IT Project Guidance

On User Modelling, Provisioning, Contextualisation, and Attribute Resolution

Version:

0.1

## Purpose

This guidance addresses the correct handling of user identity and persona attributes during the establishment of authenticated sessions. It explains the move away from in-system authentication, outlines modern identity practices using external IdPs, and provides guidance on handling contextual information such as group memberships and roles.

## Synopsis

While authentication is now typically delegated to external identity providers (IdPs), application developers still face the challenge of correctly collecting, interpreting, and storing identity and context attributes. This document clarifies what should and should not be expected from an IdP, explains how to associate multiple identifiers to a single user, and presents best-practice onboarding flows to ensure both accuracy and privacy are maintained over time.

## Contents

[Purpose 1](#_Toc201921408)

[Synopsis 1](#_Toc201921409)

[Contents 2](#_Toc201921410)

[Purpose and Audience 4](#_Toc201921411)

[Scope 4](#_Toc201921412)

[Background 4](#_Toc201921413)

[Historical Patterns: In-system Authentication 5](#_Toc201921414)

[Current Authentication Patterns: Delegated Authentication 5](#_Toc201921415)

[Current Authentication Patterns: Mixed Mode Authentication 5](#_Toc201921416)

[SaaS or SaaP 6](#_Toc201921417)

[Foundational Concepts 6](#_Toc201921418)

[Persons, Personas, Users and Identifiers 7](#_Toc201921419)

[Attributes and Contextual Relationships 9](#_Toc201921420)

[Entity Attributes 9](#_Toc201921421)

[Relationship Attributes 9](#_Toc201921422)

[Identity Attributes 10](#_Toc201921423)

[Misused Identifiers: A Note on False Universality 11](#_Toc201921424)

[Each System Has Its Own Users 12](#_Toc201921425)

[Authorisation is Local 13](#_Toc201921426)

[Identity (only) Brokering 14](#_Toc201921427)

[Context Belongs Where It Is Created 14](#_Toc201921428)

[Provisioning 15](#_Toc201921429)

[Self-Provisioning 15](#_Toc201921430)

[Manual Provisioning 16](#_Toc201921431)

[Bulk Provisioning 17](#_Toc201921432)

[Email based Invitation Provisioning 18](#_Toc201921433)

[Code Based Invitation Provisioning 19](#_Toc201921434)

[SCIM-Based Provisioning 19](#_Toc201921435)

[Considerations: Early Identity Matching via Sector Identifiers 20](#_Toc201921436)

[Guiding Principles 21](#_Toc201921437)

[Identity is not Context 21](#_Toc201921438)

[Avoid Token Enrichment by Brokers 21](#_Toc201921439)

[Contextual Placement Must Be System-Governed 21](#_Toc201921440)

[Sector Identifiers Belong in Sector Systems 22](#_Toc201921441)

[Design for IdP Independence 22](#_Toc201921442)

[The Limits of OIDC Trust Patterns 22](#_Toc201921443)

[OIDC Incompleteness 23](#_Toc201921444)

[OIDC Misuse 23](#_Toc201921445)

[Provisioning Failure Modes and Identity Ambiguity 24](#_Toc201921446)

[First-Time Identity with No Match 24](#_Toc201921447)

[Recycled Email Addresses 24](#_Toc201921448)

[Changed sub with Stable iss 24](#_Toc201921449)

[Reused sub in a Different Context 24](#_Toc201921450)

[No Issued Time or Subject Provenance 25](#_Toc201921451)

[Recommended Relying System Practices 25](#_Toc201921452)

[Avoid relying on email 25](#_Toc201921453)

[Avoid using email as a primary key 25](#_Toc201921454)

[Track iss, sub 25](#_Toc201921455)

[Track email if available 25](#_Toc201921456)

[Permit changes to email 25](#_Toc201921457)

[Track activity metadata 25](#_Toc201921458)

[Provide merge tooling 26](#_Toc201921459)

[Support identity rotation 26](#_Toc201921460)

[Treat new issuers as new users unless mapped 26](#_Toc201921461)

[Permit changes of iss and sub 26](#_Toc201921462)

[Halt suspicious merges 26](#_Toc201921463)

[Why National or Sector Identifiers Must Not Appear in Identity Tokens 27](#_Toc201921464)

[Retrieving Sector Identifiers via Authorised Post-Authentication Calls 28](#_Toc201921465)

[Contextualisation 29](#_Toc201921466)

[Modelling Group Membership and Roles 29](#_Toc201921467)

[The Misuse of Identity Providers for Group and Role Context 30](#_Toc201921468)

[On Token Enrichment and Architectural Overreach 31](#_Toc201921469)

[Source and Timing of Contextualisation 32](#_Toc201921470)

[Conclusion 33](#_Toc201921471)

[Appendices 34](#_Toc201921472)

[Appendix A - Document Information 34](#_Toc201921473)

[Versions 34](#_Toc201921474)

[Images 34](#_Toc201921475)

[Tables 34](#_Toc201921476)

[References 34](#_Toc201921477)

[Review Distribution 34](#_Toc201921478)

[Audience 35](#_Toc201921479)

[Structure 35](#_Toc201921480)

[Diagrams 35](#_Toc201921481)

[Acronyms 35](#_Toc201921482)

[Terms 35](#_Toc201921483)

[Appendix B - Document Information 36](#_Toc201921484)

[Appendix C – Use Sub to disambiguate users 36](#_Toc201921485)

# Purpose and Audience

This document provides architectural guidance on the correct handling of contextual information in federated identity systems. It clarifies the distinction between identity and context, explains the limitations of relying on identity providers (IdPs) for role or group data, and outlines reliable design strategies for contextualisation in education, organisational, and government systems.

This guidance is intended for architects, solution designers, and system owners responsible for identity and access management in federated or multi-tenant environments. It will also benefit developers implementing user provisioning, authentication, and authorisation mechanisms, particularly where identity is sourced from third-party IdPs.

Note:  
This document is currently information dense, covering multiple interrelated areas of identity architecture. Future iterations will include simplified versions to make the material more accessible to a broader audience—without compromising on accuracy or completeness. However, the effort is worthwhile, given the high risk of fundamental design errors in this space.

# Scope

The document addresses how and when user context should be assigned relative to identity provisioning. It distinguishes roles and memberships, outlines typical lifecycle stages for context assignment, warns against over-reliance on identity tokens, and recommends appropriate mechanisms for context governance. It assumes OpenID Connect (OIDC) is used for authentication and considers sectoral or national identity overlays where relevant.

# Background

Modern digital systems supporting identity management must account for three distinct but related concerns: **User Provisioning**, **Contextualisation**, and **Role Assignment**. These are often conflated in practice because they appear together in onboarding flows. However, they serve different purposes and must be kept logically separate within the system design.

**Provisioning** is the act of creating a system-recognised User. This is typically triggered by a user signing in for the first time, being invited, or being added through a synchronisation mechanism like SCIM. Provisioning creates an internal user record and links it to one or more DigitalId entries that represent their identity in external IdPs.

**Contextualisation** involves establishing the user’s position within the structural groups of the system. These may be organisational units, workspaces, classes, or other domain-specific groupings. This step is where the system determines *where* a person belongs.

**Role Assignment** defines *what* the user is allowed to do within those groups—e.g. whether they are a viewer, contributor, manager, or administrator.

In a mature system, each of these steps may be managed by different services or components. They may also occur at different times: for example, a user may be provisioned before their context is known, or assigned a new role long after their initial onboarding.

Although this document treats these concerns separately in definition, the remainder of the guidance discusses them together in relation to key onboarding patterns. This better reflects real-world flows, where provisioning, contextualisation, and role assignment often occur in concert.

## Historical Patterns: In-system Authentication

Prior to the widespread adoption of external identity providers, most applications implemented their own authentication systems. These typically involved a User table containing a username, email address, display name, and a hashed salted password—often with application-defined rules for password strength, expiry, and reset mechanisms. This model placed ongoing security responsibilities, including credential storage, authentication logic, retry monitoring, and auditing, on each individual system.

While functional in isolated contexts, these in-system authentication models created significant long-term risks. Systems built primarily to deliver business functions were now also responsible for sensitive security infrastructure. Over time, as threat surfaces grew and compliance expectations rose, this approach became both fragile and burdensome.

## Current Authentication Patterns: Delegated Authentication

Modern architectures have since moved toward a pattern of **delegated authentication**—offloading identity verification to trusted external identity providers (IdPs), typically using standards like OpenID Connect (OIDC).

In these systems, the application’s User table remains but is minimal, often containing only an internal ID and a display name. A separate DigitalId table records the link between the internal user and one or more external identities, each defined by an iss (issuer, representing the IdP) and sub (subject, the unique user ID within that IdP).

## Current Authentication Patterns: Mixed Mode Authentication

The shift to *Delegated Authentication* improves security and maintainability, however does not eliminate all edge cases.

Some users cannot be served by external identity providers. This includes the initial system administrator, who must be present before any external IdP can be integrated or used. It also includes users in underserved demographics, such as children who fall below the minimum age thresholds imposed by commercial identity services. In addition, certain operational environments—such as offline or air-gapped systems—may preclude the use of external sign-in entirely, requiring alternative local authentication methods.

In these cases, the system must still support local authentication. This is typically handled via a LocalCredential table, storing salted password hashes and/or alternate secrets, and optionally supporting multi-factor authentication (e.g. via capturing SMS or authenticator apps identifiers). Such fallback mechanisms must be treated as first-class security concerns and require the same rigour as any modern authentication system.

## SaaS or SaaP

It is important to recognise that in this regard, commercial SaaS platforms are no different as a Software as a Product (SaP), or custom-developed systems.

They will all implement one or more of these models:

* Allowing multiple simultaneous identity providers (e.g. Google and Microsoft accounts),
* Supporting local credential fallback,
* Enabling invitation-based flows with optional one-time passwords,
* relying on brokers or delegated SCIM provisioning paths.

Historically, applications maintained a User table containing usernames, display names, and email addresses, and often implemented their own authentication using password hashes. This placed long-term security responsibilities on systems not designed for it and introduced vulnerabilities.

Modern architectures favour delegating authentication to trusted external identity providers.

In these systems, a local User table still exists but contains only internal identifiers and minimal attributes such as display name. A related DigitalId table records the relationship between this internal user and their identifiers in each external identity service—typically storing an issuer (iss, representing the IdP) and subject (sub, the IdP-issued user ID).

Despite this shift, some users cannot be served by external IdPs. This includes initial system administrators and users in demographics unsupported by common IdPs (e.g. children under age thresholds). These cases require internal credential handling, such as a separate in-system LocalCredential table with optional multi-factor support. This exception underscores that some degree of security responsibility remains unavoidable.

SaaS services are no different than custom developed systems and will implement one or more of these options (allow multiple sign-ins, allowing in-system authentication).

With the background established, we now turn to the foundational concepts that underpin identity and contextualisation and provisioning thereof. These concepts form the logical scaffolding for the rest of the document.

# Foundational Concepts

Designing for contextual integrity begins with correct logical modelling of Users, Identity, Accounts, and Persons. These concepts are often conflated in practice, leading to brittle systems, integration failures, and access confusion. This section introduces each concept in turn, showing how they relate—and why, even when related for effect, they must remain distinct.

Identity is not a monolith. To reason clearly and design correctly—whether in stand-alone or federated, long-lived contexts—systems must distinguish between users, accounts, persons, personas, attributes, relationships, identifiers, and names.

## Information Elements

### Use, Users and Identification

A black and white image of a number

AI-generated content may be incorrect.

Figure 1: Users of Systems

All systems are built to be used by **Users**, who interact with them through interfaces, perform actions, and consume or manage resources.

In order to grant access, permit activities, track usage, or assign responsibility, systems must be able to distinguish—identify—their Users one from another.

A black background with white rectangles

AI-generated content may be incorrect.

Figure 2: Users are specific to a single System

Identification of a User is always relative to the system: the system itself must recognize and distinguish Users within its own domain.

The individual User record in turn then serve as the basis for session management, permissions, and audit trails.

Identification may be handled by internal logic, or more commonly now, by relying on trusted external systems—identity providers (IdPs)—to vouch for who the user is. These assertions are treated as sufficiently authoritative to link an external identity to an internal User record.

Thus, even when leveraging credentials or tokens issued by another system, identification is not merely about authentication—it is about situating the User within the system’s own domain.

However, it’s a little bit more complex. Specifically, often misunderstood, are the relationship between Users and Accounts, and Users and Persons, covered next.

### Accounts

A white rectangular sign with black text

AI-generated content may be incorrect.

Figure 3: Individual Free Consumer Facing System Users

In individual free consumer-facing system, when a person signs up, a User is created. That’s it.

#### Single User Accounts

A black and white rectangular object

AI-generated content may be incorrect.

Figure 4: Individual Account based Consumer Facing System Users

In individual *commercial* consumer-facing systems, when a person signs up, a User is created, and an Account is created for them and associated to their User record[[1]](#footnote-2).

The Account determines whether the identified User is entitled to access the system as a whole, what services and/or features they are entitled to use, enforce usage limits, etc.

They are both the User and the Account holder. Access is immediate, and billing, if applicable, is tied directly to the individual—although often deferred until after an introductory or trial period.

If the same person wishes to create a different Account, they generally must create a separate User record, which in turn means they’ll first need another external identity (e.g. a second email address).

Alternatively, If they want to coordinate with others, a different – group-account-based -- model is required.

#### Multi-User Accounts

A black and white rectangular sign with black text

AI-generated content may be incorrect.

Figure 5: Commercial Multi-User Account based Service

In multi-user-account-based systems, the model shifts. The Account is associated with a group—most often an organization such as a school, business, or agency. The Accounts billing arrangements are tied to that organization, not to the individual Users.

It is technically possible for a User to belong to multiple Accounts.

In these systems, Users must therefore be contextualized—they must be associated with the correct Account, which determines it’s associated Users their access or limits to services.

Systems that fail to model Accounts explicitly often resort to brittle workarounds—inferring account membership from email domains, relying on invitation flows without persistent linkage, or embedding account logic into role labels. These shortcuts may work temporarily, but they collapse under scale, federation, or change.

In federated environments, where identity is distributed across many institutions, this becomes even more challenging. The identity provider may not include reliable organizational identifiers, or the Tool Provider may not have a mapping between identity attributes and account records. Without a clear account model, provisioning becomes ambiguous, and access control becomes fragile.

Correct modelling requires that Users are always associated with an Account, and that the Account is treated as a first-class entity—distinct from the User, and capable of representing both individuals and organizations.

The provisioning of Users within group accounts is the crux of the matter.

### Persons, Personas, Users and Identifiers

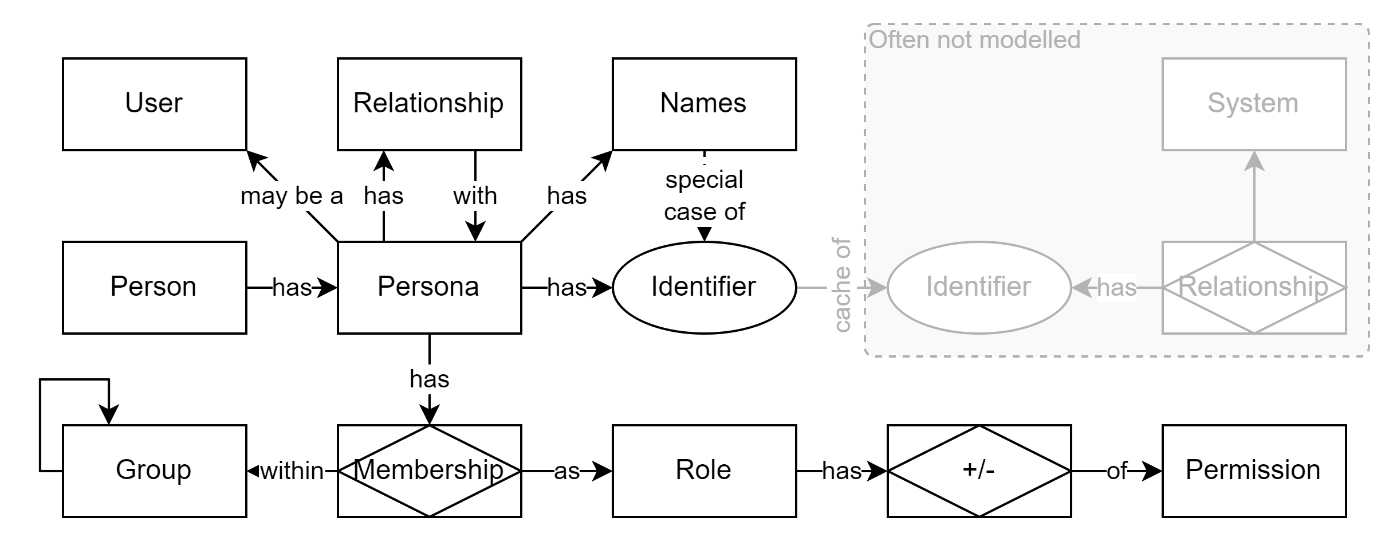


Figure 6: Person, Personas, Users, Identifiers, Names

The other foundational concept often conflated with Users is the *Person*.

Digital systems frequently treat “Users” as if they are singular, fixed, and directly represent a person. However, this assumption collapses several real-world distinctions that, if left unaddressed, inevitably surface as integration failures, access confusion, or identity mismatches. IT is the automation of the management of information used to represent the real world. As such, it requires careful analysis of the real world first, in order to represent it correctly later.

At the core of that representation is the Person—a physical human being.

#### Personas

While from a limited perspective it may appear that a Person has a single, fixed identity, in practice, a Person presents different *Personas* in different contexts, depending on their relationship with a group, service, or system. These Personas are not arbitrary; they reflect how someone is recognised socially, legally, professionally, or personally within a specific context. The boundaries between Personas serve to isolate or share information in contextually appropriate ways.

A group of white rectangular boxes with text

AI-generated content may be incorrect.

Figure 7: Person Personas

Over time, a Person may adopt new Personas or adapt existing ones. They may take on new roles, change names after marriage or transition, have separate public and private identities, or represent themselves differently across cultural or generational boundaries. These changes are normal and expected—and must be safely accommodated by systems that manage access and continuity.



Figure 8: Simplistic collapse of Person and Persona

While it is common for systems to collapse Person and Persona into a single entity, the longer a time that a system manages information on behalf of a Person, the more certainty that the system will need different Personas to manage the changes, and the boundaries that go with them.

Each Persona is associated with one or more **identifiers** issued by external systems—whether modelled or unmodelled. These include email addresses, usernames, national student numbers, tax IDs, or system-specific keys. As discussed later under *Identity Attributes*, such identifiers are not intrinsic to the Person or their Persona—they arise from a relationship with the system that issued them. Each is governed by its own rules for scope, format, uniqueness, and lifecycle. An identifier is not a universal truth, but a pointer granted by context.

Names are a special case: a pre-digital identifier used to distinguish people within unmodelled social contexts—hence the term “personal identifier.” A Person may hold several names across different settings. A Persona may present a full legal name and formal title at work, a simplified version at school, a nickname at home, and a different name entirely in an ethnic or religious context. Each of these may be valid, recognisable, and even legally sanctioned. People often carry multiple names simultaneously, expressed in different scripts, languages, or cultural conventions. The assumption of a single canonical name may be technically convenient, but it is socially and legally insufficient—especially in systems that span cultures, lifespans, or jurisdictions.

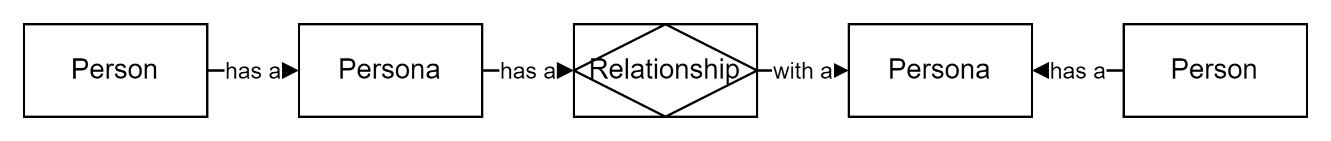


Figure 9: Personas have Relationships with other Personas

Each Persona may also form relationships with other Personas and hold memberships in various groups. These relational structures matter—particularly in systems that rely on trust, access delegation, or coordination across domains.



Figure 10: Oversimplification by collapse of User/Person/Persona

As alluded to before, many system designs simplify unintentionally. They create a User table that assumes one Persona per human, attach names and identifiers to that record, and treat it as the enduring source of truth. This works—until it doesn’t. When people appear to change (but haven’t), or when two people share an identifier (and shouldn’t), systems built on collapsed assumptions begin to leak trust, context, and correctness.

This is especially problematic in systems that model Customers, Suppliers, Students, Teachers, or Parents as separate entities. These are not distinct types of people—they are *temporal* roles or relationships to a group that a Person may hold for a duration of time. Modelling them incorrectly as separate *enduring* entities ensures that continuity is lost. A Teacher who was once a Student, or a Parent who is also a Customer, may be represented multiple times, with no awareness of the underlying Person. The same applies to Users: if a User is also a Customer or holds another role, their details are often duplicated across both the User table and the role-specific table. This is not just inefficient—it is a fundamental information modelling flaw.



Figure 11: Users have associated Persons

The majority of Users are Persona. However, not all Personas are Users. A **User** is a system-specific representation of a Persona—a thin object that anchors identity and access within a particular system. It defers social and contextual attribute collection to its associations.

Understanding this distinction is essential for designing systems that are resilient, interoperable, and capable of maintaining continuity across time, roles, and domains.

--------

This guidance does not insist that all implementations model full Person–Persona–Identifier separation in code. But it does ask that architects and system owners understand what is being flattened—and the risks of doing so invisibly. In long-lived, federated, or government-grade systems, making these relationships explicit may be the only way to sustain continuity, clarity, and trust across time and context.

Note: the distinction between intrinsic identity and contextual representation is explored further in the section titled The Misclassification of Identity Attributes. That section shows how identifiers such as national IDs, sector codes, or system numbers—often mistaken for identity attributes—are instead distinguishing attributes of a person’s relationship with a system that issued them.

### Attributes and Contextual Relationships

While Users and Persons and Personas run the risk of being collapsed into indistinct objects due to superfluous analysis, the same can occur with Attributes.

In essence, there are attributes about an *entity* – in this case, a Persona – and attributes describing the *relationship* of the entity to another context entity – the context. They are not the same.

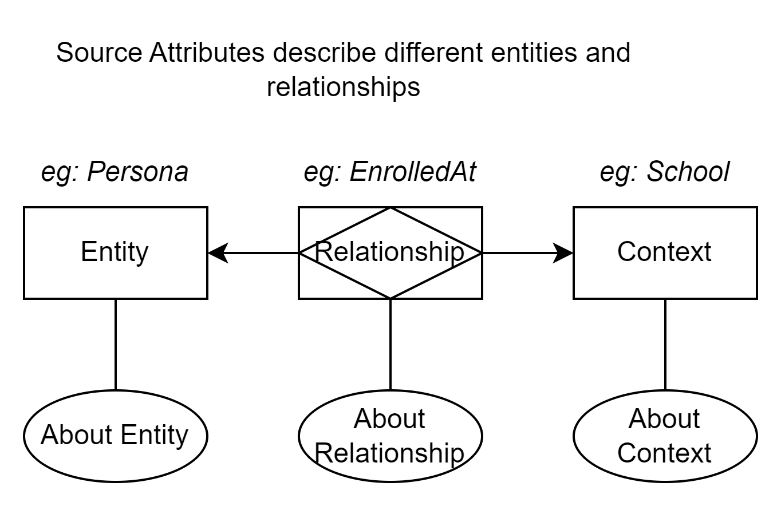


Figure 12: Different types of attributes

This same distinction exists in this context as well: in managing identity, systems must distinguish between attributes that describe the person themselves, and attributes that arise from their relationship with the system or with other entities in it. This distinction is essential for clarity, correctness, and long-term maintainability.

#### Entity Attributes

Entity attributes describe a person in themselves—as a Persona—without reference to any particular context or system. These include characteristics such as preferred name, language spoken, country of residence, or time zone. They are intrinsic or general-purpose properties of the person and can be used across contexts without loss of meaning.

#### Relationship Attributes

In contrast to entity attributes, many facts about a person only exist because they are *in relationship* to something else. These are **contextual attributes**, and they describe how the person is situated within the system—such as their role, membership type, status, enrolment level, or any conditional permissions.

Relationship attributes are not intrinsic to the person—they are specific to their connection with another entity, such as a group, organisation, project, or platform. For example, someone might be a “Lead Teacher” in one school, a “Mentor” in another, and a “Student” in a separate professional programme.

Correct modelling of these relationships typically involves join structures—records that link the user to another entity and carry attributes specific to that context. This allows the same user to hold different roles or attributes in multiple contexts, without conflict or ambiguity.

Among these contextual attributes, **roles** are just one form—a familiar and often useful construct for access control. However, they should not be treated as the only or dominant type of relationship attribute. Relationships may also represent enrolment, guardianship, supervision, historical participation, or standing invitations. These relationships often have status, duration, or provenance that roles alone cannot capture.

Systems must be careful not to reduce all contextual data to roles. Doing so leads to overly coarse-grained models and limits the system’s ability to express nuanced relationships. It can also introduce errors—such as granting unintended access, leaking assumptions across contexts, or flattening time-bound or hierarchical associations into static role labels.

This tendency to flatten contextual identifiers into intrinsic attributes also leads to their misuse across system boundaries, which is addressed next.

#### Identity Attributes

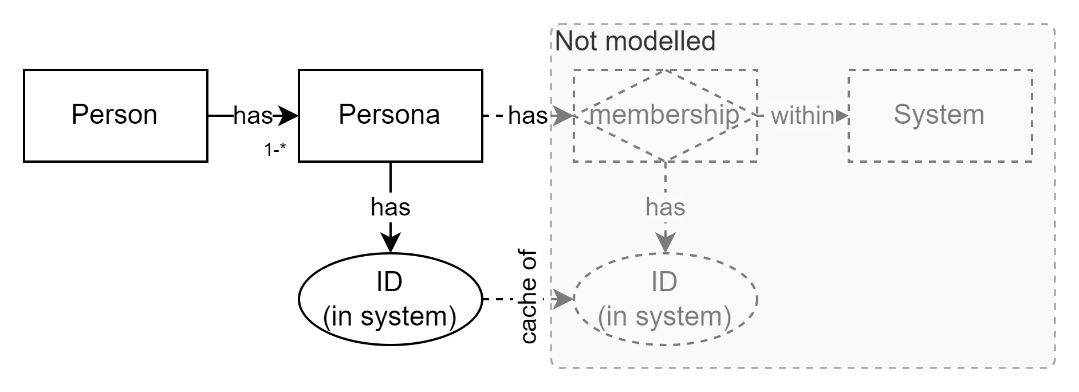


Figure 13: Unmodelled Systems

A particularly widespread design error arises when **identity attributes**—such as national student numbers, tax identifiers, or corporate user IDs—are treated as if they were entity attributes. In truth, they are not. These identifiers only exist because a person has a relationship with another system—such as a national education registry, a tax authority, or an employer.

Logically, identity attributes belong to the relationship between the person and the system that issued the identifier. However, in most implementations, that external system is not explicitly modelled. The relationship is flattened, and the identifier is stored directly on the person as if it were intrinsic. This is a practical shortcut, but it is conceptually incorrect that often causes issue later on.

The consequence of this misclassification is that systems may treat identifiers as universally valid, stable, or comparable—when in fact they are system-specific and context-bound. It also complicates scenarios where a person has multiple identities across systems, or where identifiers must be rotated, revoked, or disambiguated.

Recognising that identity attributes are relationship-bound—even if the other system is not modelled—improves schema correctness, supports better federation design, and prevents long-term data entanglement across domains.

## General Concepts

Before delving into system design, it is important to note that some of the forthcoming concepts may initially appear elementary or widely known. Nevertheless, given that these foundational principles are so often misunderstood or neglected in practice, it is prudent to review them here to ensure clarity and to help avoid common pitfalls that frequently affect many systems.

### Each System Has Its Own Users

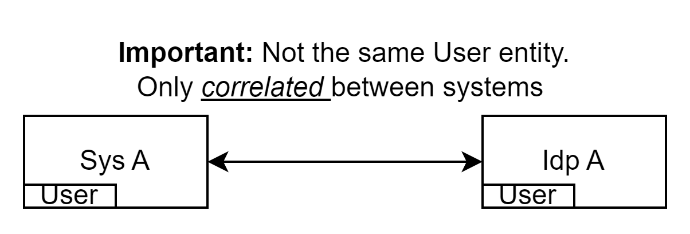


Figure 14: Each System has its own Users.

This section corrects the assumption that identity is shared simply because systems use the same identity provider or about the same person: in federated systems, each system has its own user records. Even when using a shared Identity Provider (IdP), the consuming system maintains its own internal user representation.

The IdP is also a system in its own right. It has users—its users—and its primary function is to expose - provide - a user's *identity* (excluding credentials) attribute data related to them. Hence Identity Provide

The consuming system *also* has users—its own records, profiles, and entitlements. While these two users relate to the same persona, these two system user entities are not the same; they are only linked by a trustable correlation, not by shared instance or state.

What actually happens when a user attempts to access System A is they are redirected to an IdP, which authenticates them and returns a token containing some of the user's attributes. System A then validates this token to trust it and uses any IdP user information embedded within it to in turn locate or create a local System A user record. Once created, this user record, in a different system, is distinct from the IdP’s.

From that point on, System A manages its own system user record internally, using its own mementos to continue session user information (cookies or headers). The IdP’s token is *not* reused by System A beyond the initial session establishment.

Local access, authorisation, personalisation, and auditing are based on System A's internal data—not the IdP.

### Authorisation is Local

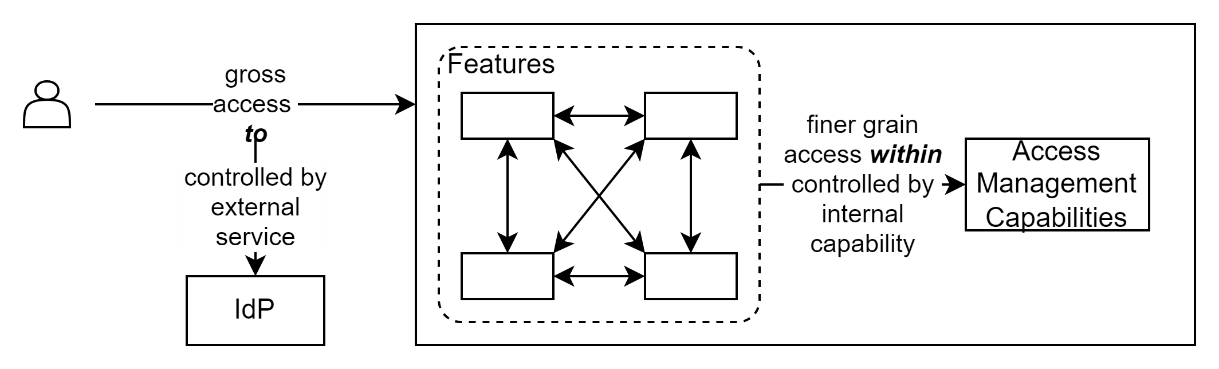


Figure 15: Access To is distinct from Access Within a system

It is important to distinguish between authorisation *to use* a system and authorisation *within* a system. The same term is often used to describe both, which has led to persistent confusion in system design.

An Identity Provider (IdP) can confirm that a user is who they claim to be, and provide access *to* a system—but it is the system itself that must determine what the user is allowed to do *within* it. These scopes are not interchangeable.

The logic for both assigning permissions and authorising actions must be embedded in the system delivering the business function, not delegated to external identity infrastructure.

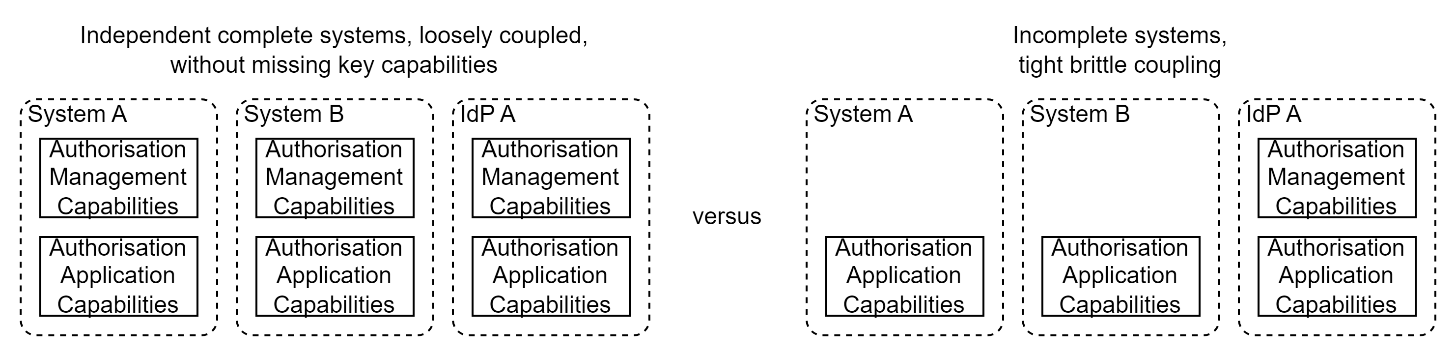


Figure 16: Authorisation Management and Implementation is Local

Using IdPs to manage permissions leads to confusion and fragility. Every system should determine its own permission, role, and access logic. A label like 'teacher' may mean something different in a learning system than in a reporting portal or finance application. Role logic must be tailored, maintained, and auditable within the consuming system.

### Identity (only) Brokering

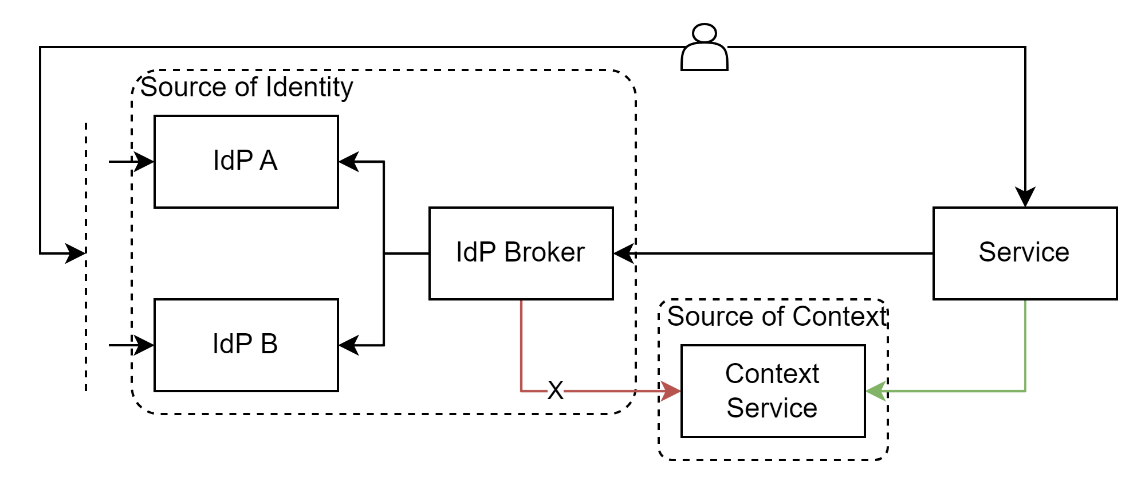


Figure 17: retrieve identity and context from separate, appropriate, sources

A secondary source of confusion arises when identity brokering is introduced. In federated environments, an intermediary broker may authenticate users by delegating to one or more upstream IdPs—for example, a national education IdP that federates to cloud-hosted school-based identity services. While brokering authentication is valid and necessary, the concept is sometimes stretched beyond its design, to brokering other information, as well as context and authentication.

It may seem convenient to treat the broker as a bridge not only for identity authentication, but also for aggregating or injecting user attributes from other systems. This conflation is problematic. An identity broker is not a general-purpose integration hub. Its role is to negotiate identity assertions—not to serve as a conduit for cross-domain attribute resolution.

Confusing these responsibilities leads to architectural drift: tokens become overloaded with external context, systems become dependent on indirect data sources, and the clarity of identity boundaries is lost. The result is fragile integration, poor traceability, and an increased surface area for data breaches.

Identity brokering does not change the fact that each consuming system maintains its own user records. Even when authentication is delegated to a broker or upstream IdP, the relying system must still represent users locally and independently. Identity brokers assert identity—they do not carry or synchronise user state, permissions, or context.

### Context Belongs Where It Is Created

Systems sometimes preload tokens with context—such as enrolment, roles, or organisational affiliation—that originates from other systems. These attributes are not identity. Nor are they attributes of the Persona - they are instead describing a persona’s *relationship* to a Context.

Being not attributes of an identity they should not be managed by an Identity Provider. Instead, they should be resolved at runtime via deliberate calls to the system that owns them.

Embedding such data in tokens violates boundaries, reduces auditability, and disperses entitlements. Instead, consuming systems should fetch what they need from source systems using authorised, traceable interfaces.

### A Note on the False Universality of Identifiers



Figure 18: Identifiers per domain

Many systems inadvertently treat contextual identifiers as if they were globally unique or stable. This is a common but dangerous simplification.

A typical example is the email address. Developers often treat email as a reliable proxy for identity, using it as a primary key, login credential, or precondition for account recovery. But email addresses can be reassigned, forwarded, aliased, or abandoned. In schools and corporates, they may be recycled across users or autogenerated with little human review. Using them as a core identity reference risks accidental user crossover, silent data leaks, and continuity failures.

Another example is the national student number (NSN). Within the education sector, the NSN is widely used to track a learner across providers—but it is only meaningful within the domain that issued it. It may not exist for all users (e.g. foreign students, young learners), and it cannot be assumed to hold significance in other sectors (e.g. health or justice). Treating an NSN as a universal identity key creates brittle integrations and breaks interoperability principles.

Similar problems occur with tax numbers, healthcare IDs, or internal system keys. Each was designed to serve a particular operational domain. Their reuse across systems—especially without clear boundaries, purpose limitation, or explicit linking—invites confusion, privacy breaches, and governance drift.

The key architectural error is assuming that *any* identifier—no matter how stable it appears—is sufficient to represent a person across systems. In practice, identifiers must always be evaluated in terms of their issuing authority, intended scope, and revocation policy. They can support continuity, but they cannot safely *define* it.

Where continuity across systems is required, it must be *managed*—through mapping, correlation, or explicit onboarding flows—not *assumed* through shared identifiers.

## Modern Identity Protocols: OAuth and OIDC

Beyond the pitfalls of incorrect information modelling, is a foundational to understand the purpose and limits of established protocols and standards.

### SAML

While this document focuses on OAuth and OpenID Connect (OIDC) as the dominant modern identity protocols, it is worth briefly acknowledging **SAML (Security Assertion Markup Language)**. SAML was widely adopted in the early 2000s for federated authentication, particularly in enterprise and education sectors. It remains in use today, especially in legacy systems and integrations with older identity providers.

However, SAML is XML-based, lacks native support for modern web and mobile application patterns, and is generally considered less developer-friendly than OAuth/OIDC. As a result, it is no longer the preferred choice for new implementations. Its continued relevance is largely tied to backward compatibility and institutional inertia rather than technical superiority.

Note: While SAML is no longer the preferred protocol for new implementations, its maturity and long-standing use across federated environments continue to offer valuable architectural insights. Many of the principles it established—around trust, assertion, and separation of concerns—remain relevant and should inform the evaluation of newer protocols, even if SAML itself is not adopted.

### OAuth-Based Access and Delegation

OAuth 2.0 is a widely adopted framework for delegated authorization. It allows a client application (System A) to obtain a token that grants it permission to access protected resources—typically hosted by another system (System B, the resource server)—on behalf of a user. This is done without requiring the user to share their credentials directly with System A.

In most implementations, the user authenticates with an identity provider (System B), which issues a token to System A. System A then uses this token to access APIs or services hosted by System B, acting with the permissions the user has granted. OAuth is therefore about authorizing a system to access resources, not about authenticating a person, authorizing actions within a system, or describing the user.

A crucial aspect influencing the user experience in OAuth and OIDC flows is the lifespan and management of the tokens issued during authorization.

Consider a scenario in which a user wishes to order prints from a photo store using images stored in Google Photos. If the user has recently logged into their Google account—perhaps even before visiting the print shop's website—the authentication flow leverages that existing session. In this situation, the user will not need to re-enter their Google credentials to grant the print store access. Instead, they may only be prompted to explicitly approve that the store is permitted to view or use their photos. Here, the permission screen is not about proving who the user is, but about granting a new application (the print store) access to specific resources.

The token issued to the print store by Google is fundamentally what enables this seamless experience. This token, typically a short-lived access token (and sometimes, a longer-lived refresh token), must be securely stored by the print store for the duration of the session or for as long as it needs to interact with the user's Google Photos on their behalf. As long as the token is valid, the user won’t have to log in again or repeat the permission step for each new operation—viewing, selecting, or printing photos can occur transparently.

The persistence and expiry of these tokens are critical. If the access token expires (which it inevitably does for security reasons), the print store may attempt to use a refresh token to silently obtain a new access token without the user’s involvement. If a refresh token is not available, or if it too has expired or been revoked (perhaps because the user changed their Google password or explicitly disconnected the store in their Google account settings), then the next operation requiring access will force a re-authentication or a fresh consent—meaning the user will see the Google login or permission screen again.

Keep in mind that the rights provided were to the System, to Read Photos, not a User.   
To provide permissions to a User, OIDC was then developed on top of OAuth.

Consider a scenario in which a user wishes to order prints from a photo store application (System A, or SysA) using images stored in Google Photos (System B, or SysB). If the user has recently logged into their Google account—perhaps even before visiting SysA’s website—the authentication flow leverages that active session with SysB. In this situation, the user does not need to re-enter their Google credentials to grant SysA access. Instead, they may only be prompted to explicitly approve that SysA is permitted to view or use their photos stored in SysB. Here, the permission screen is not about proving who the user is, but about granting a new application (SysA) access to specific resources in SysB.

The token issued to SysA by SysB is fundamentally what enables this seamless experience. This token, typically a short-lived access token (and sometimes a longer-lived refresh token), must be securely stored by SysA for the duration of the session or as long as it needs to interact with the user's Google Photos (SysB) on their behalf. As long as the token (or refresh token, used to silently refresh the access token) remains valid, the user won’t have to log in again or repeat the permission step for each new operation—viewing, selecting, or printing photos can occur transparently via SysA's requests to SysB.

If a refresh token is not available, or if it too has expired or been revoked (for example, if the user has changed their Google password or explicitly disconnected SysA in their Google account settings), then the next operation requiring access will trigger a re-authentication or fresh consent—meaning the user will once again see the Google login or permission screen or both.

It’s important to recognize that the rights provided through OAuth are granted to SysA, permitting it to read or manage resources in SysB, not directly to the user. To address the need for representing the user's identity—rather than just application-level access on behalf of the user—OpenID Connect (OIDC) was developed as an extension to OAuth, layering identity information atop the authorization model.

It is critical to recognize that this seamless exchange relies on user accounts being present within *both* the requesting and responding systems—either through prior administrative setup or by allowing users to self-provision. This prerequisite underpins the entire flow: without effective provisioning processes to establish or synchronize user identities across multiple systems, even the most robust token-based authentication and authorization models cannot deliver a smooth, integrated user experience. The role of provisioning, therefore, becomes pivotal, as it transforms abstract identity assertions into actionable user records within each participating application, and will be covered in detail further down.

### OIDC-Based Identity Assertion

While OAuth permits a system to authenticate itself to another system on behalf of a user, it doesn’t permit a user to authenticate itself to either system. To address this gap, OpenID Connect (OIDC) was introduced as an extension to OAuth 2.0.

OIDC adds a standardized authentication layer on top of OAuth’s authorization framework.

Technically, it introduces the ID token -- a structured JSON Web Token (JWT) that contains claims about the user’s identity and the authentication event – which permits client applications to verify who the user is, not just what they’re allowed to access. It also defines scopes (such as openid, email, profile) and endpoints (like userinfo) to retrieve user attributes in a consistent and secure manner. In short, OAuth 2.0 handles what the user can access; OIDC handles who the user is.

In a typical OIDC scenario, a flow is a follows: a user who wishes to sign in to Sys A (our print shop) will be redirected to a different system (defining scopes requested), an Identity [token] Provider System (Sys I). They are prompted for their credentials, and when accepted, are prompted for consent to share the requested scopes’ attributes with the Sys A (email, profile, etc.) an access token is provided [directly to the SysA, or Browser? ] … which permits Sys A to query Sys I for an ID token, [containing attributes limited to consented scope] via a secure back channel (avoiding a potentially compromisable browser), and optionally a refresh token. And then… whatever happens next…. And then Sys A creates a session record, and a local session cookie, storing any refresh token that was retrieved against it.

In a typical OpenID Connect (OIDC) workflow, when a user wants to sign in to SysA (the print shop), they’re redirected to a third-party identity provider (IdP), called Sys I, which receives details about the requested ‘scopes’ (such as openid, email, or profile). The user authenticates with Sys I and, if additional information is requested, is asked to consent to share those specific attributes (like ‘email’ or ‘profile’ info) with SysA. After the user gives consent, Sys I sends an authorization code back to SysA through the user’s browser. SysA then securely exchanges this code—using an API call to Sys I—for an access token, an optional refresh token.

With the access token, SysA can interact with the IdP’s oidc specific APIs as needed. It uses one of these APIs to retrieve the oidc ID token – a cryptographically signed but open JWT object-- which is sufficient to provide user scoped details to SysA.

At this point, with the information provided, SysA establishes a session for the user, before discarding the Sys I sourced ID token, but saving any refresh token it received.

Additionally, it issues a session cookie for ongoing authentication between the server and browser. If the session cookie expires, SysA can use the stored refresh token—so long as it’s still valid—to obtain new access tokens, keeping the user logged in without requiring another authentication step. If both tokens have expired, the user will need to log in again to continue.

It is again critical to observe that the Sys I had to be provisioned with the user beforehand, and Sys B either had to be provisioned or permit self-provisioning.

It is critical to recognize that this seamless exchange relies on user accounts being present within *both* the requesting and responding systems—either through prior administrative setup or by allowing users to self-provision. This prerequisite underpins the entire flow: without effective provisioning processes to establish or synchronize user identities across multiple systems, even the most robust token-based authentication and authorization models cannot deliver a smooth, integrated user experience. The role of provisioning, therefore, becomes pivotal, as it transforms abstract identity assertions into actionable user records within each participating application, and will be covered in detail further down.

Having established the foundational concepts, we now explore how users are introduced into systems—through various provisioning mechanisms.

# Provisioning

As pointed out a couple of times above, which ever protocol being used -- OAuth or OIDC -- for the above flows to work, it is a prerequisite that both systems be provisioned beforehand, or – potentially - permit just in time (JIT) provisioning.

Provisioning is the process by which a system creates a local representation of a person—a User—and links them to one or more external identities. It enables participation, governance, and contextualisation within the system, but does not by itself determine what the user can do (roles) or where they belong (groups or organisations). Provisioning may occur automatically, administratively, or interactively, depending on the system’s architecture and operational model.

Despite its necessity, implementing effective provisioning is notoriously challenging and can easily result in a frustrating experience for end users if not handled carefully.

Several provisioning approaches exist—including self-provisioning, manual setup by administrators, bulk imports, invitation workflows, and integrations via standards like SCIM or LTI. Each method determines how users are created, assigned context, and granted roles. We explore each of these models in detail below.

## Self-Provisioning

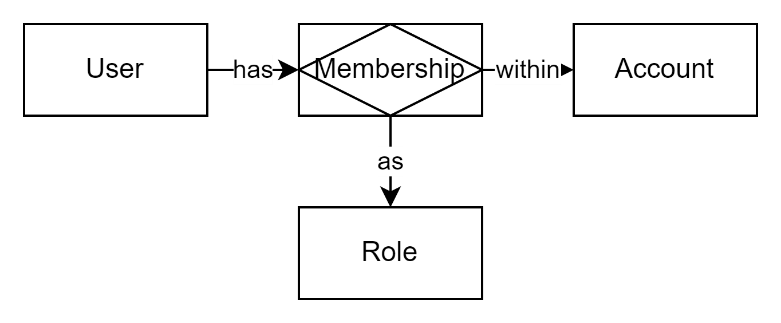


Figure 19: Self provision creates Account as needed and associates User.

Self-provisioning occurs when a person initiates their own entry into the system. This is commonly triggered by arriving at a public-facing application, choosing a sign-in method (e.g. “Sign in with Microsoft”), and authenticating via an external identity provider (IdP). If the system has no record of the user’s external identity—typically identified by a unique combination of iss (issuer) and sub (subject ID from the IdP)—then it proceeds to create a new local User record and link it to the supplied external identity.

In SaaS systems, self-provisioning often goes further: not only is a new user created, but also a new logical Account, Organisation, or Subscription. This establishes a private space within which the user operates—often including storage, settings, permissions, and the ability to create contexts such as groups or teams.

The system will typically assign the self-provisioned user a privileged system role within this account—such as Owner, Administrator, or Coordinator. This ensures they can act meaningfully within the space they just created, even though no further users or groups yet exist.

At this point, contextualisation is minimal. The user is a member of the account by default, and as they create groups or contexts (such as teams, projects, or classes), the system may automatically assign them administrative roles within each new context they create. This pattern enables immediate functionality without requiring a formal onboarding workflow.

While efficient, self-provisioning has implications. It assumes:

* The user has access to an acceptable identity provider,
* The creation of new accounts or tenancies is permitted by the system's governance rules,
* The initial user can act safely as an administrator, including inviting others or configuring access.

Systems must decide whether self-provisioning is enabled globally, or only within certain constraints (e.g. trial accounts, education providers, or domain-restricted emails).

It is also worth noting that the identities used during self-provisioning are unverified beyond what the IdP provides. If contextual trust or linkage to authoritative identity systems (e.g. a national education registry) is required, it must be established *after* provisioning.

While self-provisioning supports autonomy and speed, it is not always appropriate—particularly in systems where participation is gated, roles are pre-defined, or accounts are centrally managed. In such cases, more directed onboarding methods are preferred.

Self-provisioning often operates as **Just-In-Time (JIT) provisioning**, where the user is created dynamically at the time of first successful sign-in via an external identity provider. This means no user record exists beforehand, and no manual setup is required—only a valid identity token is needed to trigger provisioning. While efficient, this model assumes a level of trust in the external IdP and a tolerance for minimal context at the time of entry.

Where self-provisioning is inappropriate or unsupported, systems typically fall back on administrative or preloaded models. The most basic of these is **manual provisioning**.

## Manual Provisioning

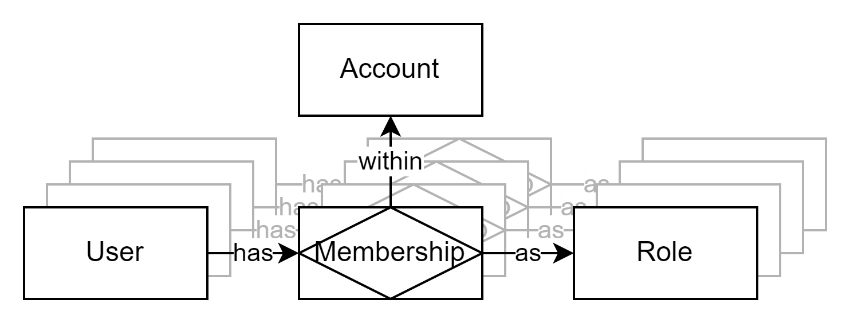


Figure 20: Manual Provisioning

Manual provisioning is the most rudimentary form of onboarding, common in legacy or small-scale systems where automation is limited or system ownership is tightly controlled. In this pattern, an administrator explicitly creates each user in the system, either directly in a user interface or through a back-office process.

The admin supplies all required attributes—name, email (if used), and optionally sets a password or triggers an email to notify new users of the setup being complete. The user may be assigned roles and group memberships immediately, or added later as needs arise.

Manual provisioning has limited scalability, offers no identity verification beyond what the administrator enters, and is prone to error or inconsistency. However, it remains useful in certain cases:

* When onboarding a small number of trusted users (e.g. system testers, early-stage deployments),
* When operating offline or without a usable IdP,
* Or when establishing foundational users in a system before automation is configured.

Because it bypasses identity provider flows, manual provisioning should only be used when auditability, context, and intentionality can be ensured through other controls.

## Bulk Provisioning

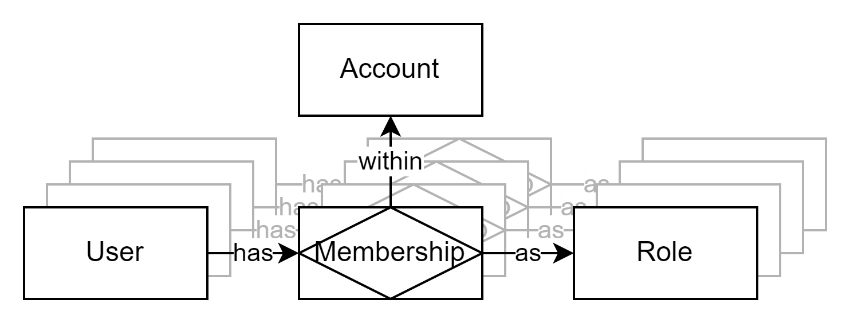


Figure 21: Bulk Provisioning (same as Manual, just automated)

Bulk provisioning is a rudimentary but widely used method for onboarding users at scale. It is especially common in environments where technical integration is limited, but a known list of users is available—such as the start of a school year, a training intake, or a pre-registered event.

The process typically begins with a manually prepared dataset—such as a spreadsheet, a Student Management System export, or an HR report. This dataset includes key fields such as name, email address, and intended role or group membership. The system then ingests this data through an upload or API call.

There are two basic implementation models:

1. **Pre-provisioning of users**  
   The system creates User records in advance, often marked as inactive or pending. When someone later signs in using an external identity provider, the system attempts to match the incoming identity—usually by email—to the existing user. If successful, the identity is linked and the user is activated. This model avoids creating users with no identity but relies heavily on email uniqueness and stability, which can be unreliable in federated or youth contexts.
2. **Bulk-generated invitations**  
   Instead of creating users up front, the system generates time-limited invitations and sends them to the listed contacts. Each invitation carries the intended role and group. When the recipient signs in, the system creates the user (if needed) and assigns the appropriate role. This approach delays user creation until necessary and avoids ghost records, but still assumes that the email contact is correct and trusted.

Bulk provisioning is operationally straightforward. It does not require deep system integration or formal IdP agreements. However, it also lacks verification, auditability, or fault tolerance. It cannot confirm whether users received the invitation, whether the identity that signs in truly belongs to the intended person, or whether the supplied data was accurate. It is therefore best used for low-risk roles, or in systems with secondary validation steps (e.g. teacher confirmation or code-based activation).

While crude, bulk provisioning remains effective in many real-world settings, particularly when paired with context-specific moderation or delayed access enforcement. More dynamic and structured approaches are covered in the next sections.

## Email based Invitation Provisioning

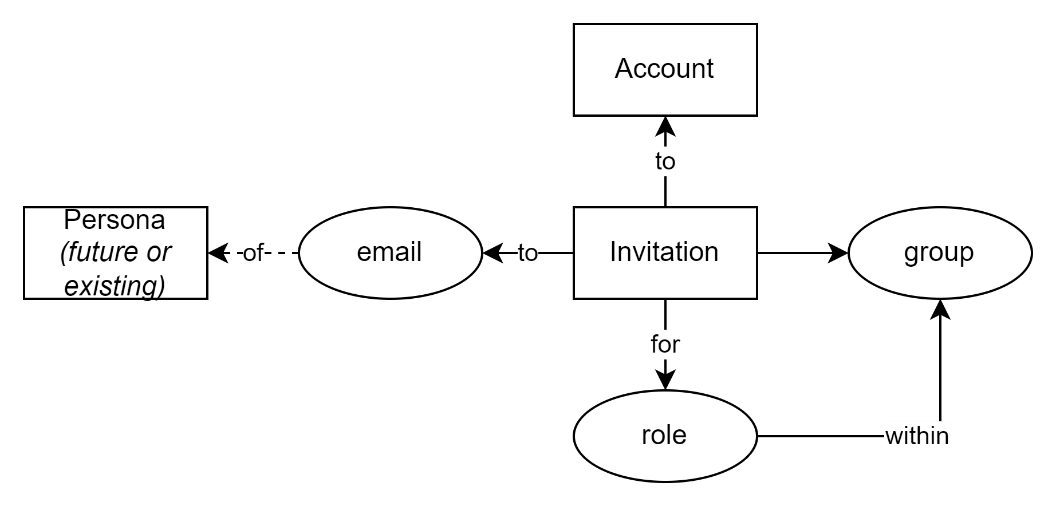


Figure 22: Invitation based Provisioning

Invitation-based provisioning occurs when an existing user initiates the onboarding of another person into the system. Rather than allowing open self-registration, this pattern enables controlled growth of participation, with context and role predefined by the inviter. It is commonly used in systems where roles carry significant responsibilities or access—such as educational platforms, organisational tools, or subscription-based services.

The process begins with an authorised user selecting a context (such as a group, class, or organisation), a role to assign, and an email address for the intended invitee. The system creates an Invitation record that includes these details, along with an expiry timestamp (e.g. valid for 24 or 72 hours). This record is not tied to an existing user; it represents a future user in waiting.

The invitation is delivered via email and contains a secure link directing the invitee to the system. Upon arrival, the invitee is prompted to authenticate using an accepted external identity provider. If no matching DigitalId is found, the system creates a new User record and links it to the external identity. It then checks for a valid invitation matching the user’s email and, if found, completes the onboarding process by:

* Associating the user with the correct Account or Subscription,
* Assigning them the specified Role within the specified Context (e.g. “Teacher in Room 5”),
* Optionally, notifying the inviter of successful onboarding.

This approach allows organisations to maintain control over who joins, and in what capacity, without requiring administrative oversight of every step. It also decouples identity authentication from role granting: the identity provider confirms who the user is (to the extent the IdP can), while the invitation process defines *why* they are joining and *what* they will be allowed to do.

Some systems allow invitations to be reused if multiple users need access to the same group in the same role (e.g. onboarding a cohort of parents or students). Others treat invitations as single-use, to preserve strict traceability.

This pattern supports both first-time users and returning users. If a user already exists and receives a new invitation, the system simply updates their contextual relationships—no new user provisioning is needed. In this way, invitations serve both as a provisioning mechanism and as a tool for expanding participation over time.

Invitation-based provisioning is often complemented by approval workflows or moderation queues in regulated or high-trust environments. In these cases, an administrator may need to review or activate the user after the invitation has been accepted but before full access is granted.

## Code Based Invitation Provisioning

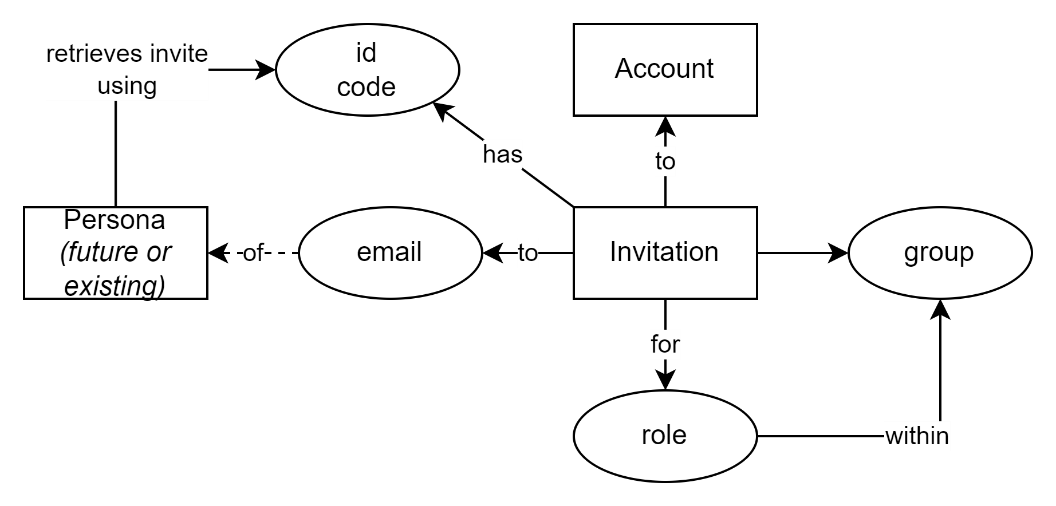


Figure 23: Code based Invitation

In addition to email-based invitations, some systems support **code-based onboarding**. In this model, the inviter generates a reusable or single-use access code linked to a predefined context and role. Instead of receiving a direct invitation link, recipients are given the code through other channels—such as printed materials, in-person sessions, or school intranet pages. Upon accessing the system, the user authenticates through an external IdP and enters the code. The system matches the code to a known invitation context, provisions the user (if not already present), and assigns the intended role.

This variant is common in classroom environments, training courses, assessments, or situations where users may not have email addresses or where mass-personalised distribution is impractical. It simplifies logistics but requires tighter controls around code expiry, misuse prevention, and optional approval.

## SCIM-Based Provisioning



Figure 24: SCIM based provisioning

While invitation-based provisioning is sufficient for most educational and civic systems—allowing existing users to add others with clear context—some enterprise or corporate environments require a more centralised and automated approach. For these use cases, SCIM (System for Cross-domain Identity Management) provides a standardised protocol for synchronising user records between identity authorities and applications.

SCIM is typically used when an upstream system, such as a corporate identity provider, school directory service, or HR platform, is responsible for defining who should exist in downstream systems.

Through SCIM endpoints on the downstream system, the source system can create, update, or deactivate users without requiring direct user action. This supports high-scale onboarding and consistent account lifecycle management across tools.

A SCIM-based integration usually results in:

* The creation of a User record (including attributes like name and email),
* The optional inclusion of group memberships from the source system,
* The alignment of account activation and deactivation with the source’s policies.

However, it is important to note that SCIM **does not convey application-specific context or roles directly**. The groups and roles described in a SCIM payload refer to constructs from the source system (e.g. Active Directory groups), which may not map cleanly to the target system’s concepts of groups, contexts, or permissions.

This leads to one of the key challenges of SCIM: **mapping**. For SCIM-based provisioning to assign roles or place users in meaningful contexts, either:

* The source system must place users into specific groups named according to conventions understood by the target application, or
* The target system must implement custom logic to interpret incoming SCIM group names and map them to internal roles and contexts.

This often requires a negotiated agreement between the source and target system owners. In practice, either side may attempt to reduce implementation burden by shifting the complexity to the other. As a result, SCIM is most successful in tightly integrated corporate ecosystems where naming conventions are predictable and change management is centrally controlled.

SCIM can be used as a provisioning tool for **structured, centrally-governed environments**. It enables automated user creation but does not replace the need for context-setting, role assignment, or identity correlation—all of which must still be handled separately.

SCIM should not be seen as a universal solution – it is **poorly suited to federated identity environments**, such as those spanning multiple schools, providers, or regional systems. In these contexts:

* The identity broker (or IdP hub) should not have complete knowledge of each downstream system’s group structures or access policies,
* The federation may involve thousands of semi-autonomous entities, making it infeasible to enforce shared SCIM group structures,
* Onboarding is more effectively handled through invitation-based flows, where context and role can be set precisely by each receiving system.

## LTI-Based provisioning

Learning Tools Interoperability (LTI) is a protocol developed by 1EdTech (formerly IMS Global) to enable secure, privacy-conscious integration between educational platforms. It is most commonly used to connect a Tool Consumer, such as a Learning Management System (LMS), with a Tool Provider, like an assessment or content delivery service. LTI is designed to support delegated trust and just-in-time (JIT) thin provisioning of user sessions, without requiring the exchange of personally identifiable information (PII).

The protocol begins with a one-time trust establishment between the Tool Consumer and the Tool Provider. Once this relationship is in place, the Tool Consumer can dynamically generate a launch link to the Tool Provider. This link includes metadata describing the context of the session—such as course name, activity type, and user role—as well as a correlation identifier that represents the user within the Tool Provider’s scope. Crucially, this identifier is not a third-party or national identifier, but a scoped token that allows the Tool Provider to associate session data without knowing the actual identity of the user.

Whether the Tool Consumer includes low-sensitivity attributes like a nickname or display name is a matter of usability rather than necessity. These attributes can improve the user experience but are not required for the protocol to function. The Tool Provider does not authenticate the user directly, nor does it rely on external identity providers. Instead, it trusts the introduction made by the Tool Consumer and provisions the session based on the metadata received. After the activity is completed, the Tool Provider returns results associated with the correlation ID to the Tool Consumer, maintaining the privacy of the user throughout the process.

LTI offers several advantages. It eliminates the need for pre-provisioning user accounts, aside from the initial trust setup. It preserves PII, enabling data exchange without violating privacy agreements or data-use policies. It also supports flexible, dynamic integration across platforms.

However, because the launch must originate from the Tool Consumer, access to the Tool Provider is inherently scoped to institutions or systems that support LTI. Where LTI is not universally adopted, access will require an alternate integration method that may compromise the privacy and simplicity that LTI is designed to uphold. It also introduces a second integration path, which undermines the value proposition of developing for LTI in the first place. Furthermore, LTI is not widely implemented across the broader ecosystem, severely limiting its use to established market incumbents. This restricts its currency and relevance in a dynamic and evolving environment where interoperability and agility are increasingly critical.

## Considerations: Early Identity Matching via Sector Identifiers

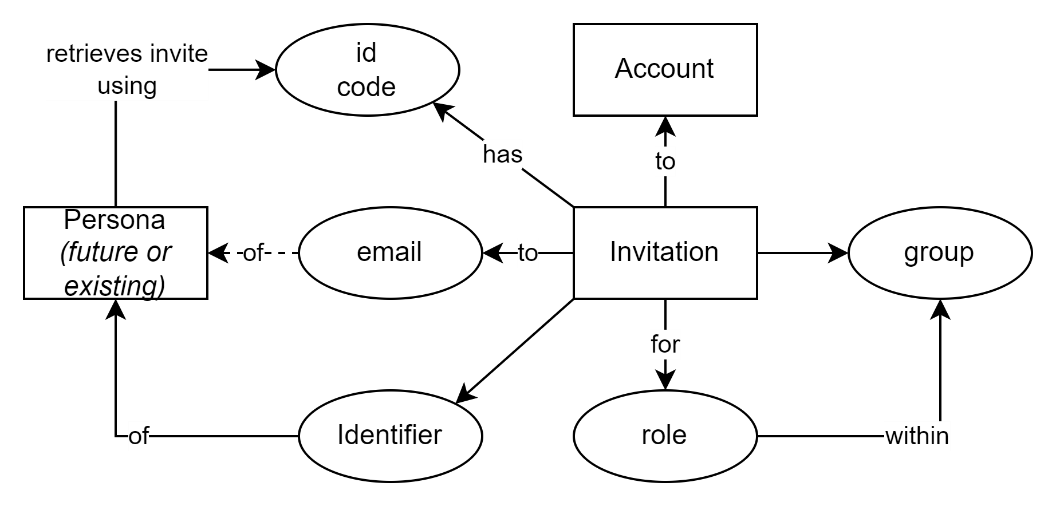


Figure 25: Invitation with embedded system Identifiers

In tightly controlled provisioning flows—such as invitations, bulk assignments, or SCIM-based imports—it is technically possible to identify users early by using a known sector or national identifier (such as an NSN) instead of relying solely on email addresses or waiting for identity protocol exchanges like OIDC. This can bypass the need for additional identity resolution services, provided the identifier is known, unambiguous, and correctly entered.

However, this approach introduces significant risks. Identifiers may be mistyped, misassigned, or reused across contexts. It also presumes that the provisioning actor has access to the identifier and understands its proper use, including any constraints on its scope, formatting, or applicability.

Where identifiers are governed, unique, and verified—such as in official sector systems—this pattern may reduce lookup overhead and simplify flows. But where data quality is mixed, such shortcuts can lead to incorrect identity-to-person associations, especially when users change affiliations or the same identifier is reused inadvertently.

Use of sector identifiers for pre-resolution must therefore be accompanied by validation steps and audit trails. No roles or access should be granted until identity integrity is confirmed. Even where this method is used, it does not eliminate the need for contextualisation logic to be handled by the relying system.

# The Limits of OIDC Trust Patterns

While the OpenID Connect (OIDC) specification assumes that the combination of iss (issuer) and sub (subject) forms a durable and globally unique user identifier, this does not hold in real-world deployments. Organisations routinely change identity providers due to contract expiry, cost considerations, feature gaps, integration barriers, or policy shifts. These transitions are not theoretical edge cases—they are routine occurrences in most medium-to-large environments.

In such transitions, a new IdP will typically generate a different sub for each user—even when the person remains the same. In some cases, the same iss value may be retained despite a change in underlying infrastructure, making it impossible to determine whether a new (iss, sub) pair represents a returning user or an entirely new one.

This challenge becomes significantly more acute in federated architectures, where each IdP is operated by a different organisation. Coordinated management of identity continuity across those entities is rarely feasible, especially in public sector or education scenarios with many independently governed domains. Even where brokers are used, the broker itself may change over time—leaving no stable sub, and no clean way to rebind existing users without administrative rework or manual correlation.

The OIDC model is technically compliant but operationally fragile in any environment where identities must persist across IdP changes. Systems that assume iss and sub are permanent risk both data fragmentation and serious user-level failures unless explicit correlation logic is introduced.

## OIDC Incompleteness

However, OIDC provides no built-in versioning, timestamp, or other metadata to help distinguish one identity issuance from another. It assumes the IdP is stable and authoritative, with no provisions for change.

## OIDC Misuse

From an OIDC compliance perspective, reusing the same iss while rotating all sub values **violates the spirit of the specification**, if not its letter. It constitutes a **non-conformant reuse** of an issuer URI and breaks downstream relying parties. However, this pattern is widespread in federated education and enterprise environments, and therefore must be treated as a real, even if non-compliant, risk.

While it is possible to define internal policies prohibiting such IdP reuse, federated systems operating at scale should not assume universal conformance.

Federation operators and brokers must either:

* enforce iss changes when IdPs change key issuance behaviour, or
* inject a custom claim (e.g. idp\_instance\_id, iss\_startdate, or similar) to version the IdP’s identity space.

Without this, systems must treat all unknown (iss, sub) pairs as ambiguous, and either:

* **disable just-in-time (JIT) provisioning** entirely,
* **require manual identity correlation by administrators**, or
* **implement heuristics or logging-based detection**, such as alerting when a known iss produces a flood of previously unseen sub values in a short period.

## Provisioning Failure Modes and Identity Ambiguity

Even when using compliant IdPs, systems must prepare for failure cases as described below.

### First-Time Identity with No Match

A new sign-in with an unknown (iss, sub) might be:

* a legitimate new user,
* a returning user after an IdP transition,
* or a recycled identity.

If JIT provisioning is enabled, this creates duplicate records. If disabled, access is denied until manual correlation.

### Recycled Email Addresses

Email addresses are often reused in schools, corporates, and shared environments. When a departing user's email is reassigned to someone new, and email is used for matching or pre-invitation, the new user may inherit the previous user’s record, permissions, or history. This results in silent data leaks, misapplied access, and serious trust violations.

**Recommendation:** matching on email is dangerous and to be avoided. The OIDC protocol’s locally unique sub is the intended user identifier.

### Changed sub with Stable iss

In some environments, institutions retain the same IdP endpoint (the iss) while replacing or restructuring the identity provider service.

This changes the sub for all users while keeping the iss unchanged.

The result is that all existing identities appear to vanish, and all new ones appear unrelated. A system cannot distinguish between new and migrated users without external correlation logic.

**Recommendation: use versioning.**

### Reused sub in a Different Context

Although unlikely in compliant IdPs, poorly designed identity systems may issue the same sub to multiple people over time. If iss and sub are naively treated as a global key, a new person may be treated as an existing one, inheriting roles or data.

**Recommendation:** use compliant identity providers. The OIDC protocol’s locally unique sub is the intended user identifier.

### No Issued Time or Subject Provenance

OpenID Connect does not include subject issuance time or lifecycle metadata. A new sub is indistinguishable from a reissued one. The system has no way to tell whether the identifier is trustworthy, recently created, or reassigned. This creates uncertainty in any model that treats sub as a durable identifier.

## Recommended Relying System Practices

The practices described below apply to the *relying system*—not to the identity provider or broker. While upstream systems may help by including additional claims (such as idp\_instance\_id), it is the responsibility of the relying party to manage ambiguity, protect continuity, and ensure security of its own user base.

### Avoid relying on email

Email should not be assumed present. The OIDC protocol does not guarantee its inclusion. The email scope must be explicitly requested by the relying party and approved by the user. Only iss (issuer) and sub (subject) are guaranteed.

### Avoid using email as a primary key

Where email is received, treat it as a mutable contact attribute, not a stable identity anchor. It is suitable for communication, not identity correlation.

### Track iss, sub

In the relying system, all identity associations must be keyed using the combination of iss:sub. This is the only durable and guaranteed identifier in OIDC.

### Track email if available

When email is present, record it as a contact attribute—useful for communications, invitations, and correlation, but not for identification.

### Permit changes to email

Emails often change—for correction, aliasing, family name changes, or reassignment. Permit changes to the email attribute within records found via iss:sub.

### Track activity metadata

Log first-seen and last-used timestamps for each identity. This enables later anomaly detection, continuity verification, and support diagnostics.

### Provide merge tooling

The system should support admin-driven manual correlation of identity records. This includes functionality to associate multiple iss:sub combinations with a single internal user where continuity is confirmed.

### Support identity rotation

Where IdP transitions are known in advance, support pre-registration or manual linking of new identities.

Within the relying system, consider adding an issReset flag to affected user records to temporarily relax identifier stability constraints.

### Treat new issuers as new users unless mapped

When a previously unseen iss value is encountered, treat it as a new and distinct issuer unless it has been explicitly mapped to a known identity space.

This includes cases where the same email is present but no continuity can be proven.

### Permit changes of iss and sub

When a user signs in with the same email but presents a different sub (and potentially a different iss), check if the issReset flag is present on an existing record. If so:

* Allow replacement of sub with the new value.
* Unset the issReset flag to prevent repeated overwrites.

### Halt suspicious merges

When a new sign-in presents a sub unknown to the system but a known email:

* If the existing user has issReset enabled, permit update the sub and lower flag.
* Otherwise:
  + If merge tooling exists, create the new user record as **disabled**, flagging for admin review.
  + If no tooling is available, deny the association and require the user to supply an alternative email address or submit to a manual resolution process.

Not all SaaS platforms will be willing or able to implement these practices. In such cases, the responsibility falls to the provider to explain how their system will avoid account crossover, identity confusion, or silent merges. Where continuity cannot be guaranteed, services must default to treating identities as distinct unless explicitly proven otherwise. This may involve disallowing reuse of email, enforcing manual identity verification, or maintaining parallel records until correlation is confirmed. Whatever the approach, the burden of clarity rests with the service.

## Why National or Sector Identifiers Must Not Appear in Identity Tokens

There is frequent interest—particularly in education, health, and government systems—in enriching identity tokens with national or sector-specific identifiers, such as a National Student Number (NSN), taxpayer ID, or health system ID. This practice appears convenient but constitutes a serious architectural flaw.

These identifiers are not identity attributes. They are contextual identifiers assigned by specific domains to support administrative or operational purposes. Their meaning, governance, and lifecycle are all tied to participation in that domain. Embedding them into identity tokens collapses critical boundaries between authentication and context, leading to several risks:

First, it breaks the separation of concerns. Identity providers are responsible for authenticating a person—not for asserting enrolment, funding, or medical records. Injecting domain-specific identifiers into tokens presumes that the IdP is authoritative across all domains, which it is not. Brokers should not attempt to inject such data either—they only amplify the design error.

Second, it invites incorrect assumptions about global uniqueness and persistence. Sector identifiers are often reassigned, merged, or revoked. They may not exist for all individuals, especially those new to the domain. Treating such identifiers as primary identity anchors leads to brittle systems and creates false continuity.

Third, it introduces unnecessary privacy and security exposure. National or sector identifiers are typically linked to sensitive records. Transmitting them in tokens—especially if opaque, unscoped, or passed between services—risks unauthorised use or correlation across domains.

Fourth, it undermines cross-domain and future-facing design. An identifier issued by the education system may be irrelevant to health or social services. Relying on sector identifiers in identity tokens hardcodes assumptions that restrict extensibility, interoperation, and reuse.

In conclusion, sector identifiers must be resolved only ***after*** identity is established—not embedded within it. They should be retrieved by the relying system, through authorised channels, under domain-specific access controls. Identity infrastructure must support modularity and information compartmentalisation. Identity tokens are not enrichment vessels. They are pointers—not payloads.

# Retrieving Sector Identifiers via Authorised Post-Authentication Calls

Once a user has authenticated via an identity provider (IdP), the relying system receives an access token—often accompanied by a profile or email scope, if granted. The token asserts only what the IdP is responsible for: a unique identity reference (via iss and sub) and optionally, personal attributes such as given\_name, family\_name, email, and birthdate. These are identity-level attributes, not sector-level identifiers, and they reflect the IdP’s understanding of the person—not their enrolment in any external system.

To retrieve sector-specific identifiers such as a National Student Number (NSN), a second, explicitly authorised call must be made—using the OAuth credentials of the relying system—to the domain system responsible for managing that identifier. This is the correct pattern. It cleanly separates responsibilities: the IdP proves *who*, and the domain system confirms *which* identifier applies to that person *in its context*.

The data required for such a match—typically given\_name, family\_name, and birthdate—may not be sufficient in all cases to uniquely identify a person. Therefore, it must be explicitly agreed between the relying system and the identifier service what minimum set of attributes will be used to attempt a match. These must be present and consistent in the user’s IdP profile at time of sign-in.

Email should generally be avoided as a matching attribute. While it may be present, it is not stable across time or organisations—especially for children or students who change or are in multiple schools. Email addresses may be institutionally reassigned, revoked, or misattributed. Using email as a secondary matching factor may seem attractive but often introduces false positives or data leakage risks. It should only be used if strong controls exist, and even then, treated as supplementary—not primary—evidence.

If a unique match cannot be found from the provided attributes, the identifier service should return no result. It must not return a best guess. Silent misbinding of a contextual identifier at scale is considered worse than no binding at all.

This model requires an additional HTTP/S call, but this is not a flaw. Making secure, OAuth-authorised REST calls between services is standard practice. It supports good design, traceability, and minimal trust between components. Critically, it avoids polluting the original authentication token with sector-specific data—preserving the modularity and neutrality of the IdP.

In short: do not expect the IdP to be responsible for sector identity. It is the responsibility of the consuming service to call the system that is. And that call must be deliberate, auditable, and based on clearly agreed matching rules.

With provisioning patterns covered, we now examine how users are placed into meaningful contexts within systems—defining their roles, memberships, and responsibilities.

# Contextualisation

While provisioning answers the question of who the user is—across systems and institutions—contextualisation answers where they belong in relation to specific domains, groups, or responsibilities. These contexts may originate in external organisational structures (such as schools, ministries, or teams), but must be mapped, validated, and governed within the system itself. Contextualisation determines how a known identity participates: what they are part of, what authority they hold, and what scope of action is appropriate.

This process is not merely about assigning permissions. It is about defining the user's presence in a context: which organisation they are joining, what role they occupy, what visibility they are entitled to, and what resources they can interact with. While some of this information may be known at the point of provisioning (e.g. invited to a specific group), it must always be treated as distinct from identity. Identity establishes the individual; contextualisation governs their place and purpose.

It is a design error to conflate identity with context. Identity is authenticated through protocols like OIDC. Context is determined by the system based on its own logic, data, and external references. The two must remain distinct.

## Modelling Group Membership and Roles

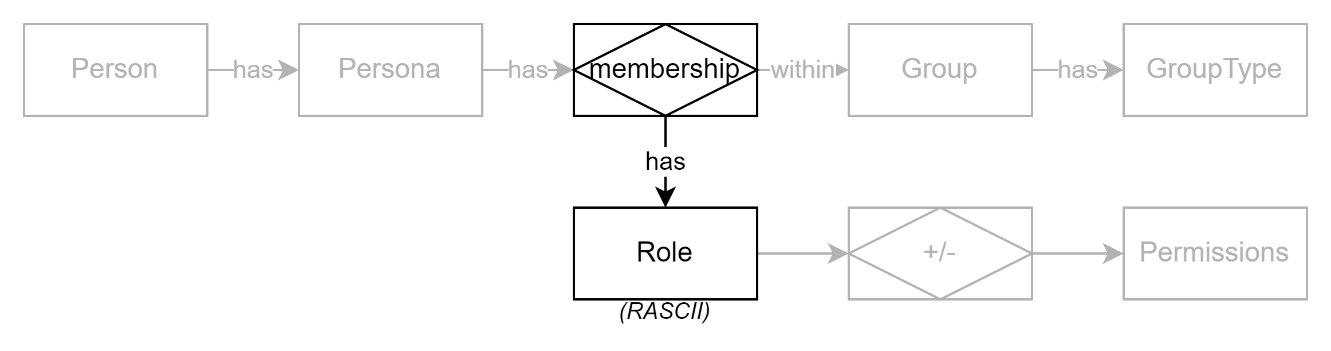


Figure 26: Membership and Roles

Membership and role are distinct concepts. Membership is the structural association—it means the person is part of the group. Role is a description of what that membership means in practice: what the person can do, what authority they hold, or how they participate. By modelling them separately, systems can express varied and overlapping relationships, such as someone who is an Observer in one group, a Lead in another, and a Contributor elsewhere. Combining these into a single notion risks confusion, reduces flexibility, and may lead to incorrect assumptions about permissions and responsibilities.

In most systems, a Persona exists meaningfully only in relationship to one or more groups. These groups may represent organisations, cohorts, teams, classes, or temporary collectives—and they define the boundaries within which a user acts, is recognised, or holds responsibility.

Group structures vary widely by domain. In organisational systems, they may follow a nested hierarchy—such as departments within branches within a company. In education, users often belong to multiple groupings, such as a class group, a year cohort, a sports team, and a pastoral care group. These may be defined structurally (e.g. “Class 10A” within “School XYZ”) or logically (e.g. “All students aged 14–16”). In communication systems, such as chat platforms, groups may be ad hoc—formed around a thread, event, or shared resource.

A single Persona is therefore typically associated with multiple groups, some in nested sequence (e.g. Department → Faculty → Institution), and others in parallel (e.g. enrolled courses, clubs, responsibilities). Groups may also evolve independently over time, with no guarantee of stable structure or naming.

Participation in a group is modelled as a membership, which may carry a role—such as Creator, Member, Contributor, Observer, or Lead. These roles describe the person’s position or authority within that group, and are separate from system-wide roles like Administrator or Auditor.

This relationship—between a person and a group, with an associated role—is foundational to system logic. It informs access control, notifications, visibility, delegation, and audit.

## The Misuse of Identity Providers for Group and Role Context

Over time, a significant and persistent architectural error has emerged: the misuse of identity providers to supply contextual group and role information. This problem has deep historical roots—originating in the LAN-era use of Microsoft Active Directory (AD), where organisational units (OUs) and security groups were often conflated with application-level roles and permissions.

In many organisations, the AD structure became the de facto source of both identity and access. Teams would create OUs such as AccountingLeader, AccountingMember, or SalesViewer, and then assign permissions in applications based on group membership within AD. This tightly coupled authentication, authorisation, and organisational logic, creating fragile and overlapping domains of responsibility.

Even as the architectural landscape shifted—moving away from OUs and toward cloud-native identity providers—the problem persisted. Modern identity platforms like Azure AD continue to blur the line by supporting custom role claims, group claims, and application-specific scopes in tokens. What was once an overreach of directory structure has returned under new labels, now carried invisibly within signed tokens and reused across environments.

At best, these claims can function as **logical tags**—informative metadata that downstream systems may choose to interpret. But the moment an application begins **trusting** these claims for group placement, role assignment, or access control, the IdP becomes entangled in business logic it cannot understand and cannot validate. This undermines system boundaries and creates brittle, untraceable dependencies between authentication infrastructure and application logic.

An identity provider should have **no downstream dependencies**. It is responsible only for asserting that the user is who they say they are—not for deciding what that user can do in any particular system. Group membership, role assignment, and context must be governed by the application, informed by its own rules, data, and trust relationships—not assumed from token claims issued in another domain.

Hinting, while seemingly convenient, can become insidious. A “hint” from an IdP is often treated as authoritative by the consuming system, especially when automation or time pressure leads to skipping validation. This creates hard-to-detect data leakage, over-permissioning, and operational ambiguity when roles change but token hints persist. These outcomes may go unnoticed until a breach, audit, or failure exposes the underlying assumptions.

The correct design is to treat identity and context as separate concerns. Identity providers authenticate the user; contextualisation systems and the application itself determine placement and permissions. This separation not only improves system integrity, but also allows each component to evolve independently.

## On Token Enrichment and Architectural Overreach

It is tempting, especially in federated environments, to solve identity problems by enriching tokens with additional attributes such as sector identifiers, school codes, or national IDs. This temptation is strongest when data consumers face integration delays or lack access to trusted resolution services. However, this approach is a mistake.

Brokers or identity middleware that insert sector identifiers into tokens breach architectural boundaries and obscure responsibility. Identity Providers (IdPs) assert who a user is within their own system. They are not responsible for providing sector-level identifiers, and doing so creates dangerous coupling between unrelated domains. It also removes the relying party's ability to verify, audit, or govern how those identifiers were determined.

The risk is compounded when token enrichment uses inferred data—such as matching a user’s name and date of birth to an external database—to embed identifiers like the NSN. These matches may be ambiguous, temporary, or incorrect. Once embedded in a token, however, they are treated as authoritative and immutable. If a later correction is made, downstream systems may not detect or accept it, leading to duplication, mis-association, or denial of access.

Instead, sector or national identifiers must be retrieved by the relying system through explicit, authenticated service calls to a trusted resolution source. This preserves control, ensures traceability, and allows identifier lookup to occur only when required. It also avoids burdening identity flows with cross-domain logic, and maintains separation between identity authentication and attribute acquisition.

Identity providers should not be asked to "enrich" identity tokens with data they do not own. Systems should be designed to call dedicated services to retrieve authoritative identifiers, based on a clear and well-scoped interface. The token should remain clean—asserting only identity, not inferred or foreign attributes.

## Source and Timing of Contextualisation

As systems move beyond initial identity establishment via OIDC tokens, they must treat contextual placement as a separate and deliberate responsibility. Identity tokens may include source system (i.e., the IdP’s) group or role claims, but these are often stale, misaligned with the relying system’s structure. While such claims might appear to be convenient hints, relying on them as authoritative information creates brittle dependencies and risks misassigning users.

Contextualisation must be driven by system-side rules and data, not inferred from third-party claims embedded in identity flows. While identity providers assert who a person is within their own scope, they are unaware of broader sectoral identity relationships, role assignments, or evolving group memberships maintained elsewhere. Therefore, even when identity has been established and optionally added to from external sector identity lookup services, contextual placement must still be performed by the relying system according to its own domain logic and policies. The combination of identity data from the IdP and additional reference data from trusted sector services enables accurate contextualisation, but only if the relying system remains the authoritative decision-maker.

Additionally, contextualisation can occur at different stages in the user lifecycle. The stage at which it occurs has implications for system accuracy, flexibility, and security:

**Pre-Provisioning Contextualisation** occurs prior to the user being created in the system. It depends on the target group or context already existing, so that it can be referenced during the provisioning event—such as in an invitation that assigns the invitee to a specific role within a specific group. This includes workflows like bulk uploads, SCIM-based data feeds, or invitation-based flows. These offer a high degree of control, but also carry dependency on upstream data quality and the correctness of the contextual mapping asserted at the time of assignment.

**Provision-Time Contextualisation** takes place during the provisioning flow itself—typically when a user signs in for the first time and a group or workspace is created then and there to associate the user to it. For example, a self-registration flow might assign the user as administrator of their own new organisation or project. This approach is efficient for isolated or tenant-based contexts but risks granting elevated permissions if not clearly bounded.

**Post-Provisioning Contextualisation** refers to the assignment or adjustment of a user's contextual relationships after their identity is confirmed and a user account has been created in the system. This may happen manually—such as when administrators review newly created users and assign them to relevant groups—or automatically, by using trusted data sources to map users into appropriate contexts. Automated contextualisation offers the greatest flexibility and responsiveness to real-world changes but must be implemented with clear rules, robust audit trails, and protections against invalid mappings, accidental privilege escalation, or misuse of external data.

# When to Enrich tokens

While Enriching tokens is warned against, it can’t always be avoided.

If the system to invoke is a SaaS that may be replaced over time, or doesn’t have APIs with sufficient capability to be used as designed, then there may be no choice but accept a sub-optimal outcome, solve this via the IdP broker having sufficient logic to query and cache Ids.

We conclude by summarising the key architectural recommendations and reiterating the importance of clean separation between identity, context, and enrichment.

# Conclusion

Temptation to embed sector or national identifiers directly into the identity token is understandable—but flawed. While it might seem efficient to use the IdP as a central point of enrichment, this introduces responsibilities that can’t be met and introduces dangerous coupling between systems with different scopes and obligations.

Identity providers should remain focused solely on authentication. Sector identifiers should be sourced from services that specialise in identity relationships. Contextual placement should be governed by the relying system itself.

OIDC remains a mature and robust protocol, and its adoption as the standard for federated authentication is appropriate. However, like all protocols, its correct use depends on understanding its limits. Identity tokens should not be treated as containers for everything. Systems should design with explicit separations: the IdP provides a local identity; an identity enrichment service may provide additional identifiers; and the application governs context.

Where IdP migration or instability is expected, systems may need to maintain auxiliary metadata (such as tracking issuer version, first seen dates, and last seen dates) and support continuity mechanisms for mapping historical identities to current ones. But this too is best handled by the relying system—not by attempting to coerce the broker into managing information it cannot verify.

Use each system for what it is good at, and maintain clean boundaries between identity, enrichment, and context.

Appendices

Appendix A - Document Information

Authors & Collaborators

* Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft

### Images

[Figure 1: Different types of attributes 9](#_Toc201899523)

[Figure 2: Unmodelled Systems 10](#_Toc201899524)

[Figure 3: Each System has its own Users. 12](#_Toc201899525)

[Figure 4: Access To is distinct from Access Within a system 13](#_Toc201899526)

[Figure 5: Authorisation Management and Implementation is Local 13](#_Toc201899527)

[Figure 6: retrieve identity and context from separate, appropriate, sources 14](#_Toc201899528)

### Tables

[Table 1: TODO Table **Error! Bookmark not defined.**](#_Toc145048484)

[Table 2: TODO Table 2 **Error! Bookmark not defined.**](#_Toc145048485)

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Acronyms

API

: [Application Programming Interface](#Term_ApplicationProgrammingInterface).

DDD

: Domain Driven Design

GUI

: [Graphical User Interface](#Term_ApplicationProgrammingInterface). A form of [UI](#Acronym_UI).

ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. [IT](#Acronym_IT) is a subset of ICT.

IT

: acronym for Information, using Technology to automate and facilitate its management.

SaaP

: Software as a Product. Installed on a PaaS or IaaS. It Is installed. Contrast with SaaS. Not necessarily multi-account/tenanted.

SaaS

: Software as a Service. The software is multi-tenanted, and is not installed by account.

PaaS

: Platform as a Service. The ability to host and run arbitrary code. A SaaS that provides configuration and scripting (e.g. Salesforce) remains a programmable SaaS.

aPaaS

: Gartner’s term for PSaaS. However, PaaS is an inaccurate starting point (it is a SaaS). Consider *asSaaS*.

asSaaS

: Application supporting Software as a Service. A SaaS that permits sending up custom code conceptually organised as “apps”. Gartner refers to them as aPaaS.

iPaaS

: integration platform as a service.

UI

: User Interface. Contrast with [API](#Acronym_API).

### Terms

Refer to the project’s Glossary.

Application Programming Interface

: an Interface provided for other systems to invoke (as opposed to User Interfaces).

Capability

: a capability is what an organisation or system must be able to achieve to meet its goals. Each capability belongs to a domain and is realised through one or more functions that, together, deliver the intended outcome within that area of concern.

Cloud Services

: (e.g. Azure, AWS) offered by Cloud Service Providers (Microsoft, Amazon). Provides CIPS, along with st

Domain

: a domain is a defined area of knowledge, responsibility, or activity within an organisation or system. It groups related capabilities, entities, and functions that collectively serve a common purpose. Each capability belongs to a domain, and each function operates within one.

Entity

: an entity is a core object of interest within a domain, usually representing a person, place, thing, or event that holds information and can change over time, such as a Student, School, or Enrolment.

Function

: a function is a specific task or operation performed by a system, process, or person. Functions work together to enable a capability to be carried out. Each function operates within a domain and supports the delivery of one or more capabilities.

Person

: a physical person, who has one or more Personas. Not necessarily a system User.

Persona

: a facet that a Person presents to a Group of some kind.

Quality

: a quality is a measurable or observable attribute of a system or outcome that indicates how well it meets expectations. Examples include reliability, usability, and performance. Refer to the ISO-25000 SQuaRE series of standards.

User

: a human user of a system via its UIs.

User Interface

: a system interface intended for use by system users. Most computer system UIs are Graphics User Interfaces ([GUI](#Acronym_GUI)) or Text/Console User Interfaces (TUI).

Appendix B - Document Information

* **OIDC Core Spec** – https://openid.net/specs/openid-connect-core-1\_0.html
* **Microsoft: Best Practices for Token Validation** – <https://learn.microsoft.com/en-us/azure/active-directory/develop/access-tokens#validating-tokens>
* **Google Workspace Identity Platform Docs** – for understanding GCP token claim limitations
* **NIST SP 800-63C** – *Digital Identity Guidelines: Federation and Assertions*
* **NZ Government Digital Identity Trust Framework (Te Tari Taiwhenua)**
* **RFC 7643/7644 (SCIM 2.0)** – for provisioning clarity
* **NZ Government Privacy Impact Assessments (PIA) and Security Risk Assessments (SRA) Guides**
* **Te Rito Guidance & SMS Standards** – where available

Appendix C – Use Sub to disambiguate users

While email is conceptually sufficiently unique to disambiguate users, emails are insufficient for this purpose in long living, large user base, systems such as government systems for education. While sufficient to disambiguate two parallel learners with the same name ([john.smith@schoolA.org](mailto:john.smith@schoolA.org) versus [john.smith@schoolB.org](mailto:john.smith@schoolB.org), or even [john1.smith@schoolA.org](mailto:john1.smith@schoolA.org) and [john2.smith@schoolA.org](mailto:john2.smith@schoolA.org)), emails may be reused over time ([john.smith@schooA.org](mailto:john.smith@schooA.org) in 2010, and john [smith@schoolA.org](mailto:smith@schoolA.org) in 2020).

For this, the OIDC specification provides the solution:

* **OIDC Core 1.0 Specification – Section 2:  
  Subject Identifier**  
  A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client.
* **Section 5.1 – Standard Claims:  
  sub**  
  *Subject - Identifier for the End-User at the Issuer.*  
  It is a **locally unique and never reassigned identifier** within the Issuer. It **must not be personally identifiable information (PII)** such as an email address.

However, a poorly implemented broker *can* (incorrectly) use email as the **sub**.  
   
If that is the case, and email addresses are reassigned (e.g. two John Smiths reusing jsmith@school.nz), the sub becomes unstable—violating OIDC intent and making identity correlation unreliable.

Options to fix this include:

* Return sub to not being an email, **and** either make it:
  + Only Locally unique, while instructing clients use a composite key (iss+sub)  
    to find identity records associated to a system user, **or**
  + Be globally unique (a UUID is satisfactory), while instructing clients use a composite key (iss+sub) *anyway* to find identity records associated to a system user (the addition of the iss prefix provides higher entropy), **or**
* Instruct brokered IdPs use a valid from/to (least preferred option, due to change management complexity).

Appendix D - Guiding Principles

The importance of systems that manage identity and participation must be designed according to clearly articulated principles. Without these, convenience, legacy patterns, or vendor defaults will shape decisions in ways that undermine long-term clarity, maintainability, and security.

Designing systems without clear principles leads to fragile, opaque behaviours that fail under change. Principled systems, by contrast, are resilient, traceable, and respectful of domain separation.

### Identity is not Context

Identity protocols (such as OIDC) exist to assert who a person is—not where they belong or what they are allowed to do. These responsibilities must be separated in system design.

### Contextual Placement Must Be System-Governed

Group membership, role assignment, and contextual positioning must be governed by the application, using its own rules, data sources, and responsibilities. This maintains domain clarity and auditability.

### Sector Identifiers Belong in Sector Systems

If a national or sector identifier is needed, it should be sourced through a verified backchannel from a sector-managed service. This preserves authoritative boundaries and reduces risk.

### Design for IdP Independence

Systems should expect identity providers to change. To retain continuity, they must track auxiliary metadata (e.g. iss version, first/last seen dates) and support mapping of historical identities.

### Avoid Token Enrichment by Brokers

Brokers and identity providers should not act as data enrichment engines. Their tokens should assert identity only. Adding sector identifiers or contextual claims creates coupling and erodes trust boundaries.

**OpenID Connect Overview and Limitations**

OpenID Connect (OIDC) is a widely adopted identity protocol that enables federated authentication across systems. It allows a user to prove who they are by authenticating through a trusted identity provider (IdP), which then issues a token that the relying system can verify.

OIDC builds on OAuth 2.0, adding a structured identity layer designed for interoperability, security, and extensibility. It standardises key concepts such as issuers (iss), subjects (sub), scopes, and claims—allowing applications to receive cryptographically signed tokens containing identity assertions. This makes OIDC a sound foundation for asserting *who* a user is—but not *what* they can do, *what roles* they hold, or *where* they belong. *Those aspects lie outside its scope, and must be determined by the relying system.*

While OIDC is well-suited to browser-based authentication and single sign-on across services, it is not a complete solution for all identity-related concerns. It does not model identity continuity across identity provider changes, does not support contextual placement, and should not be extended beyond its intended scope. Attempting to retrofit OIDC tokens to carry enriched attributes or external identifiers undermines system clarity and invites long-term fragility.

*It is essential to treat OIDC as one component—responsible for verifying who the user is—not as a universal substrate for context, continuity, or domain-specific logic.*

1. A relatively common error of conflating User and Account into a single entity is to be avoided, as this shortcut undermines clarity and later flexibility. [↑](#footnote-ref-2)