FHIR Infrastructure Reference Architecture

Contents

[1. Introduction 2](#_Toc164963296)

[2. Architecture Components 3](#_Toc164963297)

[2.1. Data Serving Components 3](#_Toc164963298)

[2.1.1. FHIR Client 3](#_Toc164963299)

[2.1.2. API Gateway 3](#_Toc164963300)

[2.1.3. FHIR Validator 4](#_Toc164963301)

[2.1.4. .well-known 4](#_Toc164963302)

[2.1.5. Authorization Server 5](#_Toc164963303)

[2.1.6. Public Key Directory 5](#_Toc164963304)

[2.1.7. Interceptor 5](#_Toc164963305)

[2.1.8. FHIR Server 7](#_Toc164963306)

[2.1.9. Permissions Management System 7](#_Toc164963307)

[2.2. Shared Components 7](#_Toc164963308)

[2.2.1. Audit and Logging 7](#_Toc164963309)

[2.3. Data Population Components 7](#_Toc164963310)

[2.3.1. Batch Orchestrator 7](#_Toc164963311)

[2.3.2. Terminology Server 8](#_Toc164963312)

[2.3.3. FHIR Mapping Engine 8](#_Toc164963313)

[2.3.4. Security Labeling Service 8](#_Toc164963314)

[2.4. Source Systems 9](#_Toc164963315)

[3. Data Flows 9](#_Toc164963316)

[3.1. Populating FHIR Server 9](#_Toc164963317)

[3.2. Data Access 12](#_Toc164963318)

[4. Discussion 15](#_Toc164963319)

[4.1. Façade vs server discussion 15](#_Toc164963320)

[4.2. Alternatives for securing PACS system 15](#_Toc164963321)

[4.3. Access Control enforcement 15](#_Toc164963322)

[4.4. Alternatives for populating FHIR server with data 15](#_Toc164963323)

[4.5. Alternative SLS implementations 15](#_Toc164963324)

[4.6. Natural and Synthetic Unique Business Identifiers for resources 15](#_Toc164963325)

[4.7. Referential integrity and the logical order of data population 15](#_Toc164963326)

[4.8. MTLS termination and token binding alternatives discission 15](#_Toc164963327)

# Introduction

This document presents a reference architecture for an organizational FHIR infrastructure that adheres to and fully supports the requirements outlined in the *Technical Guidelines Document for FHIR Infrastructure Implementation*. This architecture leverages best practices and recommended approaches to establish a robust and secure foundation for organizational FHIR infrastructure.

*It is important to stress that this document serves as a reference point only and should not be* interpreted as a requirement or constraint on implementation.

Key considerations addressed within this architecture include:

* **Data Population:** Mechanisms for populating the FHIR server with data from source systems are outlined, including data model mapping and terminology considerations.
* **Data Labeling:** A practical approach to data labeling is presented, including handling of some edge cases.
* **Secure Data Access:** The architecture outlines methods for providing controlled and secure access to both FHIR data and large binary objects, addressing potential limitations of some widely used FHIR servers.

The architecture does not explicitly address scenarios that require write access by FHIR clients or backpropagation of changes from FHIR server to the source systems.

For the sake of brevity, this document intentionally omits representation and discussion of various caching mechanisms that can be employed to optimize performance of the organizational FHIR infrastructure.

*Note that this document intentionally omits any reference to specific platforms or vendors, as it is meant to be vendor neutral. Nevertheless, the architecture is designed to be practical and implementable using platforms and vendors readily available to an average healthcare organization.*

# Architecture Components

A diagram of a company

Description automatically generated

## Data Serving Components

### FHIR Client

This architecture supports any confidential SMART on FHIR client that supports MTLS. Beyond standard SMART, clients may provide additional context or claims during authentication and authorization process (e.g. – purpose of access) – as defined in the Technical Guidelines Document for FHIR Infrastructure Implementation.

### API Gateway

To secure FHIR REST API this architecture employs API Gateway that handles:

* MTLS termination with client certificate validation (against PKD)
* Access token binding validation
* FHIR schema validation via external FHIR Validator service

REST API calls that come from unknown clients, fail token binding validation, or fail schema validation are dropped and appropriate response is returned to the client.

*Note that the above strict requirements (e.g. – MTLS, token binding, etc.) are mandatory according to the Technical Guidelines Document for FHIR Infrastructure Implementation.   
It is, however, possible to enable more relaxed flows (e.g. – regular SMART on FHIR without MTLS, etc.) using the same infrastructure.*

### FHIR Validator

A diagram of a computer

Description automatically generated

FHIR Validator component is intended for organizations with security policies mandating schema validation of incoming FHIR API requests. This component leverages the HL7 Java validator, packaged within secure, stateless Docker containers optimized for horizontal scalability and reliability. It provides a user-friendly interface for simplified FHIR profile management and basic monitoring capabilities.

To maintain separation of responsibilities and minimize potential attack, the FHIR Validator operates outside the secure network and remains under the exclusive control of the InfoSec team. FHIR profiles used by this component serve to enforce security policies alone and are not intended to validate functional message integrity or overall data quality.

*Note: To ensure optimal performance, this component cannot rely on external terminology server, therefore all the value-sets in FHIR profiles must be either expanded or underlying code-system must either be provided (i.e. - content=complete) or excluded (i.e. - content=not-present).*

Note: While FHIR Validator component is provided here for reference it does not play an active role in this architecture, primarily focused on read-only scenario designed to satisfy requirements outlined in Technical Guidelines Document for FHIR Infrastructure Implementation.   
However, given that FHIR infrastructure is intended to serve organization in wide variety of use-cases, the FHIR Validator component will be much more prominent in write oriented scenarios.

### .well-known

This component provides FHIR Authorization Endpoint and Capabilities Discovery using a Well-Known Uniform Resource Identifiers according to the [spec](https://www.hl7.org/fhir/smart-app-launch/conformance.html). It is not served by the downstream components because it can be accessed without any authentication besides MTLS.

### Authorization Server

This component is a standard OAuth2 authorization server that supports the following functionality:

* OAuth 2.0 client credentials flow, with a JWT assertion
* OAuth 2.0 authorization code flow
* Proof Key for Code Exchange (PKCE)
* Token Introspection API
  + Besides mandatory active field introspection API will return SMART-related fields as [defined](https://www.hl7.org/fhir/smart-app-launch/token-introspection.html) in SMART on FHIR spec.
* SMART scopes and SMART context

Authorization Server is used to authenticate SMART on FHIR clients and, in case of SMART App Launch flow, to authenticate the users as well. (for simplicity this architecture intentionally omits a separate IdP, although in real-world deployments it is likely that a separate IdP will be present).   
Upon successful authentication this component will authorize the client/user based on a combination of their identity, any additional claims/context they have provided, and organizational access policies stored in the Permissions Management System, subsequently issuing an opaque access token with corresponding scopes.

The client registration process is beyond the scope of this document. However, it is important to note that to enable token binding enforcement by the PAI Gateway the *client\_id* should be set to thumbprint of the client’s certificate used for MTLS.

### Public Key Directory

This component is an X509 Public Key Directory (PKD) and Certificate Revocation List (CRL) that contains public keys for all registered clients. Provided reference architecture assumes presence of local PKD that is both - synchronized with national PKD and can also accommodate additional clients. National PKD is described in Technical Guidelines Document for FHIR Infrastructure Implementation. The synchronization with national PKD and registration process for local PKD as well as supporting infrastructure is beyond the scope of this document.

### Interceptor

Note, that this Reference Architecture is built under the assumption that the organizational FHIR server provides limited functionality and does not support any version of SMART on FHIR, but fully supports search via FHIR REST API as defined in the Technical Guidelines Document for FHIR Infrastructure Implementation. Therefore, the logic for the enforcement of access control (along with some additional functionality) is performed by the Interceptor component.  
In practice, this functionality could be provided by the FHIR server itself (with or without customization) or might indeed require a separate interceptor component – depending on the capabilities of the FHIR platform chosen by the organization.

This component handles all incoming and outgoing FHIR REST API traffic and has the following roles:

#### Access token validation

Examine the token using Token Introspection API of the Authorization server and determine its validity.

#### Abstraction of the underlying FHIR server (to ensure stable API surface)

Emulate FHIR API REST functionality that is required by the Technical Guidelines Document for FHIR Infrastructure Implementation but is not supported by the organizational FHIR server using a variety of techniques (e.g. – query rewriting, separation and aggregation, content modification, caching, etc.)

#### Access control enforcement against FHIR server

Support access control capabilities as outlined in the Technical Guidelines Document for FHIR Infrastructure Implementation that are not natively supported by the organizational FHIR server using variety of techniques (e.g. – query rewriting, separation and aggregation, content modification, etc.). Access Control will be based on the scopes and additional context from the access token extracted via Token Introspection API.

#### Content filtering

Provide some level of content filtering – especially for cases where some of the elements in the resources should not be returned using a variety of techniques (e.g. – query rewriting, separation and aggregation, pre/post-request content modification, etc.)

#### Audit and Logging

Support Audit and Logging capabilities as required by the Technical Guidelines Document for FHIR Infrastructure Implementation if not natively supported by the organizational FHIR server.

Interceptor component is basically a lightweight FHIR façade that augments and expands functionality provided natively by organizational FHIR server.

Interceptor component is designed to exclusively reverse-proxy request to organizational FHIR server and is not intended to provide FHIR façade for other organizational systems (apart from PACS system – see [[4.2] Alternatives for securing PACS system](#_Alternatives_for_securing)).

Interceptor does not require TLS as it can only receive traffic from API Gateway, where MTLS channel was already validated and terminated, along with the token binding enforcement.

Note, that Interceptor is a logical component. In terms of the actual implementation it can be either a separate service a customization of the FHIR server or some combination of the two.

### FHIR Server

The FHIR server component of this architecture is any FHIR compliant server or managed service. It is recommended to select a FHIR server that supports as many of the requirements outlined in the Technical Guidelines Document for FHIR Infrastructure Implementation as possible – to reduce the complexity of the Interceptor component and minimize the need for customization.

Note, that while it is technically possible to use a FHIR façade instead of a full FHIR server, this architecture opts for full FHIR server to reduce implementation complexity.

### Permissions Management System

Permissions Management System component enables system administrators to define access control rules using combination of operation type, accessor identity and context, resource attributes and/or list of specific resources stored in the FHIR server.

Access control rules are captured in the form of Consent resources and stored in the FHIR server.

Note, that the choice of Consent resources as data model and the choice of FHIR server as storage medium is made for convenience and portability. The same information could have been stored in Consent resource outside of FHIR server or in completely different modality (i.e. – key-value database).   
However, it is anticipated that this implementation choice will simplify the support for future requirements for patient consent management.

## Shared Components

### Audit and Logging

This component enables capturing of Audit trail and Logs from the Interceptor, FHIR server and Batch Orchestrator component - as required by the Technical Guidelines Document for FHIR Infrastructure Implementation

## Data Population Components

### Batch Orchestrator

Batch Orchestrator is used to run the ingest jobs that populate FHIR server with the data from source systems. The ingest jobs might run periodically – at scheduled intervals or based on events/triggers. Ingest jobs may be full or incremental.

This component will use FHIR Mapping Engine and Security Labeling Service in the context of ingest jobs to extract data from source systems, transform it into FHIR, label it and load it into FHIR server.

### Terminology Server

The Terminology Server is used by the FHIR Mapping Engine component for terminology translation in the process of mapping source system data into FHIR. In this architecture we are using a FHIR Terminology API compliant Terminology server.

In addition, this component is used by the FHIR server to support $validate functionality.

### FHIR Mapping Engine

FHIR Mapping Engine component is used to map data from the source systems into FHIR. It uses predefined templates to transform source data into FHIR resources and utilizes Terminology Server for codes mapping.

### Security Labeling Service

Security Labeling Service (SLS) labels FHIR resources with security labels (both for information buckets as well as for general information sensitivity) according to the Information Buckets Guidelines and Requirements. When labeling the resource SLS might consider multiple factors:

* Source system/subsystem/database/table
* Additional source context
* Resource type
* Resource fields (e.g. – category, code, etc.)
* Related resources and their fields (e.g. Patient referenced by the resource being labelled, specialty of the authoring Practitioner, or Security Labels set on the Encounter in context)

Note, that it is also possible to build a flow where SLS only operates on the data from the FHIR server. This architecture assumes that it’ll be easier to collect relevant context directly from the source systems during data load process and therefore uses SLS as part of said process. However, this might not be the case for all scenarios (see [[4.5] Alternative SLS implementations](#_Alternative_SLS_implementations) for further discussion).

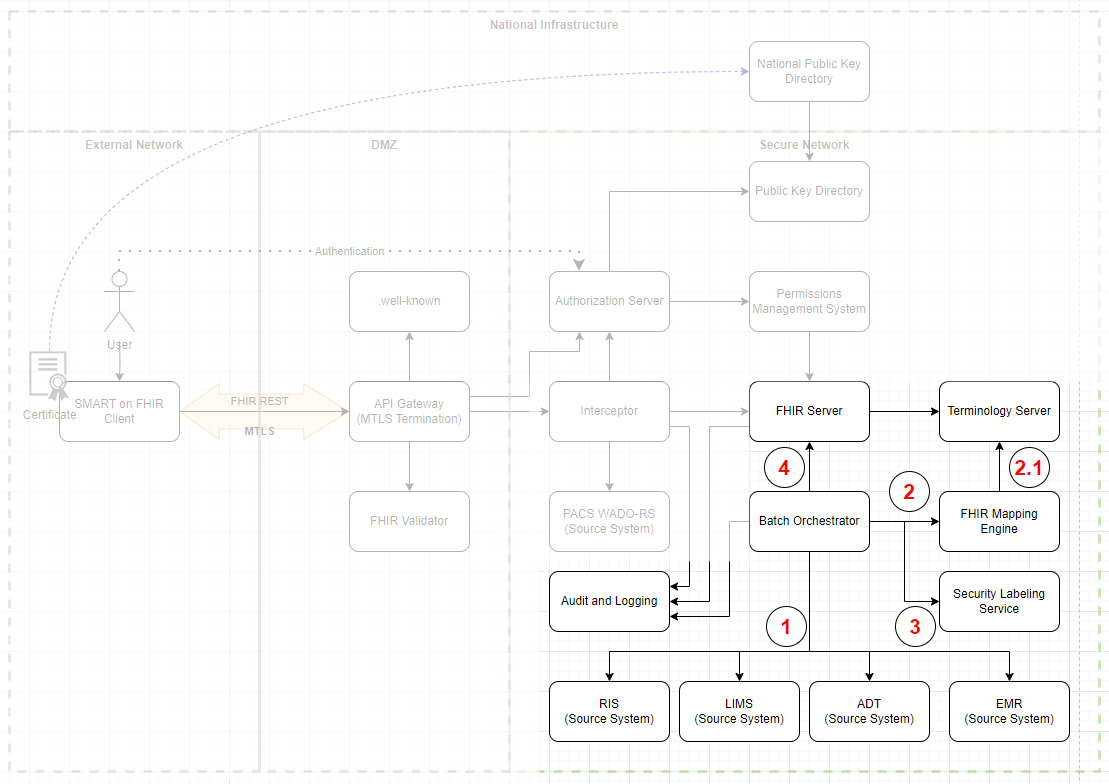
Note, that sometimes, existing resources will need to be re-labelled. Those occurrences can be roughly defined into two categories:  
- routine (i.e. – things that happen in the normal course of business operations – such as an Encounter in the source system was retrospectively marked as confidential or when a minor reaches an adulthood)  
- major (i.e. – significant changes that require special handling – such as a change in information buckets definitions)  
This architecture assumes that in the case of former (routine) - a change in the source system will trigger an update on all affected resources and they’ll flow through the Batch Orchestrator pipeline resulting in the updated resources. In the case of latter (major) the change will likely require a dedicated data fix or, in extreme cases, full reflow of data and it’s handling is beyond the scope of this document.

### Source Systems

All organizational systems that hold relevant data, such as EMR, LIMS, ADT, RIS, etc. One notable exception is PACS as DICOM instances are too large to be stored directly in the FHIR server. Instead, relevant metadata is collected and stored in FHIR server with links pointing directly at PACS using WADO-RS. Requests to WADO-RS go through Interceptor component as well – to provide access control enforcement.

# Data Flows

## Populating FHIR Server



#### **Step 1** – Extraction of incremental changeset from the source systems

In the first step Batch Orchestrator periodically executes process to extract data from the source systems.

This document assumes that FHIR Server will be populated via incremental batch process running at scheduled intervals. To satisfy data freshness requirements of the Technical Guidelines Document for FHIR Infrastructure Implementation the interval is set to 5 minutes. Assuming there is at least one change in the source systems every 5 minutes and assuming all the changes from the 5 minutes window can be processed and loaded into FHIR server in less than 5 minutes – this is an efficient strategy. There are alternative strategies that can accommodate different data change patterns – see Discussion section for more information.

For the purposes of this architecture, we assume that source systems contain the original data and FHIR server is technically a read-only replica, an ODS. Therefore, Batch Orchestrator will always extract complete (as in – all information available to date that is required to generate complete FHIR resource(s) or transaction(s)) data entities that were changed from the source system.

We also assume that the source systems allow incremental extraction of changes.

Internally Batch Orchestrator may use FIFO queues to control flow of data/tasks through the pipeline.

#### **Step 2** – Conversion into FHIR resource

Extracted data is sufficient to generate complete FHIR resource by the FHIR Mapping Engine, so no additional external calls (besides terminology) are required. Mapping engine can use variety of techniques and platforms (templates hydration, source to target mapping, FHIR SDK DOM hydration, etc.) to generate a complete and valid FHIR resource.

A unique business identifier should be included in each resource (see [[4.6] Natural and Synthetic Unique Business Identifiers for resources](#_Natural_and_Synthetic) for additional discussion).

Data that is not directly provided by the source systems (e.g. – last updated timestamp, unique business identifiers that do not have natural equivalents in the source system, etc.) will be generated by the mapping engine using contextual data at this stage.

A special case that requires additional attention is WADO-RS links in the ImagingStudy resource. For the purposes of this architecture, they are re-encoded in a way that will preserve the original link (adjusting for base path) and will also include business identifier of the ImagingStudy resource itself. This data will be used during access to enforce permissions. The same approach can be used for other links (e.g. – binary data that is not embedded inside DocumentReference.content.attachment, etc.)

In cases where mapping is required from one code system to another (to satisfy FHIR profiles requirements) – the mapping engine will use Terminology Server to do the mapping. It is assumed that translation maps were prepared and loaded into Terminology Server in advance.

In case of error, Batch Orchestrator will log it and may retry the conversion attempt. If deemed unrecoverable the problematic task can be placed in a dead letter queue for manual review.

A complete set of FHIR resources is returned to the orchestrator that may choose to validate it against a set of profiles to ensure compliance.

#### **Step 3** – Labeling

Batch Orchestrator passes the complete FHIR resource along with supporting information (e.g. – source system, source table, source system labels, additional related data, etc.) to the Security Labeling Service (SLS).

SLS will examine the resource and the provided context and apply a set of security labels representing information buckets and sensitivity levels to the resource. SLS may add additional security labels and tags if required by organizational process. The exact logic for choosing the labels is left for the specific implementation, but it is assumed that based on the combination of the data itself (the resource) and the contextual metadata (e.g. – source system, source table, source system labels, etc.) organizations can define deterministic rules for label assignment. It is possible that in some cases human intervention might be required to determine the correct labels. In such cases pipeline may include separate queue for resources that require manual review process. However, this provision is not explicitly addressed in this reference architecture. In any case, it is assumed that manual interventions will be few and far between and will mostly serve as a tool to improve automated labeling.

Once the resource is labeled it will be returned to the orchestrator along with supporting data that explains why the specific labels were chosen. The orchestrator will log the labeling details.

There are additional approaches to labeling – see Discussion section for more information.

#### **Step 4** – Reference resolution and population of the FHIR server

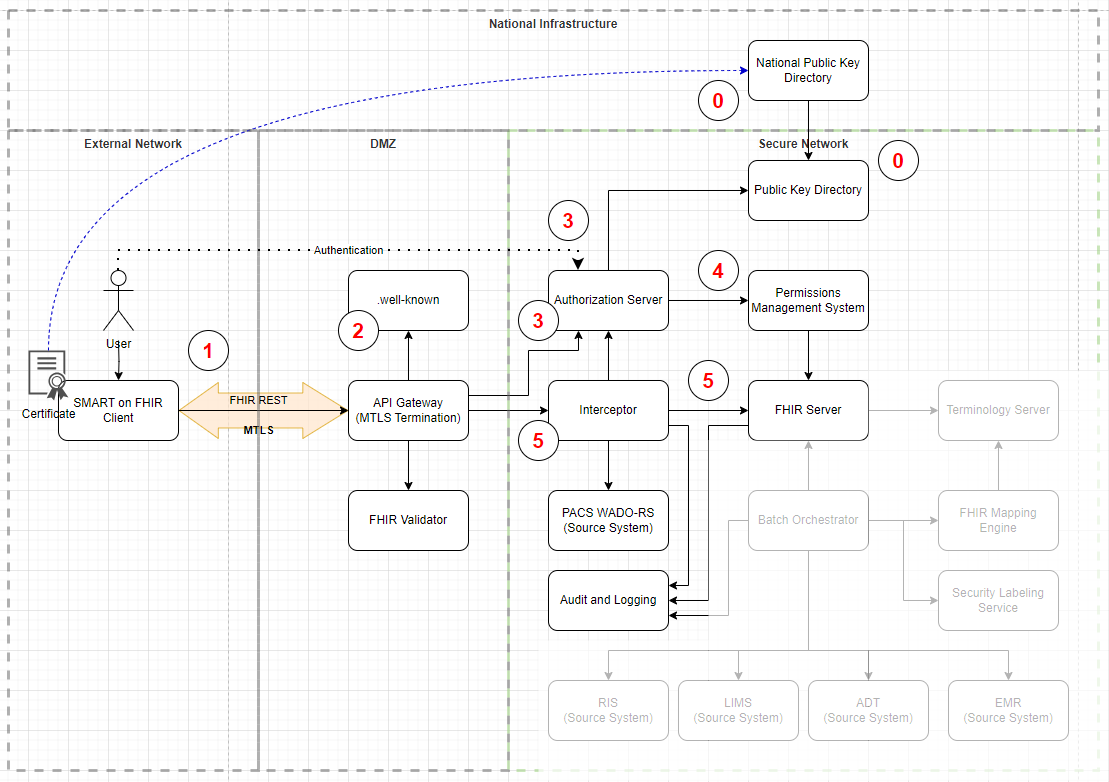
As it is assumed that FHIR server does not hold any original/unique data, it is safe to overwrite the resources in the FHIR server. However, logical ids must be preserved to maintain referential integrity of literal references. Therefore, new, and updated resources will be submitted to FHIR server via conditional update using a natural unique business identifier from the source system or one generated by the mapping engine. All references to other resources will be conditional – using the same unique business identifier.

Note, that there are alternative approaches to reference resolution and maintaining referential integrity. One example is using Mapping Engine. See [[4.6] Natural and Synthetic Unique Business Identifiers for resources](#_Natural_and_Synthetic) and [[4.7] Referential integrity and the logical order of data population](#_Referential_integrity_and) for additional discussion.

Updates will be sent to FHIR server in a transaction Bundle. While it is technically possible to combine multiple unrelated updates into a single transaction, it is ill advised, as in case of failed transaction (e.g. – due to problematic conditional reference) it’ll be difficult to separate problematic updates from valid ones. Failed transactions are logged and escalated for human intervention.

Note that in some cases (e.g. – during initial data load, when adding new types of entities, etc.) it is possible that a resource being loaded into the FHIR server must be referenced by another resource that is already present in the FHIR server and wasn’t included in the change being processed by the orchestrator. Batch Orchestrator must detect this condition and update the relevant resources in the FHIR server to preserve referential integrity – see discussion section for more details.

## Data Access



#### **Step 0** – Out of band registration of client certificate

The system assumes that each client and each server have its certificate registered with the National Public Key Directory (NPKD)– as outlined in the Technical Guidelines Document for FHIR Infrastructure Implementation. Description of the registration flow is beyond the scope of this document.

Local Public Key Directory (PKD) is synchronized with NPKD. Description of this synchronization process is also beyond the scope of this document.

#### **Step 1** – MTLS connection

Client establishes MTLS connection to the organizational API gateway using the certificate registered with the PKD. API Gateway verifies client certificate against PKD. Client must also verify server certificate against NPKD. MTLS is terminated at the API gateway.

#### **Step 2** – Client requests .well-known/smart-configuration document

Client requests discovery document according to standard SMART on FHIR flow. The document is a static resource and is hosted separately from the main FHIR server, not passing through the Interceptor.

#### **Step 3** – Client/user authentication and authorization

Client goes through standard SMART App Launch or SMART Backend Services flow against the authorization server discovered via .well-known/smart-configuration document. In both cases client must be authenticated authenticate using the same certificate registered with PKD via client-confidential-asymmetric authentication (as Technical Guidelines Document for FHIR Infrastructure Implementation do not permit public clients or confidential clients using symmetric authentication).

In the case of SMART Backend Services client goes through OAuth 2.0 client credentials flow with a JWT assertion – as defined in SMART on FHIR spec. In the case of SMART App Launch the client goes through OAuth 2.0 authorization code flow – as defined in SMART on FHIR spec. Specifics of user authentication are beyond the scope of this document.

Note that certificate thumbprints of all JWT tokens used for client authentication that are passed through API Gateway are compared to the thumbprint of the certificate used for MTLS. If they are not equal the requests are dropped.

#### **Step 4** – Authorization

Authorization server will pass client/user identity, along with any additional claims and context provided in the authentication and authorization process (either programmatically from the client or interactively – at user authentication stage) to the Permissions Management System, which will in turn provide access scopes and additional context back to the Authorization server. Assuming happy flow, the Authorization server will retain the scopes and additional context in its repository and return opaque access and refresh tokens back to the client. The access token is short-lived – according to the requirements in Technical Guidelines Document for FHIR Infrastructure Implementation.

The Authorization server itself does not require any FHIR-specific functionality and many existing commercial and OSS solutions as well as managed services can be used out of the box.

Note that authentication/authorization process described above is intentionally redacted and simplified for clarity. See [SMART on FHIR documentation](https://www.hl7.org/fhir/smart-app-launch/index.html) and Technical Guidelines Document for FHIR Infrastructure Implementation for detailed specifications and requirements.

Note that Authorization server is accessed directly, not via the Interceptor.

#### **Step 5** – Data Access

Client accesses FHIR server using the access token it received from the Authorization Server.

Note that Client should only request the data it needs – according to the minimization principles outlined in the Technical Guidelines Document for FHIR Infrastructure Implementation.

Requests from the client are passed via API Gateway (which performs MTLS termination) to the Interceptor component. Interceptor uses Introspection API of the Authorization server to decode the access token and check its validity.

Note that token binding is not checked during regular FHIR API access (as it would require API Gateway to use Introspection API to decode access tokens – which may incur tangible performance penalty). Token binding is only enforced during client authentication. However, since access tokens are short lived, and all clients are known in advance and trusted enough to participate in information exchange it is assumed that the risk for token replay attacks is low). If required, token binding can still be enforced - API Gateway could store MTLS certificate thumbprint in custom HTTP header and Interceptor can compare it to the client\_id from the access token as returned by the Introspection API.

For the purposes of this document, it is assumed that the organizational FHIR server does not support any version of SMART on FHIR, but fully supports search via FHIR REST API as defined in the Technical Guidelines Document for FHIR Infrastructure Implementation. Therefore, after checking token validity and extracting scopes and context from it using Introspection API, the Interceptor component will generate a new query to the FHIR server using original client query and access scopes - in a way that will only return data allowed by the access scopes from the token. If organizational FHIR server supports different subset of SMART on FHIR and FHIR REST API functionality – the access control approach and Interceptor component functionality might be different – see Access Control enforcement in the Discussion section for additional details.

After a query is executed by the FHIR server the results are returned to the Interceptor and then passed on to the Client.

If additional restrictions must be applied to the data (e.g. – additional filtering of outgoing data at the resource or sub-resource level, etc.) – it can be done in the Interceptor, before the response is returned to the client. Hower, this document assumes that no additional restrictions should be applied.

All requests (whether granted or denied) are logged by the Interceptor via the Audit & Logging component.

For DICOM access, WADO-RS/WADO-URI/IID requests are routed via Interceptor component as well, along with the same access token. Interceptor will validate and introspect the access token in the same way as for regular FHIR REST API access and determine if client has access to the requested content. To make the determination it’ll extract the business identifier of the ImagingStudy resource from the URL (as it was encoded at the data population stage) and will evaluate if supplied access token grants access to the said resource. It may query FHIR server to retrieve the ImagingStudy resource to make that determination. If provided access token grants access to the ImagingStudy resource the access to PACS is also allowed and the request is passed through. Note that this approach means that all DICOM data is passed through the Interceptor component – which might introduce a significant overhead. Depending on the specific scenario other approaches can be devised.

# Discussion

## Façade vs server discussion

## Alternatives for securing PACS system

## Access Control enforcement

## Alternatives for populating FHIR server with data

## Alternative SLS implementations

## Natural and Synthetic Unique Business Identifiers for resources

## Referential integrity and the logical order of data population

## MTLS termination and token binding alternatives discission