

Whose Data Is It Anyway? Designing a User-Centric and Sovereign Architecture for Media Consumption in the European Digital Space

Michiel Van de Velde

Professional Bachelor Electronics-ICT, specialisation ICT

Academic year 2025-2026

Internal Supervisor Bart Delrue

External Supervisors Pieter Colpaert and Ben de Meester

Abstract. Streaming platforms generate vast amounts of behavioural data, yet users retain little control over how this data is collected, stored, or shared. Current architectures centralise data within individual platforms, fragmenting the user's digital footprint across services while limiting transparency and sovereignty. This bachelor's thesis investigates which key design decisions must be made when building media behaviour profiles across multiple media services in a user-centric way. The proposed architecture is grounded in the Solid ecosystem and Trustflows framework, combining decentralised personal data storage with policy-driven access control. Design decisions are guided by five core principles: European data sovereignty, open-source implementation, adherence to open standards from the W3C, interoperability across autonomous stakeholders, and the formulation of competency questions that define what the system must be capable of answering. The research is informed by an expert questionnaire completed by a practitioner from the Flemish media sector and by discussions with researchers at imec-IDLab, and culminates in a conceptual architecture, a proof-of-concept implementation, and a structured set of design decisions for cross-service media behaviour profiling. The expert's assessment confirms a central finding: ecosystem adoption, not technical feasibility, is the primary barrier to realising this architecture in practice. The results demonstrate that a user-centric, interoperable, and legally compliant media data architecture is both theoretically sound and practically achievable within the European digital landscape.

Preface

This bachelor's thesis grew out of a long-standing interest in open-source software and digital sovereignty, and the conviction that the web does not have to be as centralised as it currently is. That interest took a more concrete direction during the course Web Topics Advanced, where a visit to imec-IDLab introduced me to the Solid ecosystem and the research being done there around personal data management. The connection between those technologies and the questions I had been asking about data ownership felt immediate, and when the opportunity arose to write my thesis at IDLab, I did not hesitate.

The work was conducted as a bachelor's thesis at the Odisee Hogeschool in Ghent, in collaboration with imec-IDLab in Ghent. Working within IDLab gave me access to a whole new world of ongoing research on personal data management, the Solid ecosystem, and Trustflows, all of which shaped this thesis from its earliest stages.

I want to thank prof. dr. Pieter Colpaert for welcoming me into his research group and for proposing the topic that became this thesis. Working with personal data management, the Solid ecosystem, and linked data quickly made clear how significant the research being done at IDLab is, and how much thought has gone into building a more human-centric web. During my internship Pieter consistently made time for feedback despite an exceptionally busy schedule, and he approached every idea I brought with genuine openness, never dismissing a suggestion simply because it came from a bachelor student. That attitude made a real difference to how I experienced the research process and made the work genuinely enjoyable. I also want to thank Bart Delrue for the day-to-day support and for the many conversations and follow-up meetings. I owe particular thanks to dr. Ben De Meester for helping me formulate the competency questions and for connecting me with the right people in the Flemish media industry. Through that connection a written questionnaire was completed by a practitioner at MediaNet Vlaanderen, whose perspective grounded the technical work in the realities of the sector and sharpened the design decisions documented here.

Finally, I thank my family and friends for their patience during the weeks/months when this thesis required most of my attention.

Michiel Van de Velde
Ghent, May 2026

Table of contents

Preface

List of figures

List of tables

List of code fragments

List of abbreviations

Glossary

1. Literature Study

1. Media Behaviour Profiles: Definition and Privacy Implications
2. Towards a User-Centric Architecture: Core Principles
3. The Solid Ecosystem as a Technical Foundation
4. Requirements for a User-Centric Architecture
 1. Personal Data Store
 2. Identity and Authentication
 3. Policy Enforcement
 4. Aggregation
 5. Auditability
5. Dataspace Integration and Eventual Interoperability
6. Stakeholder Considerations
7. Limitations and Challenges of User-Centric Media Profile Architectures

2. Architecture and Design Decisions

1. Competency Questions
2. Personal Data Store
3. Data Partitioning
4. Analytical Storage and Binary Assets
5. Identity Provider
6. Authorisation Engine
7. Semantic Data Model
8. Event Bus
9. Aggregation Processor
10. Inference Service
11. Policy Layer
12. Audit Layer

3. Implementation

1. 1. Define the Slice Schema
2. 2. Bootstrap the Pod and Register Slices
3. 3. Authenticate as a Service
4. 4. Write a Media Event
5. 5. The Change Processor Reacts
6. 6. Update Counters with Delete-Then-Insert
7. 7. Query the Enriched Profile
8. 8. Cross-Service Identity Linking
9. 10. User-Validated Behavioural Insights
10. Import Rate Limiting

4. Evaluation

1. Competency Question Assessment
2. Expert Feedback
3. Requirement Area Coverage

5. Conclusion

6. References

7. Use of Generative AI

List of figures

Fig. 1: Individual data fragments, each seemingly harmless in isolation, combine to form a detailed and revealing profile of a person. Own work, created with AI assistance.

Fig. 2: Organisational structure tends to mirror system design. Adapted from Sketchplanations, “Conway’s Law” [7].

Fig. 3: Key principles of a user-centric media profile architecture. Own work.

Fig. 4: Solid pod resources accessed via HTTP verbs, showing read, write, and append operations by different actor types. Own work.

Fig. 5: The user’s pod partitioned into three containers at different sensitivity levels, with access rules per partition. Own work.

Fig. 6: WebID and Solid-OIDC enabling a user to authenticate across independent services using a single user-controlled identity document. Own work.

Fig. 7: The ODRL policy layer intercepts access requests and evaluates them against user-defined permissions, prohibitions, and duties. Own work.

Fig. 8: The aggregation agent reads only new events from each service partition using LDES, derives a unified cross-service profile, and writes it back to the pod. Own work.

Fig. 9: The five architectural building blocks and their interactions, showing the full data flow from ingestion through aggregation to consumption. Own work, created with AI assistance.

Fig. 10: European Strategy for Data: a common European data space enabling trusted and interoperable data sharing across sectors. Adapted from European Commission, *A European Strategy for Data* [20].

Fig. 11: The five requirement areas that any valid user-centric media behaviour profile architecture must address, as established in the literature study. Own work.

Fig. 12: System architecture of the prototype, showing the Kvasir User Pod, identity and authentication layer, aggregation pipeline, logging layer, and policy enforcement as future work. Own work.

Fig. 13: A committed change report in Kvasir showing the six RDF triples written to the music-tracker slice for a single listen event. Own work.

Fig. 14: A GraphQL query against the video-tracker slice returning the six most recent watch events, executed through Kvasir’s GraphiQL interface. Own work.

List of tables

Table 1: Traceability between architectural roles, prototype choices, and competency questions. Own work.

Table 2: Prototype deviations from a purely Solid-native implementation, with the reason for each deviation. Own work.

Table 3: Assessment of the prototype against the three competency questions, showing evidence and limitations for each. Own work.

List of code fragments

Lst. 1: A media listen event represented as an RDF triple in Turtle syntax, using schema.org vocabulary. Own work.

Lst. 2: A listen event and a watch event from two different platforms, represented in the same RDF structure using schema.org vocabulary. Own work.

Lst. 3: A WAC policy granting Alice read access and Bob full control over a resource in Bob's pod, using URI-identified agents and explicit permission modes. Own work.

Lst. 4: An ACP policy granting read access to a resource only when the request comes from a specific agent using a specific client application, combining both conditions through a matcher. Own work.

Lst. 5: GraphQL SDL schema for the `music-tracker` slice, combining schema.org and Music Ontology vocabularies. Own work.

Lst. 6: GraphQL SDL schema for the `video-tracker` slice, using schema.org vocabulary throughout. Own work.

Lst. 7: GraphQL SDL schema for the `cross-service-profile` slice, introducing the `csp_Creator` type with user-confirmed identity links, aggregated interaction statistics, and the `csp_UserInsight` type for user-validated behavioural assertions. Own work.

Lst. 8: Bootstrap configuration declaring OpenFGA relationships for the `music-tracker-client` service account. Own work.

Lst. 9: A `schema:ListenAction` event in JSON-LD format, linking a user agent to a MusicBrainz recording URI with a deterministic subject URI. Own work.

Lst. 10: Delete-then-insert pattern for updating `schema:InteractionCounter` triples in Kvasir. Own work.

Lst. 11: GraphQL query against the `music-tracker` slice returning artists ordered by interaction count. Own work.

List of abbreviations

Abbreviation	Full form
ACP	Access Control Policy
API	Application Programming Interface
CSS	Community Solid Server
EU	European Union
GDPR	General Data Protection Regulation
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
IDLab	Internet and Data Lab
JSON-LD	JavaScript Object Notation for Linked Data
JVM	Java Virtual Machine
LDP	Linked Data Platform
LDES	Linked Data Event Streams
ODRL	Open Digital Rights Language
OIDC	OpenID Connect
OAuth	Open Authorization
RDF	Resource Description Framework
S3	Simple Storage Service
SDL	Schema Definition Language
SPARQL	SPARQL Protocol and RDF Query Language
SSE	Server-Sent Events
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAC	Web Access Control
WebID	Web Identity

Glossary

Media behaviour data Data generated when a user interacts with media services, such as playing a video, pausing content, skipping a song, listening to music, or entering a search query.

Media behaviour profile A structured representation derived from media behaviour data over time, including patterns such as preferred genres, typical moments of use, recurring interests, or cross-platform consumption habits.

Personal data Any information relating to an identified or identifiable natural person. Under the GDPR, this includes not only directly identifying information but also behavioural data that can be linked to an individual. In this thesis, media behaviour data recorded by streaming services constitutes personal data because it is tied to a specific user account and can reveal preferences, habits, and identity.

User-centric architecture An architecture in which the user remains the central point of control over where personal data is stored, who may access it, and under which conditions it may be used.

Data sovereignty The ability of a user or organisation to exercise meaningful control over data, including access, storage, sharing, and governance.

JSON-LD A serialisation format for RDF data that is valid JSON and uses a `@context` field to map JSON keys to semantic vocabulary terms.

Kvasir A personal data broker developed at imec-IDLab that exposes a Solid-compatible storage API. Kvasir organises data into schema-governed partitions called slices and uses ClickHouse for analytical storage and Redpanda for event-based change notifications.

Music Ontology An RDF vocabulary for describing music-related concepts, including tracks, recordings, artists, and listening events.

OpenFGA An open-source relationship-based authorisation engine that models permissions as typed relationships between objects, such as a user having a writer relationship on a specific slice.

Redpanda A Kafka-compatible distributed event streaming platform used in the prototype to publish change events from Kvasir slices to the aggregation processor.

Slice A schema-governed partition within a Kvasir pod. Each slice holds a specific type of data, such as music listening events or video watch events, and has its own access rules.

Solid An ecosystem of specifications and tools for decentralised personal data management, where users store data in personal online data stores.

Solid pod A decentralised personal data store controlled by the user, in which data can be stored, accessed, and shared through web standards.

Data vault A user-controlled storage environment for personal data. In this thesis, the term is closely related to the concept of a Solid pod.

WebID A globally unique web-based identifier used to identify a user or agent in the Solid ecosystem.

Solid-OIDC The authentication mechanism used in Solid, based on OpenID

Connect, that allows applications to verify a user’s identity and request access to pod resources.

RDF A standard data model that represents information as triples consisting of a subject, predicate, and object.

Triple A single RDF statement consisting of a subject, predicate, and object.

URI A Uniform Resource Identifier used to uniquely identify resources, concepts, users, or relationships on the web.

Linked Data A method for publishing and connecting structured data on the web using shared identifiers and standard data models.

Web Access Control An access control mechanism used in Solid that defines permissions for agents, groups, applications, and resources.

Access Control Policy A more flexible Solid access control mechanism that can express detailed access rules and conditions.

ODRL The Open Digital Rights Language, used to express permissions, prohibitions, and duties related to data usage.

Policy engine A component that evaluates access and usage policies before allowing or denying a data request.

Aggregation agent A user-authorised component that combines media behaviour data from multiple services and produces an aggregated cross-service profile.

Change processor A component in the aggregation pipeline that consumes change events from Redpanda, derives a unified cross-service profile, and writes it back to the Kvasir pod.

Audit log A record of data access or usage, showing who accessed which data, when, and for what stated purpose.

LDES Linked Data Event Streams, a technique for publishing and processing timestamped event data incrementally.

Data space A governed ecosystem in which different organisations can share and use data according to common rules, standards, and trust mechanisms.

Interoperability The ability of independent systems or services to exchange, interpret, and reuse data consistently.

Mosaic effect The privacy risk that separate pieces of seemingly harmless data can reveal sensitive information when combined.

GraphQL A query language and API specification that allows clients to request exactly the data they need from a server. In this thesis, Kvasir exposes each slice through a GraphQL API generated from the slice’s SDL schema.

Keycloak An open-source identity and access management server used in the prototype to issue authentication tokens for both user-facing clients and background service accounts.

MusicBrainz An open music encyclopedia that provides structured metadata for recordings, releases, and artists, identified by stable URIs. The prototype uses the MusicBrainz API to enrich listen events with album and artist metadata.

Ollama A tool for running large language models locally. In the prototype, Ollama is used by the inference service to generate recommendations from the cross-service profile without transmitting behavioural data to external APIs.

schema.org A shared vocabulary maintained by a community of major search engines and web publishers, providing standardised terms for describing entities such as actions, persons, and creative works. In this thesis, schema.org terms are used to represent media interaction events in a way that is interoperable across services.

Trustflows A framework for building trustworthy data flows that are interoperable, legally compliant, and user-centric. In this thesis, Trustflows provides the governance model that complements Solid’s storage and access control capabilities, particularly through ODRL-based usage conditions.

Introduction

Streaming has become a central part of daily digital life. In 2024, statistics from the Flemish government indicate that 41% of residents in the Flemish Region watch series and 40% listen to music at least once per week. This widespread streaming activity reflects a broader trend of growing digital engagement: 94% of residents aged 18 and older use the internet at least once per week, with 96% reporting weekly use of a smartphone or mobile phone and 83% using a computer, laptop, or tablet on a weekly basis [1].

As digital engagement continues to grow, so does the volume of data generated by users. This growth is accompanied by rising concerns about how such data is managed. Studies from SolidLab, an imec research initiative investigating user-centric personal data management, indicate that both trust and perceptions of transparent data usage remain critically low. For instance, survey results show that a significant portion of the Flemish population believes that companies are insufficiently transparent about how personal data is used, while approximately 70% of users report a lack of control over their own data [2].

This tension is further deepened by the structure of today's data economy. It is often argued that "data is the new gold." However, this analogy is fundamentally flawed. Unlike gold, data can be shared simultaneously with multiple parties without being depleted, a property that makes concentration of data control arguably wasteful from a societal perspective. Despite this, data flows predominantly toward a relatively small number of organizations, particularly large tech companies based in the United States and China. These companies manage data within their own infrastructure and derive significant economic value from it. This concentration has implications not only for privacy, but also for **data sovereignty and national security**, while simultaneously acting as a barrier to innovation: smaller and emerging companies are frequently excluded from the data ecosystems they need in order to compete on equal footing [3].

The European Union has recognized data concentration as a structural problem, responding with substantial investment through initiatives such as the Digital Europe Programme [4] and GAIA-X [5]. These programs envision a human-centric data economy in which individuals retain meaningful control over their personal data. The gap between policy ambition and practical implementation remains significant, particularly in the media domain, where behavioural data is continuously generated but governance frameworks for user-centric control are still immature. Two promising building blocks for such an architecture are the Solid ecosystem and Trustflows. Solid enables users to store their personal data in decentralized online data stores called pods that they manage and control, allowing them to grant or revoke access to applications as needed. Trustflows complements this by providing a framework for building trustworthy data flows that are interoperable, legally compliant, and user-centric. Together, they provide a foundation for rethinking how personal media data is stored, shared, and governed.

This bachelor's thesis, conducted at imec-IDLab, addresses the following research question: **Which key design decisions need to be made when building media behaviour profiles across multiple media services?**

To answer this question, competency questions are first formulated to define the func-

tional requirements of the proposed architecture and assessed through an expert questionnaire completed by a practitioner from the Flemish media sector and through discussions with researchers at imec-IDLab. A literature review of relevant technologies and open standards informs the design choices throughout. The findings culminate in both a designed architecture and a working prototype, demonstrating that user-centric management of personal media data is not only theoretically sound but also practically achievable.

The outcome of this research is a conceptual architecture, a proof-of-concept implementation, and a structured set of design decisions for cross-service media behaviour profiling.

1. Literature Study

The central research question of this thesis asks which design decisions are required when building media behaviour profiles across multiple services in a user-centric way. Answering this question requires a clear understanding of what such profiles are, why current approaches fall short, and which technologies and architectural principles can support a more user-centric solution. The sections that follow first define media behaviour data and examine its privacy implications, then establish the core principles of a user-centric architecture, introduce the Solid ecosystem as a technical foundation, describe the individual architectural building blocks in detail, and finally situate the proposed approach within the broader European data space landscape while reflecting on the stakeholders it affects and the limitations it inherits.

1.1. Media Behaviour Profiles: Definition and Privacy Implications

In this thesis, media behaviour profiles refers to profiles created when a user interacts with a media platform. This includes direct actions, such as playing a video, pausing content, skipping a song, or entering a search query. It also includes patterns derived from these actions over time, such as favourite genres, typical moments of use, or preferences across different platforms.

This distinction is important because raw events and derived profiles are not the same. A single play, pause, or search action may seem harmless, but when many of these events are combined, they can reveal detailed information about a person. For example, long-term media behaviour may expose routines, interests, political views, emotional states, health-related concerns, or cultural identity. This is often referred to as the mosaic effect: separate pieces of data may appear insignificant, but together they can form a sensitive and revealing profile.

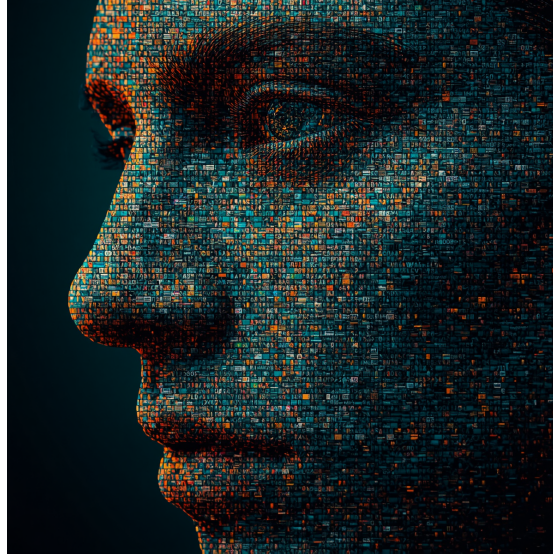


Fig. 1: Individual data fragments, each seemingly harmless in isolation, combine to form a detailed and revealing profile of a person.

A well-known example outside the media domain is Target’s reported use of shopping data to predict whether customers were pregnant and to target them with related advertisements [6]. Similar risks exist in media environments. Viewing history, listening behaviour, and search activity can reveal sensitive personal information, even when the original data appears ordinary. Media behaviour data should not be treated as low-risk simply because individual events seem harmless. The privacy risk is increased by the fact that most media platforms store this data centrally. Platforms often collect, analyse, and retain behavioural data within their own infrastructure, while users have limited insight into what is stored, how long it is kept, how it is used, or with whom it is shared. This creates an imbalance: platforms can build detailed profiles of users, while users have little control over those profiles. In this context, user control means more than the ability to access or delete data. It includes the ability to determine where data is stored, which actors may access it, for which purposes it may be used, how long it may be retained, and whether derived profiles may be shared with others.

Centralised storage can also create security and governance risks. Large behavioural datasets may become attractive targets for breaches, legal requests, and secondary uses beyond the user’s original expectations. Once behavioural data has been used to train models or create derived profiles, it may also be difficult to fully remove its influence, even if the original data is deleted. This creates tension with rights such as data erasure under the GDPR [8].

However, decentralised storage does not automatically remove these risks. The MyData initiative helps explain why. MyData is a human-centric data movement that argues that individuals should not only have formal legal rights over personal data,

but also practical tools to access, control, reuse, and benefit from data about themselves. It promotes a shift away from platform-controlled data silos toward models in which the individual becomes a meaningful point of integration across services [23]. In this view, decentralisation is valuable only if it strengthens practical agency, transparency, interoperability, and accountability.

Applied to media behaviour profiles, this means that moving data from central platform databases into pods does not by itself make the architecture safer. When data is distributed across multiple pods, services, or intermediaries, security responsibility also becomes more distributed. Each pod provider, client application, aggregation component, and data-using service becomes part of the trust chain. This may make it harder to secure every storage location, application, and access path consistently. The relevant design question is therefore not whether centralised or decentralised storage is inherently safer, but whether the architecture gives users meaningful control over storage, access, reuse, usage conditions, and accountability while making security responsibilities clear and governable [23].

1.2. Towards a User-Centric Architecture: Core Principles

A user-centric media profile architecture inverts the traditional platform-controlled model. Instead of each platform storing and governing user data independently, the user maintains their data in a personal data store. Platforms may only access or contribute to this data with explicit user consent, for a clearly defined purpose, and within a specified time frame. This introduces several requirements that guide the design choices discussed in this thesis.

Sovereignty is a foundational principle in the European data landscape, ensuring that individuals or organisations retain full control over the data they generate. Without sovereignty as an explicit design constraint, any user-centric system risks becoming another form of delegated control, where users technically have access to their data but lack the practical means to govern it.

Interoperability and Open Standards are equally important. Just as the internet depends on shared protocols such as HTTP, HTML, and URLs, user-centric systems need common standards so that independent services can exchange and interpret data. This idea is closely related to the principles discussed in the UGent micro-credential on Knowledge Graphs, where interoperability is treated as a central principle for connecting distributed data sources. Conway's Law illustrates why this matters: system design often mirrors the communication structures of the organisations that create it [7]. Without deliberate effort toward interoperability, data silos are likely to remain as a natural consequence of organisational boundaries, directly limiting the possibility of cross-service integration.

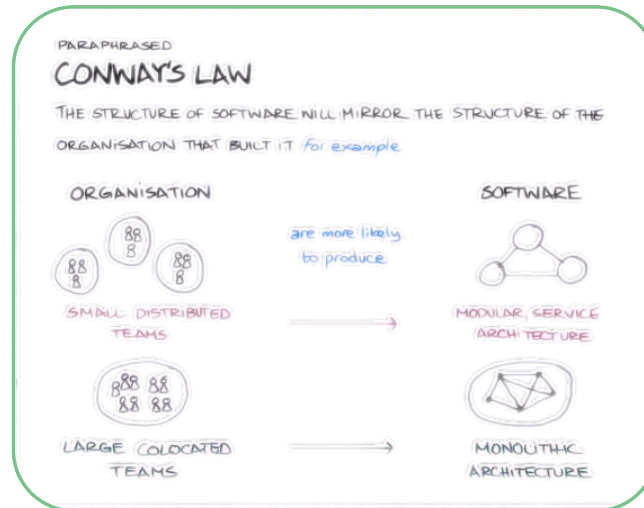


Fig. 2: Organisational structure tends to mirror system design. Adapted from Sketchplanations, “Conway’s Law” [7].

Compliance in a decentralised ecosystem cannot be implemented solely through the internal governance of one platform, because media behaviour data may involve several independent actors, including media services, pod providers, aggregation agents, application developers, and data consumers. Each actor may control only part of the data flow. As a result, compliance requires more than one organisation enforcing its own rules. It depends on shared policies, interoperable technical standards, verification mechanisms, contractual agreements, and alignment with regulatory requirements such as the GDPR [8].

Gaia-X is relevant in this context because it approaches trusted data sharing as a federated governance problem. Its Digital Clearing House concept provides verification mechanisms that can check whether participants and services comply with Gaia-X rules before they take part in data exchange [5]. This illustrates how decentralised or federated data ecosystems may require trusted verification services to support accountability across organisational boundaries. For this thesis, Gaia-X is therefore useful not as a direct replacement for Solid, but as an example of how broader data-space initiatives address trust, compliance, and interoperability at an ecosystem level.

Auditability follows directly from the compliance requirement: relevant data access and usage events should be traceable and verifiable when accountability, compliance, or dispute resolution requires it. Without transparent records of who accessed sensitive behavioural data, when access occurred, and for which stated purpose, it becomes difficult to detect misuse or demonstrate that data was handled according to agreed policies.

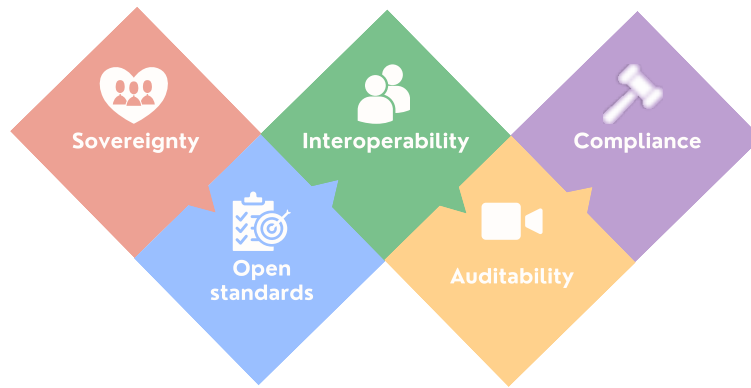


Fig. 3: Key principles of a user-centric media profile architecture.

1.3. The Solid Ecosystem as a Technical Foundation

“When I invented the World Wide Web, I envisioned technology that would empower people and enable collaboration... Solid returns the web to its roots by giving everyone direct control over their own data.” - Sir Tim Berners-Lee, creator of the World Wide Web [9]

The Solid initiative provides a concrete technological foundation for user-centric data management [10]. Its central concept is the **data pod** (or data vault), a decentralised storage space where individuals manage their own information independently of any platform. Users determine who can access their data, for what purpose, and for how long, directly supporting the principle of data **sovereignty** by shifting control away from what we call centralised intermediaries.

Technically, a Solid pod is an HTTP-accessible storage container that exposes its contents as Linked Data resources. Each resource is addressable by a URI and can be retrieved, created, updated, or deleted through standard HTTP verbs, which means that a pod is, in essence, a personal **slice** of the web that the user controls.

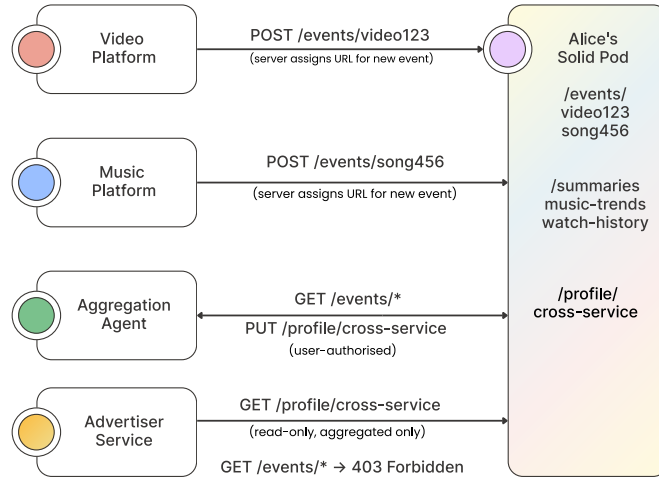


Fig. 4: Solid pod resources accessed via HTTP verbs.

As illustrated in figure 4, different actors interact with different parts of the pod using different verbs. A media service may POST a new interaction event or DELETE a retracted one, while an aggregation agent uses GET to read events and PUT to write back an updated profile. A recommendation service is restricted to GET on the aggregated profile container. Requests to raw event containers return 403 Forbidden. This per-resource access control is enforced by WAC or ACP policies attached directly to the pod, not by the applications themselves.

Data inside a pod is typically represented in RDF (Resource Description Framework). RDF is a standard model for describing data as relationships between things. Instead of storing information in isolated tables or application-specific formats, RDF represents data in simple statements, called triples: a subject, a predicate, and an object. For example, a media event could state that a user played a specific song at a specific time. Because each part of the statement can be identified using a URI, different applications can refer to the same concepts in a consistent way.

Listing 1: A media listen event represented as an RDF triple in Turtle syntax, using schema.org vocabulary.

```
@prefix schema: <https://schema.org/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix ev:     <https://alice.pod.example/events/> .

ev:event1 a schema:ListenAction ;
  schema:agent <https://alice.pod.example/profile/card#me> ;
  schema:object <https://musicbrainz.org/recording/song123> ;
  schema:startTime "2024-05-08T14:32:00Z"^^xsd:dateTime .
```

RDF data can be serialised in formats such as Turtle or JSON-LD. Turtle is compact and commonly used in Linked Data environments, while JSON-LD is easier to integrate with web applications that already use JSON, since it is valid JSON that any standard JSON parser can process and allows a `@context` field to be added to existing data structures without requiring dedicated RDF tooling. Other types of resources, such as images or audio files, can also be stored in a pod.

RDF is useful because it supports semantic interoperability. This means that data written by one application can be understood by another, as long as both applications use the same vocabulary or can map between different vocabularies. For media behaviour data, this is especially relevant. A video platform and a music platform may generate different kinds of interaction events, but RDF makes it possible to describe those events in a shared structure. For example, both a watched video and a played song can be represented as media interactions with a timestamp, a content item, and a user action, using vocabularies such as schema.org or domain-specific ontologies.

Listing 2: A listen event and a watch event from two different platforms, represented in the same RDF structure using schema.org vocabulary. Both use the same predicate set, making them directly comparable by any consuming application.

```
@prefix schema: <https://schema.org/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix ev:     <https://alice.pod.example/events/> .

ev:event1 a schema:ListenAction ;
  schema:agent <https://alice.pod.example/profile/card#me> ;
  schema:object <https://musicbrainz.org/recording/song123> ;
  schema:startTime "2024-05-08T14:32:00Z"^^xsd:dateTime .

ev:event2 a schema:WatchAction ;
  schema:agent <https://alice.pod.example/profile/card#me> ;
  schema:object <https://www.wikidata.org/wiki/Q57> ;
  schema:startTime "2024-05-08T20:15:00Z"^^xsd:dateTime .
```

Solid defines two main access control mechanisms. **Web Access Control (WAC)** is the older and more widely implemented model. It works in a way that is similar to file permissions: users, groups, applications, and resources are identified by URIs, and permissions are attached to specific resources or containers [11]. These permissions usually define whether an actor can read, write, append to, or control a resource.

WAC is relatively simple, which makes it easy to understand and implement. The following example shows a WAC policy granting Alice read access and Bob full control over a resource in his pod:

Listing 3: A WAC policy granting Alice read access and Bob full control over a resource in Bob's pod, using URI-identified agents and explicit permission modes.

```
@prefix acl: <http://www.w3.org/ns/auth/acl#>.

<#read-auth>
  a acl:Authorization;
  acl:agent <https://alice.example/profile#me>;
  acl:accessTo <https://bob.example/pod/history.ttl>;
  acl:mode acl:Read.

<#owner-auth>
  a acl:Authorization;
  acl:agent <https://bob.example/profile#me>;
  acl:accessTo <https://bob.example/pod/history.ttl>;
  acl:mode acl:Read, acl:Write, acl:Control.
```

However, this simplicity also limits what it can express. For example, WAC is not well suited for more specific conditions such as “only allow access when the request comes from a specific application” or “only allow access when the user is authenticated through a trusted identity provider.”

Access Control Policies (ACP) provide a more flexible alternative [12]. ACP allows access rules to be expressed in a more detailed way, including conditions about which user, application, or context is involved. This makes ACP more suitable for systems where access should depend not only on who is requesting the data, but also on why and under which conditions the data is being accessed. The following example shows an ACP policy that grants read access only when Alice is using a specific trusted application:

Listing 4: An ACP policy granting read access to a resource only when the request comes from a specific agent using a specific client application, combining both conditions through a matcher.

```
@prefix acp: <http://www.w3.org/ns/solid/acp#>.

<#matcher>
  a acp:Matcher;
  acp:agent <https://alice.example/profile#me>;
  acp:client <https://music-app.example/clientid>.

<#policy>
  a acp:Policy;
  acp:allow acp:Read;
  acp:allOf <#matcher>.

<#acr>
  a acp:AccessControlResource;
  acp:resource <https://bob.example/pod/history.ttl>;
  acp:accessControl <#policy>.
```

Both WAC and ACP support an important Solid principle: access control should not be hidden inside each individual application. Instead of allowing every media platform to decide for itself what it can do with user data, access rules are linked directly to the user's resources in the pod. For example, a user can decide that a music service may read their listening history, while another service may only access an aggregated profile. This makes the data less dependent on the internal policies of one platform and supports a more user-centric form of data governance.

However, this does not mean that the pod host is completely irrelevant. The server still enforces the access rules, which means that the hosting provider remains part of the trusted infrastructure. Nevertheless, Solid improves on traditional platform-controlled models because access policies are defined around the user's data and can, in principle, move with the pod if the user changes provider.

Authentication is handled through **Solid-OIDC** [13], which builds on the widely adopted OAuth 2.0 and OpenID Connect standards. Users are identified through a **WebID** [14], a globally unique, user-controlled URI, enabling identity verification across services. The WebID is typically hosted within the user's own pod, which means that the user's identity document is itself a piece of data they control, and can be updated, extended, or moved without requiring the cooperation of any identity provider.

For deployment within the European context, the Community Solid Server (CSS), developed by imec and Ghent University, offers a ready and open-source implementation of the Solid specifications [15]. CSS is modular by design, with configurable components for storage backends, authentication flows, and access control mechanisms. This makes it useful for research prototypes and experimental deployments where different architectural choices need to be tested. Joachim Van Herwegen and Ruben Verborgh explicitly describe CSS as an implementation of the Solid specifications tailored towards research and development, prioritising modularity and extensibility over production-scale throughput [24].

While Solid provides a useful foundation for user-controlled storage and access control, it does not solve every governance problem on its own. Access control mainly answers the question: “Is this agent allowed to access this resource?” However, it does not fully answer questions such as: “For which purpose may the data be used?”, “How long may it be retained?”, or “Can it be shared with others?” These questions are especially important for media behaviour profiles, because the sensitivity of the data depends not only on who accesses it, but also on how it is used. Therefore, additional mechanisms are needed for policy enforcement, cross-service aggregation, auditability, and legal compliance. These are discussed in the following sections.

1.4. Requirements for a User-Centric Architecture

The previous sections have established the principles, privacy concerns, and technical standards that a user-centric media profile architecture must draw on. This section derives five concrete requirement areas from that body of literature, each specifying what a component must be capable of, independent of which technology implements it. The GDPR and the Solid ecosystem together ground the requirements for personal data storage [8, 10]. WebID and Solid-OIDC define what an identity and authentication mechanism must provide [13, 14]. ODRL and the Trustflows framework establish what a policy layer must be able to express [17, 21]. The MyData model of user-authorised data integration informs the aggregation requirements [23]. Finally, GDPR accountability and transparency obligations set the bar for auditability [8, 26].

Personal Data Store A user-centric architecture requires a personal data store that enforces access policies per resource, exposes its contents in a machine-readable and interoperable format, and supports data portability so that the user is not locked into a single provider [8]. Without per-resource access control, data minimisation cannot be achieved: services would need to be trusted to self-limit their access rather than being technically constrained. Without interoperability, data contributed by one service cannot be consumed by another.

For media behaviour data, the store must also support partitioned storage. Raw interaction events, per-service summaries, and an aggregated cross-service profile have different sensitivity levels and should be governed by different access rules. A music service must be able to append new listening events without gaining read access to video behaviour. A recommendation service must be able to read an aggregated profile without gaining access to raw events. The store must therefore allow different access rules to be attached to different parts of the data.

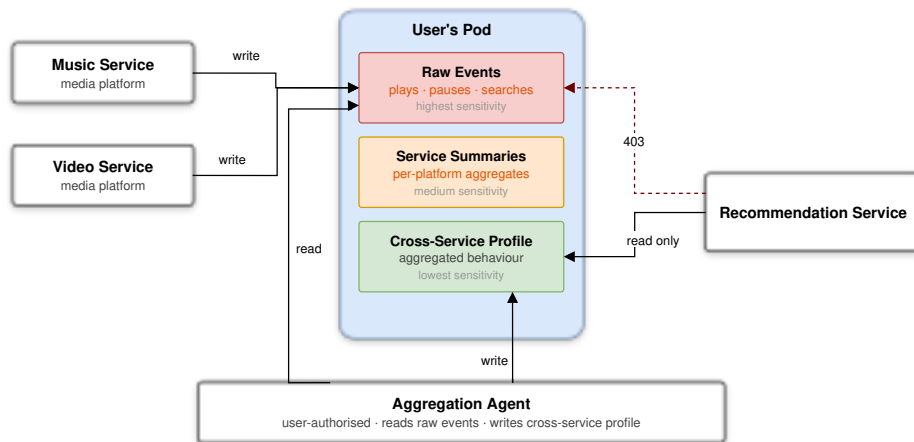


Fig. 5: The user's pod partitioned into three containers at different sensitivity levels. Media services may only write to raw event containers. The aggregation agent reads raw events and writes back a cross-service profile. The recommendation service may only read the aggregated profile, and is denied access to raw events.

For continuously generated media behaviour data, the store must additionally handle high-volume ingestion and support change notifications so that downstream processors can react to new events incrementally rather than polling the full dataset.

For high-volume media behaviour data, the choice of Solid server implementation matters significantly. Kvasir is a cloud-native, microservice-based platform developed at imec-IDLab that functions as a scalable data broker within the Solid and Linked Data ecosystem [16]. Unlike CSS, which prioritises modularity and research flexibility, Kvasir is built around industry-standard infrastructure including Apache Kafka for event streaming and ClickHouse for analytical storage. It exposes data through multiple APIs including native RDF formats such as JSON-LD and Turtle, GraphQL, and S3-compatible interfaces, and provides built-in multi-tenancy, fine-grained access control, and real-time push notifications. This makes Kvasir well-suited for applications that generate continuous streams of behavioural data, where a file-based reference implementation would not meet the performance requirements.

Identity and Authentication The architecture requires a shared, user-controlled identity mechanism that allows data from different services to be attributed to the same user without depending on any single platform's identity system. Without such a mechanism, a cross-service profile cannot be built: each platform would identify the user differently, and there would be no stable point of integration. In Solid, this role is fulfilled by the WebID [14], a globally unique URI the user controls. Authentication is handled through Solid-OIDC [13], which builds on OpenID Connect. It allows applications to verify a user's identity and request access to resources in the user's pod.

However, identity also creates privacy challenges. If the same WebID is used across many services, it may become a point of correlation. Several services could recognise that they are interacting with the same user. For this reason, privacy-preserving deployments may require different WebIDs for different contexts, such as personal media use, professional activity, or research participation.

Together with WAC or ACP, WebID and Solid-OIDC support a user-centric access model. Users can identify themselves across services, while access to their data remains governed by policies attached to resources in their own pod. This supports portability, but it also shows that identity design must carefully balance convenience and privacy.

For media behaviour profiles, this means that identity design must allow users to separate different services and access decisions. What a music platform knows about a user should not automatically become available to a video platform simply because both use the same identity mechanism.

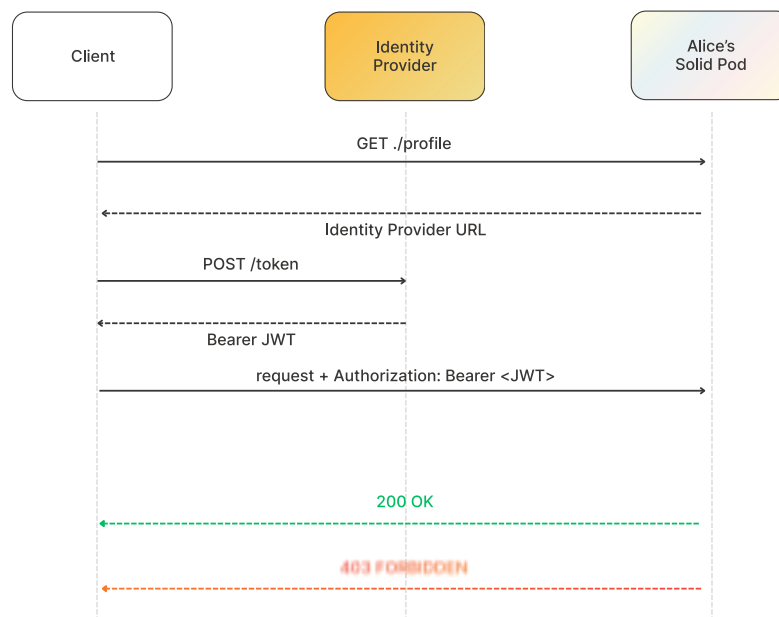


Fig. 6: WebID and Solid-OIDC enable a user to authenticate across independent services using a single user-controlled identity document hosted in their pod.

Policy Enforcement Access control alone is insufficient. The architecture requires a policy layer capable of expressing and enforcing conditions on data use that go beyond simple read and write permissions. A service may be technically permitted to read listening-history data without that permission saying anything about how long the data may be retained, whether it may be shared, or whether it may be used for advertising. These usage constraints cannot be expressed in WAC or ACP.

This is where the concept of Trustflows becomes relevant. Trustflows.eu [21] describes Trustflows as an approach for building trustworthy data flows that are interoperable, legally compliant, and user-centric. In this thesis, the term is used to refer to governed data flows in which data does not simply move between actors, but moves together with explicit conditions, responsibilities, and evidence for accountability.

For this reason, the architecture requires a policy layer. Data exchanges can be governed through policies expressed in ODRL: the Open Digital Rights Language [17].

ODRL can describe permissions, prohibitions, and duties related to data use. For example, a policy could allow a recommendation service to use viewing-history data for personalisation, prohibit sharing it with advertising networks, and require deletion after a defined period.

A policy engine can evaluate these rules before access is granted. In this architecture, the policy layer sits between the requesting service and the data stored in the pod. When a service requests access, the policy engine evaluates the request against the applicable ODRL policy by checking four elements: the requesting party, the requested action (such as read or write), the target resource, and any constraints attached to the policy, such as a stated purpose, a retention limit, or a prohibition on secondary use. If the request satisfies all conditions, access is granted. If any element does not match, access is denied.

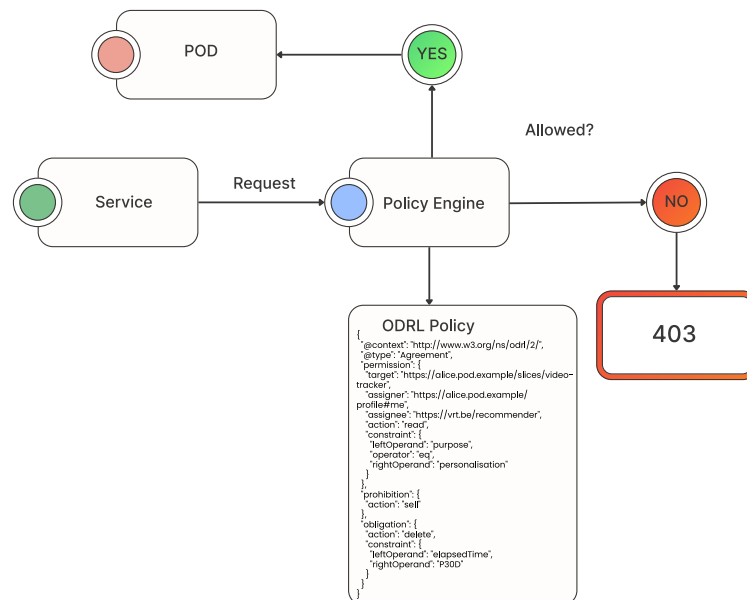


Fig. 7: The ODRL policy layer intercepts access requests and evaluates them against user-defined permissions, prohibitions, and duties before granting or denying access to pod resources.

This makes governance more proactive. Instead of only detecting misuse afterwards, some restrictions can be enforced before data is accessed. However, technical enforcement has limits. Once a service has legitimately received data, a policy engine cannot fully prevent later misuse outside the pod environment. This is why policy enforcement must be combined with audit logs, accountability mechanisms, and legal safeguards.

In this architecture, Trustflows therefore connects policy enforcement with accountability. The policy layer helps determine whether a data exchange is allowed, while audit mechanisms provide evidence of which exchanges occurred and under which

stated conditions. Together, these mechanisms support data flows that are not only technically possible, but also governed, explainable, and verifiable.

Aggregation The architecture requires a user-authorised aggregation component that can read permitted data from multiple service partitions, derive a unified cross-service profile, and write that profile back to the store under separate access rules [23]. This component must not require platforms to share data directly with each other: instead, each service contributes to the user’s store, and the aggregation component operates on that store on the user’s behalf.

The purpose of this separation is to avoid giving every platform direct access to all raw behavioural data. Instead, services can contribute events or summaries, while the aggregation agent produces a profile that can later be shared under separate access rules. For example, a video platform and a music platform may both contribute interaction data, but a recommendation service may only receive access to the aggregated profile.

The aggregation component must therefore be configurable at the level of the user’s consent: which services are included, which data types are considered, and how long intermediate data is retained must all be governed by the user’s stated preferences rather than by the component’s internal logic [23].

For continuously generated media behaviour data, efficiency is important. Reprocessing the full dataset every time the profile is updated would become inefficient as the amount of data grows. Event streaming techniques such as Linked Data Event Streams (LDES) can help address this [18]. An LDES represents data as a stream of timestamped events that can be processed incrementally. This allows the aggregation agent to consume only new or changed events instead of recalculating the entire profile each time.

This is especially relevant for media behaviour data, where new events may be generated frequently. Incremental aggregation can reduce processing overhead and make the profile easier to keep up to date.

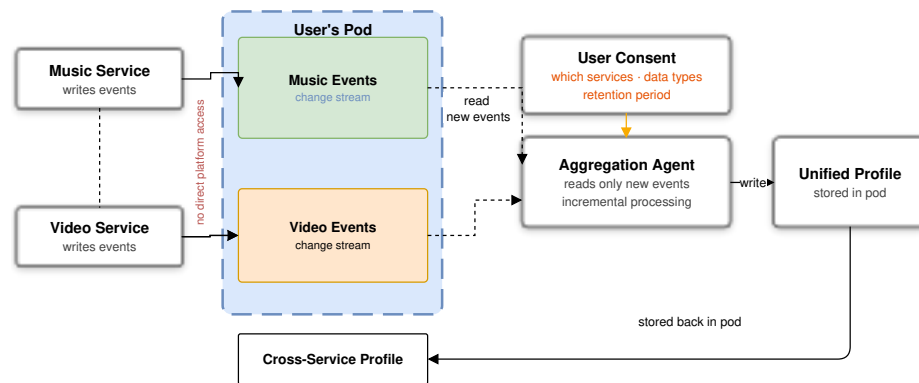


Fig. 8: The aggregation agent reads only new events from each service partition using LDES, governed by user consent settings. It derives a unified cross-service profile and writes it back to the pod, without any platform having direct access to another platform’s data.

Auditability Because data is no longer governed by one central platform, the archi-

architecture requires mechanisms that make access and usage transparent and verifiable after the fact [26]. In a centralised system, a single party holds all access logs internally. In a decentralised architecture, no such party exists, which means auditability must be designed in explicitly rather than assumed. The architecture therefore requires that every access by a service, a policy engine, or an aggregation component produces a traceable record of who accessed what, when, and for which stated purpose. These records must be available to the user directly, not just to the hosting provider, so that users and regulators can verify that data was handled according to agreed conditions. This supports compliance with GDPR principles of transparency, accountability, and data protection by design [8], but it also goes beyond compliance: auditability is what makes the architecture’s claims about user control credible in practice.

These five building blocks are not independent. They form a chain: when a media service records a user interaction, it authenticates with its own service credentials and writes the event to the container in the user’s pod to which the user has previously granted it write access. The policy engine checks whether this write matches the user’s stated consent before it is permitted. Periodically, the aggregation agent reads permitted data from multiple service containers, computes a cross-service profile, and stores it back in the pod. Every access by the service, the policy engine, and the aggregation agent is recorded in an audit log the user can inspect. This chain connects identity, storage, policy, aggregation, and auditability into a single governed data flow that is transparent at every step.

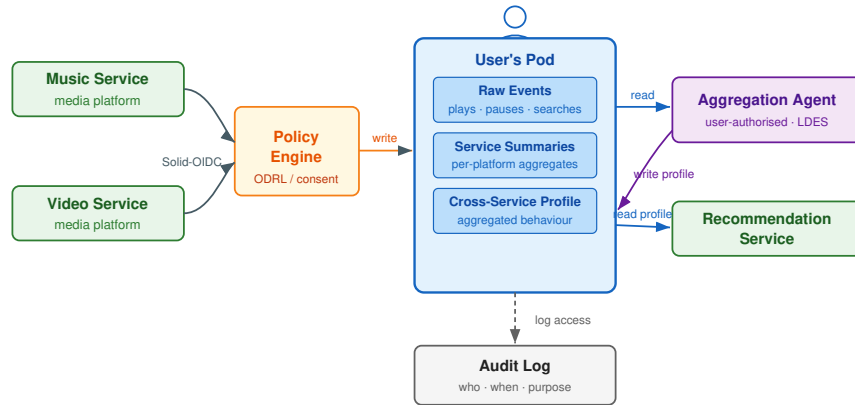


Fig. 9: The five architectural building blocks and their interactions. Media services authenticate via Solid-OIDC and write events through the policy engine to the user’s pod. The aggregation agent reads events and writes back a cross-service profile. Every access is recorded in the audit log.

1.5. Dataspace Integration and Eventual Interoperability

The proposed architecture also fits within the broader European data space vision. The European Strategy for Data promotes trusted and interoperable data sharing across sectors [19], while initiatives such as Gaia-X show that decentralised ecosystems still require shared governance, verification mechanisms, and trust frameworks [5]. In this thesis, Solid addresses the user-controlled personal data layer, while data

space initiatives provide useful context for future organisational interoperability.

European Strategy for Data

A common European data space, a single market for data



Fig. 10: European Strategy for Data: a common European data space enabling trusted and interoperable data sharing across sectors (European Commission).

This connection is expressed through eventual interoperability: the system does not need to align with every future data space standard from the start, but it should use explicit semantics, stable identifiers, and open standards so that later integration remains possible [22].

1.6. Stakeholder Considerations

To situate the proposed architecture within the Flemish media landscape, a structured written questionnaire was conducted with a practitioner from the Flemish media sector. MediaNet Vlaanderen is a cross-media consultation platform that brings together more than 80 organisations active in content creation and distribution within the Flemish media sector. The respondent is a Programme and Communications Director there, with direct operational experience in media data governance and involvement in the Solid4media initiative, an ongoing industry project that explores the application of Solid technology to media data management and closely aligns with the scope of this thesis.

The consultations suggest that media behaviour profiles may be valuable not only for personalisation, but also for organisations that analyse media trends and innovation, with analyses of media professionals' evaluations of technological innovation described as "very valuable." At the same time, a concern was raised about transparency: user-facing insights should remain simple and understandable, so that transparency does not become too complex to interpret in practice. This supports the need for usable consent flows, clear access explanations, and manageable policy interfaces as part of the proposed architecture.

A personal data vault architecture is considered feasible only in the medium term. The main barrier is ecosystem adoption: such a system becomes valuable only when a sufficiently large pool of participating services and users exists. This reinforces the idea

that the proposed architecture is not only a technical system, but also a sociotechnical one that depends on scale, trust, governance, and stakeholder incentives to deliver its intended value.

1.7. Limitations and Challenges of User-Centric Media Profile Architectures

Several challenges remain. The first is adoption. A user-centric architecture only becomes useful when multiple services support it. If only one or two platforms participate, the value of a cross-service media profile remains very limited. This creates a chicken-and-egg problem: platforms may hesitate to adopt the system until users demand it, while users may not see its value until enough platforms support it. Technical design alone cannot solve this. Adoption would also require incentives, regulation, partnerships, and use cases that clearly show the benefits of the approach. This barrier was also identified in the expert questionnaire conducted for this thesis, where a personal data vault architecture was described as feasible only in the medium term precisely because ecosystem adoption is a precondition for its value [23].

Performance is another important design constraint. A decentralised architecture may introduce more latency than a centralised platform model. Data may need to move between services, pods, policy engines, and aggregation agents before a result can be produced. Research into decentralised search and querying over Solid pods has shown that distributing data across many independent storage locations introduces significant overhead compared to centralised indexing, particularly for operations that span multiple pods [25]. Some of this overhead can be reduced through caching, pre-computed profiles, or incremental updates, but performance remains a relevant constraint especially for features that require fast responses.

A third challenge is usability. Giving users more control over their data is valuable, but it can also make the system harder to use. If users are asked to approve too many permissions or understand complex policies, they may become overwhelmed or simply accept requests without reading them. This would weaken the goal of meaningful user control. The MyData initiative explicitly identifies practical usability as a key requirement for user-centric data systems, noting that formal legal rights over data are insufficient without tools that are understandable and manageable in practice [23].

Standardisation is also necessary. Technologies such as Solid, ODRL, and LDES provide important building blocks, but they do not automatically guarantee interoperability. Services still need to agree on shared vocabularies, data models, profile structures, and conformance rules. Without this coordination, different implementations may technically follow the same standards but still fail to work together in practice. As Colpaert argues, governance and shared semantics are just as important as the technical standards themselves [22].

Finally, security and threat modelling remain major concerns. Moving data away from centralised platforms does not remove security risks, it changes where those risks appear. A systematic assessment of the Solid protocol against security and privacy obligations has shown that most known threat categories apply to Solid environments, including risks such as token theft through malicious pods and identity injection via WebID documents [26]. Because responsibility for security is distributed across pod providers, client applications, and identity providers, consistent enforcement becomes harder to guarantee than in a centralised system.

These limitations do not make the proposed architecture unsuitable, but they show that its success depends on more than the technical design. Adoption, performance, usability, standardisation, and security all influence whether a user-centric media profile architecture can work in practice. The approach should therefore be understood as a sociotechnical system: it requires not only protocols and software, but also governance, incentives, and trust.

2. Architecture and Design Decisions

The previous chapter established the theoretical foundations and requirements for a user-centric media behaviour profile architecture. It identified five requirement areas that any valid architecture must address: a personal data store with per-resource access control and portability, a shared identity mechanism, a policy layer for usage conditions beyond access control, a user-authorised aggregation component, and auditability mechanisms that make data access transparent and verifiable. These are summarised in Fig. 11.

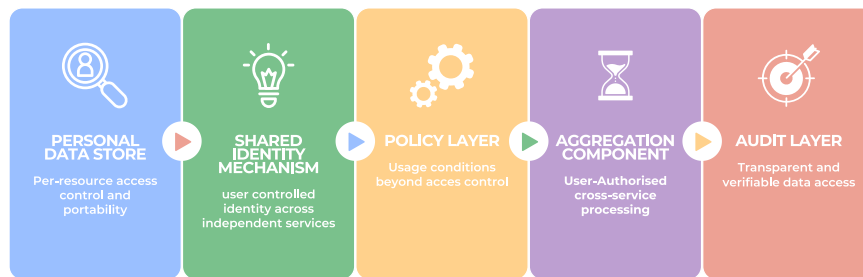


Fig. 11: The five requirement areas that any valid user-centric media behaviour profile architecture must address, as established in the literature study.

This chapter translates those requirements into concrete architectural design decisions. It answers the central research question of this thesis: **which key design decisions need to be made when building media behaviour profiles across multiple media services?**

The chapter distinguishes between two levels of design. At the abstract level, each section defines a component role and the responsibility that role must fulfil. At the prototype level, each section describes one concrete implementation choice used in the proof of concept. This distinction is important because the architectural contribution of this thesis is **not tied to one specific technology stack**. A deployment using the Community Solid Server instead of Kvasir, WAC or ACP instead of OpenFGA, or LDES instead of Redpanda would still implement the same architecture if the same responsibilities are fulfilled.

The design decisions are guided by five principles established earlier in this thesis.

European data sovereignty requires that personal media data remains under user control and is governed in line with European values and regulation. W3C open standards ensure that data and identity are not locked into proprietary formats. Open-source implementation supports transparency, reproducibility, and adaptability. Interoperability across autonomous stakeholders allows independent media services, storage providers, and processing components to participate without sharing one internal infrastructure. Finally, competency questions provide the functional test: the architecture is valid only if it can answer the questions it was designed to support.

2.1. Competency Questions

Before defining the architectural components, it is necessary to define what the system must be able to answer. Competency questions translate the research question into concrete functional requirements. Three questions were formulated and assessed through a structured written questionnaire completed by a practitioner from the Flemish media sector. The respondent holds a senior position at MediaNet Vlaanderen, has operational experience with media behaviour data, and is involved in the Solid4media initiative. The questionnaire therefore provided practical input on which capabilities are most relevant from a media-sector perspective.

CQ1: Which content has a given user consumed within a specified time period?

This is the foundational retrieval question. The system must be able to retrieve a user's listening and viewing history, filtered by time, through an authorised access point. Answering this question requires a queryable data store, stable user identifiers, and a semantic data model that represents media events consistently across services. The expert rated this as the most relevant competency question.

CQ2: Which services may access which parts of a user's consumption history, and under what conditions? This is the governance question. The system must support selective disclosure, for example allowing one application to access watched films from the past year without exposing music listening history or raw interaction events. Answering this question requires data partitioning, per-partition access control, and a policy layer capable of expressing usage conditions. The expert rated this question highly, confirming the practical importance of user-controlled access and sharing.

CQ3: How can consumption data from multiple platforms be integrated in a standardised way? This is the interoperability question. The system must allow data from independent media services to be combined without requiring those services to share internal databases or proprietary schemas. Answering this question requires shared semantics, stable external identifiers, and an aggregation component that operates on behalf of the user. The expert rated this question moderately, noting that standardised integration would be valuable for media trend analysis and innovation research, while ecosystem adoption remains the larger practical barrier.

Together, these competency questions define the minimum capability of the architecture. A system that cannot answer CQ1 cannot function as a media behaviour profile store. A system that cannot answer CQ2 cannot be considered user-centric. A system that cannot answer CQ3 does not justify the cross-service premise of this thesis.

With these requirements established, Fig. 12 presents the full system architecture.

Reading the diagram from left to right follows the data flow. The Music Service and Video Service write media events into the Kvasir User Pod, where they are stored in typed slices. Each slice emits change events to Redpanda, which triggers the Change Processor. The Change Processor enriches raw media events with external metadata, derives aggregated insights, and writes the resulting cross-service profile back into the pod. Ollama reads this profile to generate local recommendations. The identity and authentication layer governs each interaction: Keycloak issues service tokens and OpenFGA evaluates whether the requesting service has the required relationship on the target slice. The logging layer records authentication, authorisation, and change events for retrospective accountability. The ODRL-based policy layer is included in the conceptual architecture but remains future work in the prototype. It would, for example, allow a user to permit a recommendation service to read their viewing history for personalisation while prohibiting that service from retaining a copy or sharing it with third parties.

Three implementation choices in the prototype deviate from the Solid-native mechanisms recommended in the literature. These are described in full in the section on prototype deviations at the end of this chapter.

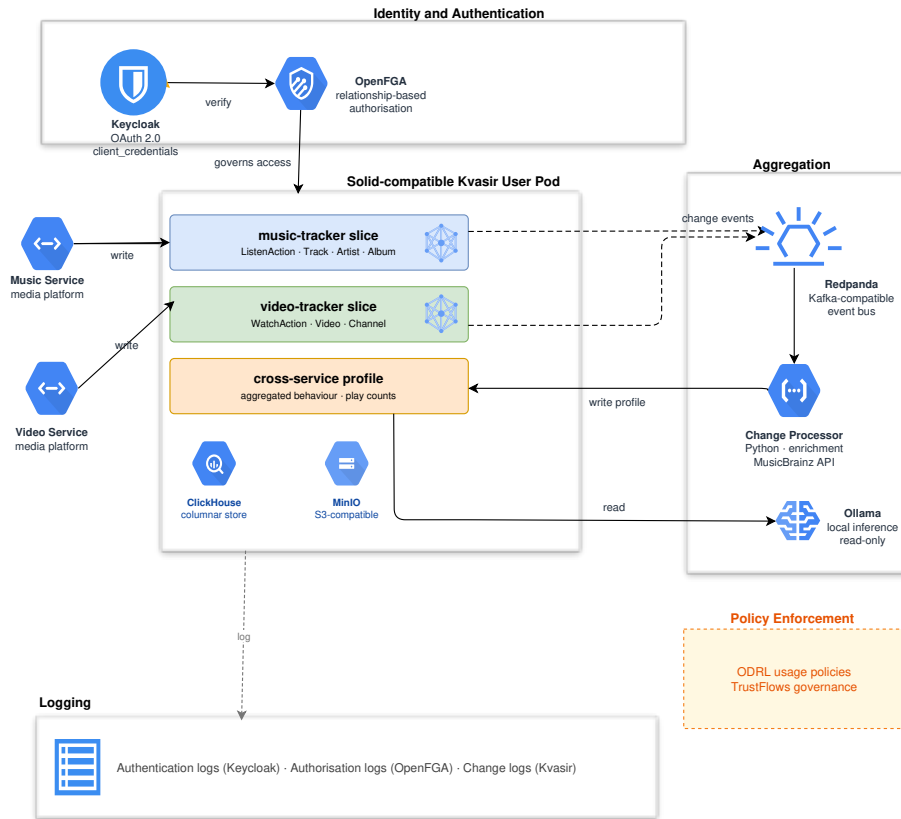


Fig. 12: System architecture of the prototype. The identity and authentication band at the top governs access through Keycloak and OpenFGA. Media services write events into typed Kvasir slices. Change events flow through Redpanda to the Change Processor, which enriches the events and writes a cross-service profile back to the pod. Ollama reads the profile for local inference. The logging layer records authentication, authorisation, and change events. ODRL-based policy enforcement is shown as future work.

The relationship between the architectural roles, prototype choices, and competency questions is summarised in Table 1.

Table 1: Traceability between architectural roles, prototype choices, and competency questions.

Architectural role	Prototype choice	Main responsibility	Competency questions
Personal data store	Kvasir User Pod	Store user-controlled media behaviour data	CQ1, CQ2
Data partitioning	Kvasir slices	Separate raw and aggregated data under different access rules	CQ2, CQ3
Analytical storage	ClickHouse and MinIO	Support high-volume event queries and binary asset storage	CQ1, CQ3
Identity provider	Keycloak and Solid-OIDC endpoint	Identify users, services, and background components	CQ1, CQ2
Authorisation engine	OpenFGA	Enforce per-slice read and write permissions	CQ2
Semantic data model	JSON-LD, schema.org, Music Ontology	Represent media events consistently across services	CQ1, CQ3
Event bus	Redpanda	Notify processors when new events are written	CQ1, CQ3
Aggregation processor	Python change processors	Enrich events and derive cross-service profiles	CQ1, CQ3
Inference service	Ollama	Generate recommendations from the profile without external data transfer	CQ1
Policy layer	ODRL, future work	Express and enforce usage conditions after access	CQ2
Audit layer	Authentication, authorisation, and change logs	Record who accessed or changed which data	CQ2

2.2. Personal Data Store

In conventional media platforms, behavioural data is stored within the platform’s own infrastructure. The platform determines what is retained, how it is analysed, and with whom it is shared. A user-centric architecture reverses this model. Media behaviour data must be stored in a personal data store controlled by the user, while services receive access only under explicitly authorised conditions.

The architectural requirement is therefore that the personal data store must support four properties. First, it must enforce access rules at the level of individual resources or partitions. Second, it must expose data in a machine-readable and interoperable format. Third, it must support portability so that the user is not locked into one provider. Fourth, it must provide a way for downstream components to detect changes, because media behaviour data is generated continuously.

The Solid ecosystem provides the natural foundation for this role. A Solid pod is a user-controlled data store that exposes resources through web standards and allows access to be governed by policies attached to the data. In principle, several implemen-

tations could fulfil this role, including the Community Solid Server, Inrupt Enterprise Solid Server, or future Solid-compatible servers. A non-Solid personal SPARQL endpoint or linked data server could also satisfy the same architectural role if it provides equivalent guarantees for user control, access control, interoperability, and portability.

The prototype uses Kvasir as the personal data store. Kvasir is a Solid-based data broker developed at imec-IDLab. It extends the standard Solid resource model with schema-governed slices, a GraphQL query interface, event notifications, and analytical storage. These additions make it suitable for high-volume structured media behaviour data. In Fig. 12, Kvasir is shown as the central Kvasir User Pod.

This choice should not be interpreted as making Kvasir the architecture itself. Kvasir is one implementation of the personal data store role. The architectural decision is that media behaviour data must be stored around the user rather than inside individual media platforms.

2.3. Data Partitioning

A user-centric system must not store all media behaviour data in one undifferentiated graph. If all data is stored together, selective sharing becomes difficult. A music service that writes listening events could gain visibility into video behaviour. A recommendation service that only needs an aggregated profile could receive unnecessary access to raw events. This would violate data minimisation and weaken user control.

The architecture therefore requires data partitioning. Raw listening events, raw viewing events, enriched media metadata, and aggregated cross-service profiles should be stored in separately governable areas. Each partition must be independently authorisable, independently queryable, and independently subscribable.

In a standard Solid deployment, this role could be fulfilled through containers governed by WAC or ACP policies. In a SPARQL-based system, named graphs could serve a similar purpose. Another alternative would be to use separate pods for different media domains. The mechanism is less important than the property it provides: access must be controllable at a level more precise than the whole pod.

The prototype implements partitioning through Kvasir slices. A slice is a schema-governed partition of the user pod, with its own data model, access rules, and change stream. The prototype defines two input slices: **music-tracker** for listening events, tracks, albums, and artists, and **video-tracker** for watch events, videos, and channels. A third partition, the **cross-service profile**, stores the aggregated output produced by the Change Processor. This partition holds two kinds of data. The first is per-creator interaction counters, derived by aggregating listening and viewing events across the music and video slices. The second is confirmed identity links between music creators and video creators, expressed as **schema:sameAs** triples. When a user confirms that a SoundCloud artist and a YouTube channel belong to the same person, the system writes this assertion to the cross-service profile. The pod thereby becomes the authoritative source of identity decisions rather than relying on heuristic name matching.

This directly supports CQ2. A service may be granted write access to the **music-tracker** slice without receiving access to the **video-tracker** slice. A recom-

mentation service may be granted read access to the aggregated profile without seeing raw event histories. The user does not need to understand the full RDF graph to benefit from selective disclosure. The partitioning structure makes access boundaries explicit.

The trade-off is that Kvasir slices do not behave exactly like standard Solid containers. They are accessed through GraphQL queries and JSON-LD delta writes rather than only through LDP HTTP operations [27]. This improves query expressiveness and analytical performance, but it means that the prototype is not a pure standard Solid resource implementation. The architecture remains Solid-based in orientation because it is centred on a user-controlled pod, linked data, and access rules attached to user data. The prototype, however, uses Kvasir-specific mechanisms to support the workload.

Replacing Kvasir with a different Solid server would therefore require significant rewriting of the implementation. The ingestion layer would need to use LDP HTTP instead of Kvasir’s GraphQL interface. The change processor would need to consume Solid notifications or LDES instead of Redpanda events. Access control would need to be reconfigured from OpenFGA to WAC or ACP. These are implementation changes, not changes to the architectural roles.

2.4. Analytical Storage and Binary Assets

Media behaviour profiles are append-heavy and aggregation-heavy. Every play, pause, skip, search, or watch action creates a new event. Useful profile queries often aggregate over large event sets: most-played artists, most-watched channels, repeated interests, recent consumption patterns, or changes over time. The storage layer must therefore support efficient analytical queries, not merely the storage of RDF triples.

A traditional RDF triplestore offers strong semantic query capabilities through SPARQL, which is valuable for graph traversal and semantic reasoning. However, media behaviour profiling also requires fast aggregation over large volumes of time-stamped events. A relational database such as PostgreSQL can support aggregation, but does not offer native semantic graph capabilities. A column-oriented analytical store is therefore a suitable trade-off for the prototype workload.

The prototype uses ClickHouse [28] as the analytical storage backend inside Kvasir. ClickHouse is optimised for fast aggregations over append-heavy datasets, which fits the profile of media behaviour events. The trade-off is that the prototype does not expose native SPARQL querying. Queries are issued through Kvasir’s GraphQL layer.

Binary assets are stored separately. Album cover art from MusicBrainz and artist thumbnails from Wikipedia are stored in MinIO, an S3-compatible object store. Kvasir stores references to these assets in the knowledge graph. This keeps the analytical store focused on structured data while allowing media-related files to be retrieved when needed.

This decision supports CQ1 and CQ3 by making profile queries and cross-service aggregation practically feasible at larger volumes.

2.5. Identity Provider

Access control only works when the system can reliably determine who is making a request. The architecture must distinguish between a music service, a video service, an aggregation processor, a recommender, and a human user. If these actors cannot be identified separately, the system cannot assign different permissions to them.

In the Solid ecosystem, the natural identity mechanism is WebID combined with Solid-OIDC [13, 14]. A WebID is a URI that identifies a user or agent. Solid-OIDC allows applications to authenticate and request access to resources in a user's pod. This is especially suitable for browser-based interactions where a human user logs in and grants an application access.

Kvasir supports this model through its `/podId/solid/` endpoint. A Solid client can authenticate using Solid-OIDC and interact with resources through standard Solid mechanisms. RDF files written through this endpoint are ingested into Kvasir's knowledge graph, keeping the Solid-compatible interface connected to the internal storage model.

The prototype also contains automated background components. The music tracker, video tracker, change processor, and recommender operate continuously without a user logging in for every request. For this class of actor, the OAuth 2.0 `client_credentials` flow is more appropriate. A service authenticates using its `client_id` and `client_secret`, receives a bearer token, and attaches that token to its requests.

The prototype uses Keycloak as the OIDC provider for these service-to-service interactions. Each service is registered as a separate Keycloak client. Kvasir validates the received token and passes the authorisation decision to OpenFGA. This separates identity from authorisation: Keycloak determines who is making the request, while OpenFGA determines what that actor is allowed to do.

The `client_credentials` flow is therefore not a replacement for Solid-OIDC. It complements Solid-OIDC for automated service components. A complete deployment would use Solid-OIDC for human users and browser-based applications, and `client_credentials` for trusted background services.

In the current prototype, the web application and browser extension interact with Kvasir exclusively through the GraphQL slice API rather than through the Solid-compatible LDP endpoint. The Solid endpoint is available but not exercised by the prototype clients.

The limitation is that Kvasir's Solid endpoint does not currently enforce WAC or ACP policies. Access control for requests, including requests arriving through the Solid-compatible endpoint, is handled through OpenFGA. This is a prototype constraint rather than a conceptual rejection of Solid-native access control.

2.6. Authorisation Engine

Authentication establishes who is making a request. Authorisation determines whether that actor may perform a specific operation on a specific resource. For this architecture, authorisation must happen at the level of individual partitions rather than the entire pod. A service should be able to write to one slice while being denied access to another. Access rules must be attached to the user's data environment rather

than hidden inside the internal logic of each media service.

Solid provides two native mechanisms for this role. Web Access Control [11] defines permissions such as read, write, append, and control over resources or containers. Access Control Policies [12] provide a more expressive model that can include conditions such as which application is making a request. Outside Solid, Open Policy Agent and Casbin provide policy-as-code alternatives.

The prototype uses OpenFGA. OpenFGA is a relationship-based authorisation engine. It models access as relationships between subjects and objects. For example, the music tracker may hold the `writer` relationship on the `music-tracker` slice, while the web application may hold the `reader` relationship on the `cross-service profile`. When a request reaches Kvasir, Kvasir asks OpenFGA whether the requesting service has the required relationship on the target slice. If the relationship exists, access is granted. If it does not, the request is rejected.

OpenFGA is used because Kvasir does not yet implement WAC or ACP for its Solid-compatible storage API. The choice is therefore pragmatic. It allows the prototype to demonstrate slice-level access control even though it deviates from Solid-native authorisation mechanisms.

This decision supports CQ2 by enforcing selective access to different parts of the user's media behaviour data. It also highlights an important distinction: the architecture requires fine-grained authorisation, but does not require one specific authorisation technology. WAC, ACP, OpenFGA, or another mechanism could fulfil the role if it supports independently governable partitions.

2.7. Semantic Data Model

Cross-service media profiling is only possible if data written by one service can be understood by another. In a centralised system, this is often solved through one shared internal database schema. In a decentralised system, no such shared schema can be assumed. Interoperability must instead be achieved through explicit semantics, shared vocabularies, and stable identifiers.

RDF provides the foundational data model. It represents information as triples and allows resources, users, actions, and media objects to be identified by URIs. RDF can be serialised in different formats. Turtle is compact and common in linked data environments. JSON-LD is better suited to web applications because it remains valid JSON while preserving linked data semantics through a context document.

The prototype uses JSON-LD as its serialisation format. Media interactions are represented using schema.org types such as `ListenAction`, `WatchAction`, and `InteractionCounter`. Music-specific entities use the Music Ontology, including `mo:Track`, `mo:Record`, and `mo:MusicGroup`. Agent descriptions use FOAF, including `foaf:name` and `foaf:maker`. External identifiers from MusicBrainz and YouTube anchor media objects to existing public knowledge graphs.

Each event receives a deterministic URI derived from the event type, content identifier, and timestamp, for example:

```
urn:combine-ld:watch:<videoId>:<timestamp>
```

This has an important practical benefit. If the same event is imported twice, it receives the same identifier and overwrites the existing event rather than creating a duplicate. This makes imports repeatable and improves recovery after failures.

The cross-service profile slice extends this further. It uses `schema:sameAs` to link a music creator URI to a video creator URI when the user confirms the match. This is the most direct example of semantic interoperability in the prototype: a standard RDF predicate connects two external platform identities, making data written by one service understandable in the context of another without any platform-to-platform exchange.

This decision directly supports CQ1 and CQ3. CQ1 requires media events to be queryable across time. CQ3 requires events from different services to be integrated in a standardised way. JSON-LD and shared vocabularies make this possible, but they do not guarantee interoperability by themselves. Services must still agree on modelling conventions, vocabulary terms, identifier strategies, and edge cases. Vocabulary governance is therefore part of the architectural problem, not merely an implementation detail.

2.8. Event Bus

A media behaviour profile should remain current. If a user listens to music or watches a video, downstream processors should not need to repeatedly scan the full dataset to detect that a new event exists. The architecture therefore requires a change notification mechanism. This mechanism decouples the personal data store from the aggregation processor: the store publishes a change event when data is written, and the processor reacts to that event independently.

The literature-aligned option is Linked Data Event Streams [18]. LDES represents changes as linked data resources that consumers can follow incrementally. This aligns well with RDF and the Solid ecosystem. Other options include Apache Kafka, Apache Pulsar, webhooks, or server-sent events.

The prototype uses Redpanda. Redpanda implements the Kafka protocol while avoiding the operational overhead of a full Kafka cluster. It is practical for a proof of concept because it offers low-latency event streaming and works with existing Kafka client libraries.

In the prototype, Kvasir publishes a change event whenever a `ListenAction` or `WatchAction` is written to a slice. The Change Processor holds an open SSE subscription on Kvasir's `onListenActionAdded` endpoint and processes each new event as it arrives. Kvasir internally backs this subscription with Redpanda, but the processor interacts only with Kvasir's GraphQL subscription API. This supports CQ1 and CQ3 by allowing the profile to be updated incrementally.

One important operational characteristic is that Kvasir SSE subscriptions are forward-looking only. Events written before the subscription was opened are never delivered. The cross-service profile therefore only reflects behaviour that occurred while the processor was actively running. Any historical data that predates a processor restart is not automatically included. A separate backfill step is required to incorporate those events into the profile.

This is a deliberate deviation from LDES. LDES is better aligned with linked data standards and would be preferable for a production deployment that prioritises Solid ecosystem interoperability. Redpanda is better suited to the prototype’s need for low-latency push-based processing. A future standards-aligned implementation could replace Redpanda with LDES without changing the architectural role of the event bus.

2.9. Aggregation Processor

The aggregation processor is the component that gives the profile meaning. It reads permitted events from multiple partitions, enriches them with metadata, derives aggregated insights, and writes the result back to the user’s pod. This component is powerful because it has access to raw behavioural data. Its permissions must therefore be narrow, explicit, and auditable.

The architectural requirement is minimal privilege. The processor should only read the slices it needs, only write to the designated output partition, and operate within user-defined access boundaries. Several technologies could fulfil this role, including Apache Flink, Spark Streaming, or simpler event-driven scripts. The key question is not which processing framework is used, but whether the processor operates within transparent and enforceable limits.

The prototype uses Python change processors. The music processor subscribes to `onListenActionAdded`. When a listening event is added, it extracts the MusicBrainz recording identifier and retrieves metadata such as artist credits, album title, track listing, release date, and release group. It then retrieves cover art and artist images, stores binary assets in MinIO, and writes enriched track, album, and artist entities back to the `music-tracker` slice.

The processor also updates interaction counters. Because RDF triples are not incremented in place, the prototype uses a delete-then-insert pattern. The current counter is read, the existing `schema:InteractionCounter` triple is deleted, and a new triple is inserted with the updated count. The video processor follows the same pattern for videos and channels, subscribing to `onWatchActionAdded`.

For each incoming event, the processor also upserts the corresponding creator entry in the cross-service profile, updating the interaction counter using the same delete-then-insert pattern. The aggregated output is therefore kept current in real time, one creator entry per artist or channel.

The aggregated output is written to the `cross-service profile` partition. This supports CQ3 because data from multiple services can be combined without requiring those services to exchange data directly. It also supports CQ2 because downstream services can receive access to the aggregated profile rather than the raw event history.

Because the event bus does not replay history, the processor has no visibility into events written before it started. To address this, a separate Python function implements a backfill mechanism. It paginates through all historical `listenActions` and `watchActions` using GraphQL cursor pagination, counts interactions per creator, and writes the aggregated results to the cross-service profile in bulk. This step is run once after the initial data import or after a processor restart. During large backfills involving tens of thousands of events, OAuth access tokens may expire before the op-

eration completes. The prototype handles this by refreshing the token explicitly between the video and music phases of the backfill.

The trade-off is trust. The Change Processor determines which derived facts are written to the profile. If its logic is opaque or its permissions are too broad, it could undermine the user-centric guarantees of the architecture. For this reason, aggregation logic must be inspectable, permissions must be limited, and processor activity must be logged.

2.10. Inference Service

A media behaviour profile becomes valuable when it supports useful services such as recommendations, personal analytics, or media trend summaries. However, using the profile creates a new privacy risk. If the profile must be sent to an external recommendation API, the user loses control at the moment the insight is generated.

The architecture therefore requires the inference service to operate with scoped, read-only access and without retaining the profile data it consumes. The inference mechanism itself is flexible. It could be a language model, a collaborative filtering engine, a statistical model, or a rule-based recommender. The architectural constraint is that the service should only access the data it is authorised to read and should not persist unnecessary copies.

The prototype uses Ollama as a locally running inference service. The recommender reads permitted data from the **cross-service profile**, constructs a prompt from the most-watched videos or profile summaries, and sends that prompt to a local model. No watch-history data is transmitted to an external API.

This demonstrates that user-centric architectures do not prevent personalisation. They change the conditions under which personalisation happens. Instead of platforms accumulating behavioural data centrally, recommendation services access only the authorised profile data needed for a specific purpose.

2.11. Policy Layer

Access control answers whether a service may access a resource at the moment of the request. It does not fully answer what the service may do with the data after access has been granted. For media behaviour data, this distinction is essential. A service may be allowed to read a profile for personalisation, but not to use it for advertising, sell it to third parties, or retain it indefinitely.

This requires a policy layer that expresses usage conditions. ODRL [17] is the appropriate standard for this role because it can describe permissions, prohibitions, and duties. For example, a policy may permit a recommender to use viewing history for personalisation, prohibit sharing it with advertising networks, and require deletion after a defined period. Trustflows [21] provides a governance-oriented framework for making such data flows trustworthy across autonomous stakeholders. The Data Privacy Vocabulary can complement ODRL by expressing processing purposes and legal bases under the GDPR.

In the conceptual architecture, the policy layer sits between consuming services and the user's pod. It evaluates not only whether access is allowed, but also whether the

intended use matches the user's consent and applicable conditions.

In the prototype, this layer is not implemented. OpenFGA provides access control and demonstrates selective sharing, but it does not enforce post-access usage restrictions. The prototype therefore validates storage, partitioning, identity, authorisation, aggregation, inference, and logging, but it does not fully validate usage control beyond the pod boundary.

This limitation is important. The proof of concept shows that user-centric media profile management is technically feasible, but it should not be interpreted as a complete production-ready governance system. A production deployment would need an ODRL-based policy layer, usable consent interfaces, contractual safeguards, and audit mechanisms that can verify whether stated usage conditions were respected.

2.12. Audit Layer

A user-centric architecture must not only enforce access rules. It must also make data flows inspectable after the fact. Users, regulators, and auditors need evidence of what happened: which service accessed which data, when, under which authorisation, and for which stated purpose. Without auditability, users may technically control access but have no practical way to verify whether that control was respected.

The audit layer can be implemented in different ways. Logs may be stored inside the user's pod, giving the user direct access to their data-flow history. An external audit service could provide stronger oversight. A tamper-evident append-only log could be used in higher-assurance regulatory contexts. The minimum requirement is that access and change events are recorded with enough detail to support accountability.

The prototype supports auditability at three levels. Keycloak records authentication events, showing when services obtain access tokens. OpenFGA records authorisation decisions, showing which access checks were performed. Kvasir and the Change Processor record change events, showing which data was written and which derived updates were produced.

These logs make the flow from ingestion to enrichment to consumption traceable. However, audit logs are retrospective. They do not prevent misuse by themselves. They provide evidence after the fact. Their value increases when combined with preventive access control and the future ODRL policy layer. f ### Prototype Deviations from the Solid-Native Design

The prototype is Solid-based in architectural orientation, but not purely Solid-native in every operational mechanism. This distinction is important for interpreting the proof of concept correctly.

The architecture follows Solid principles by placing data in a user-controlled pod, representing information as linked data, supporting Solid-OIDC through Kvasir's Solid endpoint, and attaching access decisions to the user's data environment rather than to individual media platforms. However, the prototype introduces implementation-specific mechanisms to support high-volume media behaviour processing.

Table 2: Prototype deviations from a purely Solid-native implementation, with the reason for each deviation.

Concern	Solid-native option	Prototype option	Reason for deviation
Resource access	LDP HTTP resources	Kvasir GraphQL and JSON-LD deltas	More expressive querying for structured media profiles
Access control	WAC or ACP	OpenFGA	Kvasir does not yet implement WAC/ACP on the Solid endpoint
Change notification	LDES	Redpanda	Low-latency push-based processing for the prototype
Service authentication	Solid-OIDC browser flow	Keycloak `client_credentials`	Automated background services need non-interactive authentication
Usage control	ODRL policy enforcement	Conceptual only	Left as future work beyond the proof of concept

These deviations do not invalidate the architecture, but they define the scope of the prototype. The prototype demonstrates that the architectural roles can be assembled into a working system. It does not claim to be a fully standards-complete Solid deployment.

3. Implementation

This chapter demonstrates the architecture described in the previous chapter through a concrete proof-of-concept. The prototype extends the `kvasir-music-tracker` example application published by imec-IDLab. That application provides a Spotify client, a Python change processor that fetches MusicBrainz metadata via SSE subscriptions, a basic Vue.js web application for browsing recently played tracks, and a Docker Compose environment preconfigured for a demo user `alice`. This thesis extends that foundation by adding a `video-tracker` slice and change processor, a cross-service profile slice, user-controlled identity linking between music and video creators, a local inference component, a significantly expanded web application, and a browser extension. The browser extension is an extension for Firefox that captures events directly from YouTube and SoundCloud. On YouTube it detects watch events via DOM observation and the `yt-navigate-finish` event. On SoundCloud it detects listen events by observing the player badge for track changes. Both content scripts relay events to a background script that forwards them to the pod as JSON-LD change requests. The Spotify client from the original demo remains functional alongside the extension, giving the prototype three independent event sources: Spotify, YouTube, and SoundCloud. The chapter walks through the complete data flow: from defining a slice schema and registering a pod, through writing a media event, to the enrichment and aggregation steps that produce the cross-service profile. Two slices are used throughout: `music-tracker` for listening behaviour and `video-tracker` for watch behaviour.

3.1. 1. Define the Slice Schema

Each slice is defined by a GraphQL SDL schema. The schema declares which types are queryable, which mutations are available, and which subscriptions clients can open. Two vocabularies are combined in the music slice: `schema.org` for interactions and the Music Ontology for tracks, records, and groups.

The music tracker schema exposes four queryable types and a subscription that fires whenever a new listen event is written:

Listing 5: GraphQL SDL schema for the `music-tracker` slice, combining `schema.org` and Music Ontology vocabularies. The `@generateMutations` directive auto-generates `add` mutations for every queryable type.

```
type Query @generateMutations(operations: ["add"]) {
  listenActions(id: [ID], pageSize: Int = 100, cursor: String,
orderBy: [String]): [schema_ListenAction!]
  tracks(id: [ID], pageSize: Int = 100, cursor: String, orderBy:
[String]): [mo_Track!]
  albums(id: [ID], pageSize: Int = 100, cursor: String, orderBy:
[String]): [mo_Record!]
  artists(id: [ID], pageSize: Int = 100, cursor: String, orderBy:
[String]): [mo_MusicGroup!]
}

type Subscription {
  onListenActionAdded: schema_ListenAction!
}

type schema_ListenAction {
  id: ID!
  schema_startTime: String!
  schema_agent: ID!
  schema_object: mo_Track!
}

type mo_Track {
  id: ID!
  dc_title: String
  mo_track_number: String
  mo_duration: Int
  foaf_maker: [mo_MusicGroup!]
  schema_interactionStatistic: [schema_InteractionCounter!]
}
```

The `@generateMutations(operations: ["add"])` directive on the `Query` type instructs Kvasir to auto-generate `add` mutations for every queryable type. The `schema_InteractionCounter` type additionally carries `remove` to support the delete-then-insert counter update pattern used by the change processor.

The video tracker schema follows the same structure using `schema.org` throughout:

Listing 6: GraphQL SDL schema for the `video-tracker` slice, using `schema.org` vocabulary throughout.

```
type Query @generateMutations(operations: ["add"]) {
  watchActions(id: [ID], pageSize: Int = 100, cursor: String,
orderBy: [String]): [schema_WatchAction!]
  videos(id: [ID], pageSize: Int = 100, cursor: String, orderBy:
[String]): [schema_VideoObject!]
  channels(id: [ID], pageSize: Int = 100, cursor: String,
orderBy: [String]): [schema_Person!]
}

type Subscription {
  onWatchActionAdded: schema_WatchAction!
}
```

The cross-service profile slice introduces a `csp_Creator` type that holds aggregated interaction counts alongside confirmed cross-platform identity links:

Listing 7: GraphQL SDL schema for the `cross-service-profile` slice. The `schema_sameAs` field stores user-confirmed cross-platform identity links, `schema_interactionStatistic` accumulates counts across both music and video slices, and `csp_UserInsight` stores user-validated behavioural assertions.

```
type Query @generateMutations(operations: ["add"]) {
  creators(id: [ID], pageSize: Int = 100, cursor: String,
orderBy: [String]): [csp_Creator!]
  insights(id: [ID], pageSize: Int = 100, cursor: String,
orderBy: [String]): [csp_UserInsight!]
}

type csp_Creator {
  id: ID!
  schema_name: String
  schema_sameAs: [ID]
  schema_isBasedOn: String
  schema_interactionStatistic: [csp_InteractionCounter!]
}

type csp_InteractionCounter @generateMutations(operations:
["add", "remove"]) {
  id: ID!
  schema_interactionType: ID!
  schema_userInteractionCount: Int!
}

type csp_UserInsight @generateMutations(operations: ["add",
"remove"]) {
  id: ID!
  schema_description: String!
  schema_name: String!
  schema_actionStatus: String!
  schema_dateCreated: String!
}
```

The `schema_sameAs` field links a music creator URI to a video creator URI once the user has confirmed the match. The `schema_interactionStatistic` field accumulates interaction counts aggregated across both the music and video slices.

3.2. 2. Bootstrap the Pod and Register Slices

The pod for user `alice` is created automatically when Kvasir starts, using the bootstrap configuration in `kvasir-config/application.yaml`. The configuration sets a default JSON-LD context covering `schema.org`, Music Ontology, Dublin Core, and FOAF, and registers five Keycloak clients:

- `music-tracker`: a browser-facing client used by the web application
- `music-tracker-client`: a service account used by the change processor, with reader, writer, and deleter access to the `music-tracker` slice, writer access to the MinIO cover-art bucket, and writer and deleter access to the `cross-service-profile` slice for upserting creator entries
- `video-tracker`: a browser-facing client
- `video-tracker-client`: a service account with reader, writer, and deleter access to the `video-tracker` slice and writer and deleter access to the `cross-service-profile` slice
- `recommender-client`: a service account used by the inference service, with read-only access to the `cross-service-profile` slice

The OpenFGA relationships are declared inline in the bootstrap configuration:

Listing 8: Bootstrap configuration declaring OpenFGA relationships for `music-tracker-client`, granting read, write, and delete access to the `music-tracker` slice, write access to the MinIO cover-art bucket, and write and delete access to the `cross-service-profile` slice.

```
- client-id: music-tracker-client
  enable-service-account: true
  openfga:
    relationships:
      - target-resource: "/slices/music-tracker"
        relations: ["reader", "writer", "deleter"]
      - target-resource: "/s3/music-tracker/cover-art"
        relations: ["reader", "writer"]
      - target-resource: "/slices/cross-service-profile"
        relations: ["writer", "deleter"]
```

This means that when Kvasir starts, the pod, the clients, and all access-control relationships already exist, requiring no manual setup.

The slices themselves are registered by POSTing the SDL schema to the pod's `/slices` endpoint. On success Kvasir returns `201 Created` with the slice URL in the `Location` header.

3.3. 3. Authenticate as a Service

All Python services authenticate using the OAuth 2.0 `client_credentials`

flow. The service exchanges its `client_id` and `client_secret` for a bearer token from Keycloak, then attaches the token to every subsequent request. Kvasir validates the token and asks OpenFGA whether the token's subject has the required role on the target slice before executing the operation.

3.4. 4. Write a Media Event

A media client writes a `ListenAction` or `WatchAction` by posting a JSON-LD delta to the slice's changes endpoint. Each event receives a deterministic URI constructed from the event type, content identifier, and timestamp:

`urn:combine-ld:watch:<videoId>:<timestamp>`

An example listen event for a MusicBrainz recording:

Listing 9: A `schema:ListenAction` event in JSON-LD format, linking a user agent to a MusicBrainz recording URI. The deterministic subject URI is derived from the event type, recording identifier, and timestamp.

```
{
  "@context": {
    "schema": "http://schema.org/",
    "mo": "http://purl.org/ontology/mo/"
  },
  "@id": "urn:combine-ld:listen:b36c079c-79e4-4f9e-b4f9-cbb6a3d4d5a1:2024-11-15T20:32:00Z",
  "@type": "schema:ListenAction",
  "schema:startTime": "2024-11-15T20:32:00Z",
  "schema:agent": { "@id": "http://localhost:8080/alice" },
  "schema:object": {
    "@id": "https://musicbrainz.org/recording/b36c079c-79e4-4f9e-b4f9-cbb6a3d4d5a1",
    "@type": "mo:Track"
  }
}
```

Because the URI is deterministic, re-importing the same event overwrites the existing triple rather than creating a duplicate. Fig. 13 shows the committed change report for one such event, displaying the six RDF triples written to the `music-tracker` slice.

Kvasir

GraphQLS3 BrowserChangesSlices

alice (logout)

Change Details

Change Report ID

Pod ID

Inserts

Result Code

http://localhost:8080/alice/slices/music-tracker/changes/2SawMT1ZjYTFmMzgzZDZkYjBhMjZmYyYyYyNDY3Nw==

http://localhost:8080/alice

6

COMMITTED

Slice ID

Deletes

Timestamp

http://localhost:8080/alice/slices/music-tracker

16 Mar, 2026 @ 02:16:07

Records

Sign	Subject	Predicate	Object
+	https://musicbrainz.org/recording/e54bd471b9c2-4f71-a22c-a89d998c73f	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://purl.org/ontology/mo/Track
+	um:spotify-play-4d8adeec-a058-4a0e-83f1-a271e8d72a50	http://schema.org/agent	http://example.org/alice
+	um:spotify-play-4d8adeec-a058-4a0e-83f1-a271e8d72a50	http://schema.org/object	https://musicbrainz.org/recording/e54bd471b9c2-4f71-a22c-a89d998c73f
+	um:spotify-play-4d8adeec-a058-4a0e-83f1-a271e8d72a50	http://schema.org/provider	https://www.spotify.com
+	um:spotify-play-4d8adeec-a058-4a0e-83f1-a271e8d72a50	http://schema.org/startTime	2026-03-16T13:16:07Z
+	um:spotify-play-4d8adeec-a058-4a0e-83f1-a271e8d72a50	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/ListenAction

Kvasir ©2024 by IDLab - Discover

6.9.32025-09-26 13:27:23

Fig. 13: A committed change report in Kvasir’s Changes view for the **music-tracker** slice. The six inserted triples record the track as a **mo:Track** linked to its MusicBrainz URI, and the listen event as a **schema:ListenAction** with agent, object, provider, and timestamp. The deterministic subject URI follows the **urn:combine-ld:listen:** pattern.

3.5. 5. The Change Processor Reacts

When the event reaches Kvasir, it is committed to ClickHouse and a change message is published to the Redpanda topic for the **music-tracker** slice. The change processor, which holds an open SSE subscription on **onListenActionAdded**, receives the event and begins enrichment.

The processor extracts the MusicBrainz recording ID from the track URI and calls the MusicBrainz API to retrieve the full release metadata: artist credits, album title, release date, track listing, and release group. It then:

1. Fetches the album cover art and stores it in the MinIO bucket if it does not already exist, storing the resulting S3 URI as **foaf:depiction** on the release group.
2. Fetches an artist thumbnail from the Wikipedia API and stores it as **foaf:depiction** on the artist.
3. Writes the enriched track, album, and artist objects back to the slice via change requests.
4. Updates the play counters for the track, album, and all credited artists.

3.6. 6. Update Counters with Delete-Then-Insert

Play and watch counts are stored as **schema:InteractionCounter** triples attached to each track, album, artist, video, or channel. Because RDF triples cannot be incremented in place, the processor uses a delete-then-insert pattern driven by a **with**-clause that first reads the current counter value:

Listing 10: Delete-then-insert pattern for updating `schema:InteractionCounter` triples. The `with`-clause reads the current count before the delete and insert operations are executed.

```
with-clause:
  read current schema_userInteractionCount
  for the track, album, and artists

delete:
  remove existing InteractionCounter triples

insert:
  write new InteractionCounter triples
  with count + 1 (or 1 if no counter existed)
```

The same pattern is used by the watch processor for video and channel watch counts.

One consequence of using SSE subscriptions is that they deliver only events written after the subscription was opened. If the processor was not running when a batch of events was imported, those events are invisible to it. In the prototype, clearing the ClickHouse storage while the processor was offline caused the cross-service profile to become empty. New events subsequently produced correct profiles, but the historical behaviour was absent until a manual backfill was run.

3.7. 7. Query the Enriched Profile

After the change processor has committed the enrichment, the slice can be queried for the aggregated profile. For example, to retrieve the top-played artists:

Listing 11: GraphQL query against the `music-tracker` slice returning all artists ordered by interaction count, including their name, depiction, and aggregated play count.

```
{
  artists(orderBy:
    ["schema_interactionStatistic.schema_userInteractionCount desc"])
  {
    id
    foaf_name
    foaf_depiction
    schema_interactionStatistic {
      schema_userInteractionCount
    }
  }
}
```

The recommender reads aggregated creator data from the cross-service profile, constructs a prompt from the most-watched channels and most-listened artists, and sends that prompt to Ollama for inference, keeping all profile data within the local environment at all times. Fig. 14 shows a live query against the `video-tracker` slice, demonstrating that recent watch events are stored and retrievable through the GraphQL interface.

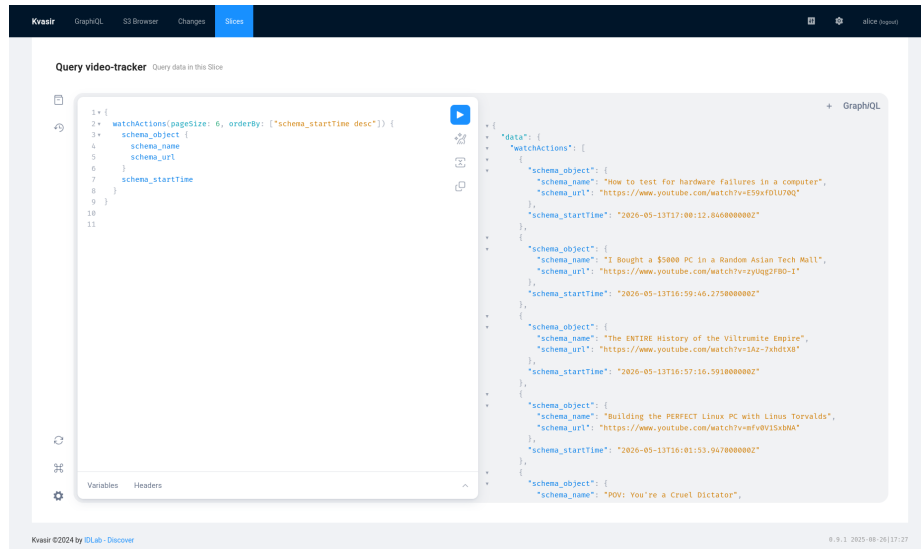


Fig. 14: A GraphQL query against the **video-tracker** slice returning the six most recent watch events with video title, URL, and timestamp. The query is executed through Kvasir’s built-in GraphQL interface, authenticated as user **alice**.

3.8. 8. Cross-Service Identity Linking

The web application includes a “Link Profiles” view where users can review and confirm fuzzy identity matches between music creators and video creators. The matching heuristic compares creator names from the **music-tracker** and **video-tracker** slices. When names are similar enough, a suggested link is surfaced in the UI.

When a user confirms a match, the application writes a **schema:sameAs** triple to the **cross-service profile** slice, linking the music creator URI to the corresponding video creator URI. The pod thereby becomes the authoritative source of cross-platform identity rather than the heuristic matcher.

A separate “Unified” view reads these confirmed **schema:sameAs** links to display cross-platform activity. It combines aggregated listening and viewing counts for creators whose identity has been explicitly verified by the user, joining data from both slices through the user-authored link.

One practical detail affects name-based matching. Some music platforms return comma-separated multi-artist strings, for example “Kanye West, Ye, Andre Troutman” for a collaborative track. The prototype captures only the primary artist name to ensure stable matching. Full multi-artist attribution is retained in the track metadata, but only the first credited artist contributes to the identity link candidate list.

The prototype implements automatic identity resolution via Wikidata’s public SPARQL endpoint. When the change processor handles a SoundCloud or MusicBrainz listen event, it queries Wikidata using the SoundCloud slug (P3040) or MusicBrainz artist ID (P434) to retrieve the corresponding YouTube channel ID (P2397). The watch processor performs the reverse: a YouTube channel ID is used to

look up its SoundCloud slug. When a match is found, `schema:sameAs` is written automatically without user intervention, and `schema:isBasedOn` stores the Wikidata entity URI as provenance. The UI distinguishes Wikidata-verified links from user-confirmed ones through separate visual indicators. User confirmation remains the fallback for creators not present in Wikidata.

One limitation remains: creator identity links are stored per user. Two users observing the same cross-platform identity would each hold an independent copy. For well-known public creators this is mitigated by Wikidata resolution, which produces identical links for all users. For obscure creators, the per-user model reflects that cross-platform identity is genuinely a personal assertion rather than a shared fact.

3.9. 10. User-Validated Behavioural Insights

The system includes a feedback loop that grounds personalisation in explicit user validation. The inference service generates behavioural hypotheses from the aggregated cross-service profile, surfaces them as questions in the web application, and stores the user's validated answers as RDF in the pod. These stored answers are incorporated into subsequent recommendation prompts, allowing the profile to accumulate user-authored assertions alongside passively collected interaction data.

Insight generation. The `/insights` endpoint of the inference service queries the `cross-service-profile` slice for the top-watched video channels and most-listened music artists, and computes a set of context metrics: the number of creators active on each platform, the share of total activity concentrated in the top three creators, and the set of creators with confirmed cross-platform identity links. These signals are assembled into a structured prompt sent to Ollama, which returns a JSON array of insight objects. Each object contains a statement (a behavioural claim phrased declaratively, for example “You primarily watch technology content”) and a question (a yes/no reformulation presented to the user).

Storing validated answers. When the user responds, the web application posts the question, the answer text, and a status value to the `/insights/answer` endpoint. The endpoint writes a `csp:UserInsight` node to the `cross-service-profile` slice via a JSON-LD change request. The `schema:actionStatus` field distinguishes three cases: `confirmed` (the user agreed with the claim), `denied` (the user rejected it), and `corrected` (the user provided an alternative). The write uses the `video-tracker-client` service account, which holds writer access to `cross-service-profile`. The `csp_UserInsight` type is defined in the updated `cross-service-profile` SDL shown in Listing 7.

Feeding back into recommendations. Both recommendation endpoints retrieve stored insights before constructing the prompt. Confirmed insights are presented as established user preferences. Denied insights are explicitly marked as claims the user has rejected, preventing the model from reproducing them in subsequent sessions. Corrected insights present the original assumption alongside the user's correction. This allows the model to reason from a user-validated profile rather than inferred behaviour alone, and prevents the same incorrect assumption from recurring across sessions.

3.10. Import Rate Limiting

The initial data import sent five concurrent requests per batch with no delay between batches. At 42,000 records this produced approximately 8,400 batches in under a minute, far exceeding Kvasir's processing capacity of around 66 records per second. The result was a flooded Kafka topic and a backlog that Kvasir could not clear in time.

The fix was to add a 150-millisecond pause between batches. This reduces the effective ingestion rate to approximately 33 records per second, comfortably below Kvasir's capacity and leaving headroom for other operations during import.

A side effect of the original flood was that resetting the Kafka consumer offset to recover from the crash caused approximately 29,000 queued messages to be skipped. Only 13,044 of the original 42,000 records reached ClickHouse. Because each record uses a deterministic URI as its RDF subject identifier, the import can be re-run safely without producing duplicates: records already present in ClickHouse are overwritten with identical data, while missing records are inserted fresh.

4. Evaluation

This chapter evaluates the proof-of-concept prototype against the competency questions defined in the architecture chapter and assesses how well the implemented system fulfils the five requirement areas established in the literature study.

4.1. Competency Question Assessment

The three competency questions were formulated before the architecture was designed and serve as the functional test for the system. Table 3 summarises the assessment for each question.

Table 3: Assessment of the prototype against the three competency questions.

Competency question	Supported?	Evidence	Limitation
CQ1: Which content has a given user consumed within a specified time period?	Yes	GraphQL queries on the <code>music-tracker</code> and <code>video-tracker</code> slices return listen and watch events ordered by timestamp. The implementation chapter demonstrates a working query returning the six most recent watch events through Kvasir’s GraphiQL interface.	No native SPARQL query layer. All queries must be issued through Kvasir’s GraphQL API.
CQ2: Which services may access which parts of a user’s consumption history, and under what conditions?	Partially	OpenFGA enforces per-slice access control. Services hold explicit relationships (<code>reader</code> , <code>writer</code> , <code>deleter</code>) on individual slices. A music tracker client cannot read the video slice.	No ODRL-based usage control is implemented. The prototype enforces access at the pod boundary but cannot enforce post-access conditions such as retention limits or restrictions on secondary use.
CQ3: How can consumption data from multiple platforms be integrated in a standardised way?	Partially	Both slices use JSON-LD with shared schema.org vocabulary. The cross-service profile uses <code>schema:sameAs</code> to link music and video creator identities confirmed by the user. For well-known creators, links are resolved automatically via Wikidata SPARQL and stored with <code>schema:isBasedOn</code> provenance. The aggregation processor derives unified interaction counts across both slices.	Interoperability depends on vocabulary agreements between services. The prototype enforces these agreements internally, but a production deployment requires ecosystem-wide adoption of shared vocabularies and identifier conventions.

4.2. Expert Feedback

The competency questions were assessed through a structured written questionnaire

completed by a practitioner from the Flemish media sector. The respondent rated CQ1 as the most relevant, reflecting the practical importance of queryable consumption history for personalisation and editorial research. CQ2 was rated highly, confirming that user-controlled selective disclosure is a meaningful requirement from the media sector's perspective. CQ3 was rated moderately: not because cross-service integration is unimportant, but because ecosystem adoption is the primary barrier rather than technical feasibility.

This assessment aligns with the prototype's own profile. The system performs well on CQ1 and demonstrates the access control mechanisms required for CQ2. CQ3 is the most aspirational of the three. The architecture provides the vocabulary and linking mechanisms, but realising it at scale requires coordinated participation from independent media services that currently have no structural incentive to align on shared standards.

4.3. Requirement Area Coverage

The five requirement areas identified in the literature study provide a second evaluation frame. The prototype addresses four of them in full or in part.

Personal data storage is implemented through Kvasir, with per-slice partitioning and independent access control per slice. The shared identity mechanism is implemented through Keycloak for service accounts and through Kvasir's Solid-OIDC endpoint for user-facing authentication. The aggregation component is implemented through the Python change processors, with real-time updates driven by SSE subscriptions and a backfill mechanism for historical data. Auditability is partially implemented: Keycloak records authentication events, OpenFGA records authorisation decisions, and Kvasir records change events for each slice.

The policy layer is the only requirement area not implemented in the prototype. ODRL-based usage control within the Trustflows framework remains as future work. This is the most significant gap between the conceptual architecture and the prototype, and it means the current system validates access governance but not post-access usage governance.

5. Conclusion

This thesis set out to answer the following research question:

Which key design decisions need to be made when building media behaviour profiles across multiple media services?

The research identified ten design decisions that any system addressing this question must resolve. These concern where data is stored, how it is partitioned, how actors are identified, how access is authorised, how media events are represented, how changes are propagated, how raw events are aggregated, how the resulting profile is consumed, how usage conditions are expressed, and how access is audited. Each decision was first defined at an abstract level as an architectural role with a specific responsibility, then implemented in a concrete proof-of-concept prototype.

The prototype demonstrates that a user-centric cross-service media behaviour profile system is technically feasible. A personal data pod, governed by per-slice access control, semantic data representation, and a user-authorised aggregation pipeline, can store and integrate media behaviour data from independent services without those services sharing infrastructure or data directly. Local inference allows personalisation to occur without transmitting behavioural data to external APIs.

The central architectural finding is that a pod should not be limited to storage alone. The paper on What's in a Pod [29] argues that a pod is better understood as a hybrid, contextualised knowledge graph from which multiple views and APIs can be derived, rather than a single document hierarchy. This thesis gives that insight a concrete application: the prototype demonstrates that a pod must also support user-authored assertions. Confirmed cross-platform identity links, validated behavioural insights, and governed access rules are all data the user writes into the pod to define how their profile is interpreted and shared. The user is thereby not a passive data subject managed by individual platforms, but an active integration point who authors the semantic connections that make cross-service profiling possible.

The main limitation of the prototype is its depth of coupling to Kvasir-specific mechanisms. GraphQL querying, schema-governed slices, ClickHouse-backed analytical storage, and Redpanda-based event streaming are Kvasir features that a standard Solid pod does not provide. The architectural roles are separable in principle, but migrating to a different Solid server would require significant rewriting of the implementation. The ODRL-based policy layer also remains unimplemented. The prototype enforces access at the pod boundary but applies no usage conditions to data after it has been read.

Future work should address three areas. First, adding ODRL-based usage control within the Trustflows framework would bring the prototype closer to production data space practice. Production implementations such as Eclipse Dataspace Components [20], the technical backbone of Gaia-X and data spaces such as Catena-X, enforce ODRL policies at the transfer boundary: before data leaves the source, both parties negotiate a policy agreement, and the transfer is refused if the receiving party does not satisfy the stated conditions. Adding a connector-level policy negotiation layer around Kvasir slice access would achieve this and is primarily engineering work. The genuinely hard open problem is post-transfer enforcement: once data has been received,

no technical mechanism currently prevents a service from retaining it past a stated expiry or using it for purposes beyond what was agreed. Enforcement at that point is contractual and audit-based, not technical. Completing the Trustflows integration addresses the boundary layer. Post-transfer enforcement remains an open research problem across all current data space implementations. Trustflows is introduced in this thesis as a core building block alongside Solid, but the prototype only realises the Solid side of that pairing. Second, replacing Redpanda with LDES would align the change notification mechanism with Solid and linked data standards, improving portability and long-term interoperability. Third, the prototype implements Wikidata-based identity resolution for known creators. Extending this to contribute verified links back to Wikidata, closing the knowledge graph contribution loop, remains future work.

The main contribution of this thesis is a structured design-decision framework for user-centric cross-service media behaviour profiling, validated through a proof-of-concept implementation and assessed through expert feedback from the Flemish media sector.

The broader challenge is sociotechnical. The prototype shows that the architecture works. The larger question is whether independent media services, infrastructure providers, users, and regulators can align around common standards, governance frameworks, and incentive structures at sufficient scale. The expert questionnaire confirmed that practitioners in the Flemish media sector see the concept as feasible in the medium term, but identify ecosystem adoption as the primary barrier. This thesis addresses the technical side of that challenge. The social, regulatory, and commercial dimensions remain open problems.

References

- [1] Vlaamse overheid, “Mediagebruik,” Statistiek Vlaanderen, 2024. [Online]. Available: <https://www.vlaanderen.be/statistiek-vlaanderen/media-en-mediagebruik/mediagebruik>. [Accessed: Mar. 17, 2026].
- [2] M. Maes, A. Bourgeois, S. Van Damme, and R. De Wolf, SOLID Monitor 2024: Privacy, persoonlijke data & datakluisen, 3rd ed. imec, 2024. [Online]. Available: https://solidlab.be/wp-content/uploads/2025/01/SM_2024_online_version.pdf. [Accessed: Mar. 17, 2026].
- [3] imec, “Data mogen niet het nieuwe goud zijn,” Jan. 3, 2022. [Online]. Available: <https://www.imec.be/nl/articles/data-mogen-niet-het-nieuwe-goud-zijn>. [Accessed: Mar. 18, 2026].
- [4] European Commission, “Digital Europe Programme.” [Online]. Available: <https://digital-strategy.ec.europa.eu/en/activities/digital-programme>. [Accessed: Mar. 18, 2026].
- [5] Gaia-X European Association for Data and Cloud AISBL, “About Gaia-X,” Gaia-X. [Online]. Available: <https://gaia-x.eu/about/>. [Accessed: May 9, 2026].
- [6] C. Duhigg, “How Companies Learn Your Secrets,” *The New York Times Magazine*, Feb. 16, 2012. [Online]. Available: <https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html>. [Accessed: Apr. 20, 2026].
- [7] J. Spooner, “Conway’s Law,” Sketchplanations. [Online]. Available: <https://sketchplanations.com/conways-law>. [Accessed: Apr. 15, 2026].
- [8] Art. 17 GDPR Right to erasure (‘right to be forgotten’). [Online]. Available: <https://gdpr-info.eu/art-17-gdpr/>. [Accessed: Apr. 20, 2026].
- [9] Solid Project, “Home.” [Online]. Available: <https://solidproject.org/about>. [Accessed: Apr. 15, 2026].
- [10] Solid Project, “Solid: Your data, your choice.” [Online]. Available: <https://solid-project.org/>. [Accessed: Apr. 15, 2026].
- [11] W3C, “WebAccessControl.” [Online]. Available: <https://www.w3.org/wiki/WebAccessControl>. [Accessed: Apr. 15, 2026].
- [12] Solid Community Group, *Access Control Policy (ACP)*, W3C Solid Community Group Draft Report. [Online]. Available: <https://solidproject.org/TR/acp>. [Accessed: Apr. 15, 2026].
- [13] A. Coburn, elf Pavlik, and D. Zagidulin, *Solid-OIDC*, W3C Solid Community Group Editor’s Draft. [Online]. Available: <https://solidproject.org/TR/oidc>. [Accessed: Apr. 15, 2026].
- [14] Solid Community Group, *WebID 1.0: Web Identity and Discovery*, W3C Editor’s Draft. [Online]. Available: <https://www.w3.org/2005/Incubator/webid/spec/identity/>. [Accessed: Apr. 20, 2026].
- [15] imec-IDLab, Ghent University, “Community Solid Server.” [Online]. Available:

<https://github.com/CommunitySolidServer/CommunitySolidServer>. [Accessed: Apr. 20, 2026].

[16] imec-IDLab, “Kvasir Server Documentation - What is Kvasir?” [Online]. Available: <https://kvasir.pages.ilabt.imec.be/kvasir-server/what-is-kvasir.html>. [Accessed: Apr. 20, 2026].

[17] R. Iannella and S. Villata, Eds., *ODRL Information Model 2.2*, W3C Recommendation, Feb. 15, 2018. [Online]. Available: <https://www.w3.org/TR/odrl-model/>. [Accessed: Apr. 20, 2026].

[18] P. Colpaert *et al.*, *Linked Data Event Streams (LDES)*, SEMIC / IDLab, Ghent University - imec. [Online]. Available: <https://semiceu.github.io/LinkedDataEventStreams/>. [Accessed: Apr. 20, 2026].

[19] European Commission, *A European Strategy for Data*, Communication COM(2020) 66 final, Feb. 19, 2020. [Online]. Available: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52020DC0066>. [Accessed: Apr. 20, 2026].

[20] Eclipse Foundation, *Eclipse Dataspace Protocol Specification*. [Online]. Available: <https://projects.eclipse.org/projects/technology.dataspace-protocol-base>. [Accessed: Apr. 20, 2026].

[21] Trustflows.eu, “Meetups, use cases and resources about trustworthy data flows.” [Online]. Available: <https://trustflows.eu/>. [Accessed: Apr. 24, 2026].

[22] P. Colpaert, “Eventual Interoperability,” Pieter Colpaert, Jan. 8, 2026. [Online]. Available: <https://pietercolpaert.be/interoperability/2026/01/08/eventual-interoperability>. [Accessed: Apr. 24, 2026].

[23] A. Poikola, K. Kuikkaniemi, O. Kuittinen, H. Honko, A. Knuutila, V. Lähteenoja, A. Bowyer, V. Lähteenoja, and C. Wilson, “MyData in Motion: Evolving Empowerment for 2025 and beyond,” 4th ed., MyData Global, 2025. [Online]. Available: <https://mydata.org/wp-content/uploads/2025/05/MyData-in-Motion-Evolving-Empowerment-for-2025-and-beyond-layout-v4-1.pdf>. [Accessed: May. 9, 2026].

[24] J. Van Herwegen and R. Verborgh, “The Community Solid Server: Supporting research & development in an evolving ecosystem,” *Semantic Web*, vol. 15, no. 6, pp. 2597–2611, 2024. DOI: <https://doi.org/10.3233/SW-243726>.

[25] M. Ragab, Y. Savateev, H. Oliver, T. Tiropanis, A. Poulouvassilis, A. Chapman, R. Taelman, and G. Roussos, “Decentralized Search over Personal Online Datastores: Architecture and Performance Evaluation,” in *Proc. 24th Int. Conf. Web Engineering (ICWE 2024)*, Lecture Notes in Computer Science, vol. 14629, pp. 49–64, Springer, 2024. DOI: https://doi.org/10.1007/978-3-031-62362-2_4.

[26] C. Esposito, O. Hartig, R. Horne, and C. Sun, “Assessing the Solid Protocol in Relation to Security & Privacy Obligations,” *Information*, vol. 14, no. 7, p. 411, 2023. DOI: <https://doi.org/10.3390/info14070411>.

[27] S. Speicher, J. Arwe, and A. Malhotra, Eds., *Linked Data Platform 1.0*, W3C Recommendation, Feb. 26, 2015. [Online]. Available: <https://www.w3.org/TR/ldp/>. [Accessed: May 15, 2026].

[28] ClickHouse, Inc., “ClickHouse.” [Online]. Available: <https://clickhouse.com/>. [Accessed: May 16, 2026].

[29] R. Dedecker, W. Slabbinck, J. Wright, P. Hochstenbach, P. Colpaert, and R. Verborgh, “What’s in a Pod? A knowledge graph interpretation for the Solid ecosystem,” in *Proc. Workshop on Storing, Querying and Benchmarking Knowledge Graphs (QuWeDa 2022)*, co-located with ISWC 2022, 2022. [Online]. Available: <https://solid-labresearch.github.io/WhatsInAPod/>. [Accessed: May 27, 2026].

7. Use of Generative AI

AI tools were used in the preparation of this thesis in three ways.

Text editing and review. Claude by Anthropic and ChatGPT by OpenAI were used as writing assistants throughout the drafting process. They were used to rephrase sentences, improve clarity, identify inconsistencies between sections, and check that technical claims were accurately stated. All content, arguments, design decisions, and conclusions are the author's own. The AI tools did not generate original analysis or interpret research results.

Diagram creation. Several figures in this thesis were created or styled with AI assistance. Figures 1 and 9 were generated with AI assistance and subsequently reviewed, adjusted, and approved by the author before export.

Code assistance. Claude Code and ChatGPT were used during development of the proof-of-concept prototype to assist with boilerplate code, syntax, and debugging. All architectural decisions, data models, and integration logic were designed, implemented, and verified by the author.