attribution with one application credential.
- **Tradeoffs:**

- Your backend must maintain user mappings and apply the correct user context to every request.
- Proxying adds latency and infrastructure to operate.
- Medplum WebSocket subscriptions do not support this On-Behalf-Of pattern.
- **Best fit:** A backend must mediate Medplum access and cannot use supported user-token exchange.

With server-held token exchange, the Medplum credential represents the user. With On-Behalf-Of, it represents the application, which selects the user context for each request. Prefer user-scoped exchange when either approach meets your requirements.

Implementation: [On-Behalf-Of](#).

Next: For authorization-code flow or token exchange, choose your matching key below. For On-Behalf-Of, define your backend's user mapping and continue to [Before You Implement](#).

Decision 3: How will you match external identities to Medplum users?

For external IDP authorization-code flow or token exchange. Choose which identifier connects the external login to the correct Medplum account.

Option A: Email matching

Medplum matches users by email by default, including with external IDPs.

- **Advantages:**
 - Simple when each person has a distinct email and changes are infrequent.
- **Tradeoffs:**
 - Email changes require updating the Medplum login account, not just the clinical profile's contact email.
 - Shared email addresses cannot distinguish separate people.
- **Best fit:** Distinct emails and manageable account updates. An external IDP alone is not a reason to change the matching key.

Option B: External ID matching

Use the IDP's subject identifier. Enable subject-based matching and include the corresponding externalId when provisioning through the [Invite API](#).

- **Advantages:**
 - Identity remains stable when contact email changes.
 - Separate people can have distinct identities even if they share a contact email.
- **Tradeoffs:**
 - Provisioning and login must agree on the identifier; changing IDPs requires a migration plan.
- **Best fit:** An established IDP provides stable identifiers, email changes frequently, or people share contact addresses.

Pair external IDs with [SCIM](#) when your IDP should drive user provisioning and lifecycle management in Medplum.

Implementation: [Using External IDs](#), [Updating User Emails](#).

Before You Implement

How will users join your application?

A Patient or Practitioner record alone does not create a login. User and ProjectMembership are protected resources. For administrator-led provisioning, use the [Invite API](#) to create or link the user, profile, and membership. Other supported entry points include [SCIM](#) and [patient self-registration](#).

Choose [project-scoped or server-scoped users](#) based on whether identities belong to one project or need access across projects. For patient and caregiver applications, give each person their own identity and explicitly grant access to the appropriate records.

Who manages MFA, recovery, and logout?

Use Medplum's MFA for Medplum-managed password login and the external IDP's MFA for external login. Decide how recovery and offboarding work, including ending both external and Medplum sessions where applicable.

Implementation: [MFA](#), [Access Token Management](#), [Logout](#).

Record your choices for each population. Test sign-in, account changes, recovery, token expiry, logout, and offboarding. If your backend mediates access, also verify that requests use the correct user's context and fail when that context is missing.