Gen3 FHIR Server

Gen3 Community Feature Document

Author(s)

Author Name	Email Address	Organization or Team
Andriana Sielli	sielliandriana@gmail.com	GSoC contributor- Summer 2025

Reviewer(s)

Reviewer Name	Email Address	Review Date
Alex Vantol	avantol@uchicago.edu	
Kyle Burton	burtonk@uchicago.edu	

Major Version History

Version Name or Number	Date	Comments/Major changes
Version 1.0	Oct 2025	

Requirements

Purpose

The purpose of this design is to enable secure, standards-based access to FHIR (Fast Healthcare Interoperability Resources) data within the Gen3 ecosystem, bridging clinical and research data management through a modular, interoperable, and FAIR-aligned architecture.

This feature integrates a HAPI FHIR JPA server with Gen3's Arborist authorization service via a FastAPI proxy, ensuring that FHIR resources can be accessed and managed in compliance with Gen3's fine-grained access control policies. It addresses the current gap between Gen3's data governance layer and the FHIR standard, allowing biomedical and clinical data to be queried, stored, and retrieved securely and reproducibly.

In Scope

- Implementing a FastAPI-based proxy for secure, asynchronous communication between clients and the HAPI FHIR server.
- Integrating with Arborist to enforce Gen3-style resource-level authorization using bearer tokens and security tags.
- Supporting core FHIR operations (CRUD) on standard NCPI resource profiles.

- Containerizing the architecture using Docker and Docker Compose for reproducible local and cloud deployments.
- Preparing full documentation, testing, and deployment workflows.

Use Cases

#	Context	Priority (A/B/C)	Use Case(s)
1	Researcher	A	As a researcher, I want to query FHIR resources (Patient, Observation, Condition, etc.) so that I can access the clinical data I need for my study in a secure and controlled manner.
2	Clinician	A	As a clinician, I want to search for patient-related studies and clinical observations using FHIR queries so that I can quickly find relevant information to support patient care and clinical decision-making.

User Flow

1. User Authentication and Access Control

1. The user logs in to the Gen3 ecosystem using their Gen3 credentials.
2. The client application receives a JWT bearer token for the session.
3. When the user sends a request to the FHIR Proxy, the proxy validates the token against Arborist to confirm the user's permissions.
4. The proxy filters and annotates FHIR requests according to resource-level access policies, ensuring that users see only the data they are authorized to access.

2. Querying and Retrieving FHIR Data

5. The user sends a FHIR query to the proxy (e.g., retrieve all Observations for a Patient).
6. The proxy forwards the authorized request asynchronously to the HAPI FHIR Server.
7. The HAPI FHIR server executes the query against its PostgreSQL database.
8. Results are returned to the proxy, which applies any remaining security tags or transformations before sending the response back to the user.

Functional Requirements

- **Support the NCPI FHIR Data Model**
  The system must support core NCPI FHIR resource profiles (e.g., Patient, Observation, ResearchStudy) and validate all ingested or queried data against these profiles.

- **FastAPI Proxy for Secure Access**
  All API requests must be routed through a FastAPI-based proxy that validates JWT tokens via Arborist, filters resources based on user permissions, and adds Gen3-compliant security tags.

- **Asynchronous Data Handling**
  Support asynchronous ingestion and forwarding of FHIR resources to the HAPI FHIR server using httpx.AsyncClient to enable scalable, high-throughput workflows.

- **Semi-Automated Data Ingestion**
  Allow batch ingestion of datasets into the FHIR server with validation and error reporting.

- **Custom Authorization Middleware**
  Implement middleware or a plugin that enforces fine-grained access control, filtering FHIR resources based on user roles and permissions defined in Arborist.

- **Secure Querying with Enforced Filtering**
  Enable users to query FHIR data while ensuring access restrictions are applied to all returned resources.

- **Containerized Deployment**
  Deploy the proxy, HAPI FHIR server, and Arborist in Docker containers for reproducibility and ease of setup.

- **Testing and Validation**
  Provide unit and integration tests (using pytest and pytest-httpx) covering token validation, proxy routing, resource forwarding, and permission enforcement.

- **Documentation and Onboarding Support**
  Include comprehensive setup guides, usage instructions, and example workflows to support reproducibility and new user onboarding.

- **Future-Ready Extensions**
  Enable Elasticsearch-backed full-text search for advanced queries and support Helm-based Kubernetes deployment for scalable cloud-native operations.

Technical Requirements

- **Code Quality and Maintainability**
  Code must be clean, well-documented, and consistent with Gen3 coding standards to ensure long-term maintainability.
- **Error Handling and Reliability**
  The system must gracefully handle edge cases, provide clear and human-readable error messages, and ensure resilience under failure conditions.
- **Performance and Scalability**
  The system must scale efficiently under load, performing comparably to existing Gen3 services when deployed in production environments.

- **Deployment and Configuration**
  All components must be containerized, configurable via environment variables, and deployable in both local and cloud-native environments using Docker and Helm.
- **Documentation and Transparency**
  Up-to-date, public-facing documentation must be provided, including setup guides, configuration examples, and usage workflows for developers and users.

Design and Testing

Additional Background

The Gen3 FHIR Server builds upon existing open-source components — primarily HAPI FHIR, FastAPI, and Gen3's Arborist authorization service. Early prototypes demonstrated feasibility for integrating FHIR-based data services within Gen3, showing that secure, standards-compliant data exchange could be achieved through a proxy-based architecture. This current design refines those efforts by introducing asynchronous request handling, improved modularity, and enhanced support for NCPI-compliant data models.

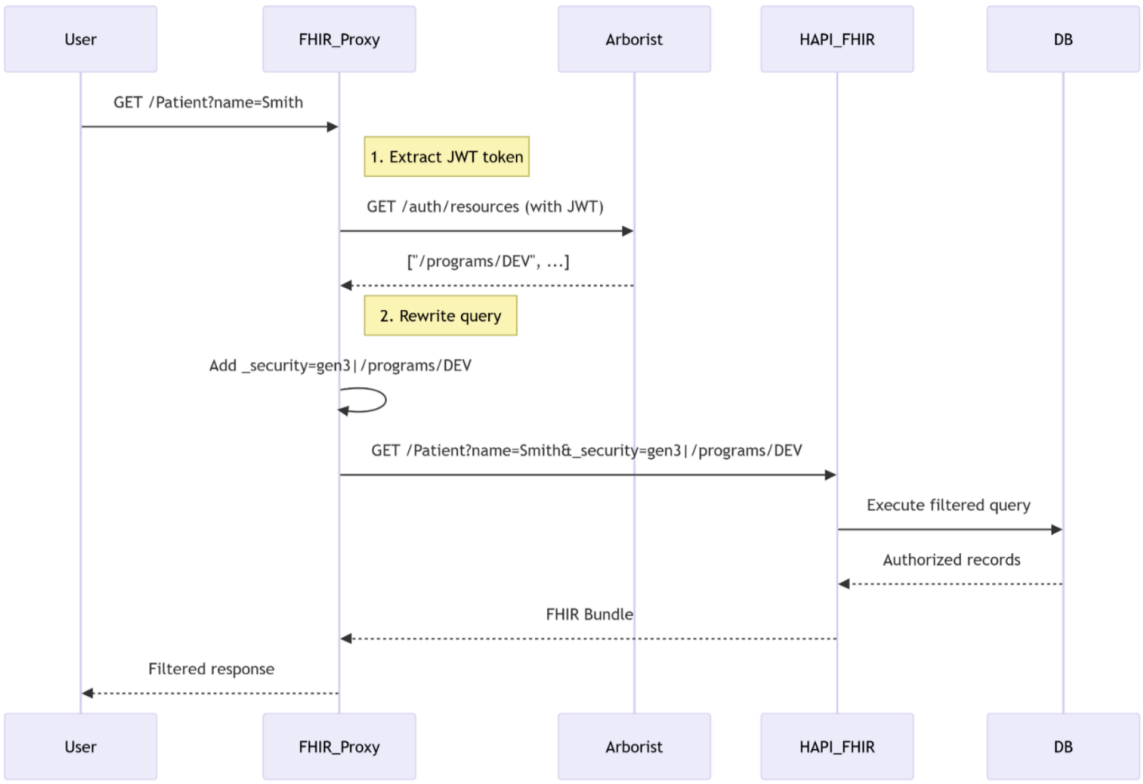
The decision to use FastAPI (for performance and async support) and HAPI FHIR (for standards-compliant data storage) was based on extensive community adoption, documentation quality, and ease of integration with existing Gen3 infrastructure.

Design Summary

The proposed design integrates a FastAPI-based FHIR Proxy that securely intermediates communication between clients and a HAPI FHIR JPA Server. The proxy validates access tokens via Arborist, filters requests based on user permissions, and appends Gen3-compliant security tags to ensure proper data governance.

Data ingestion and querying are handled asynchronously to support high throughput. All services are containerized and orchestrated via Docker Compose for local testing, with future support for Helm-based Kubernetes deployment to enable scalable, cloud-native operation. Integration testing ensures reliable interaction between proxy, HAPI FHIR, and Arborist services.

Architectural Diagram



AuthN/AuthZ

Authentication (**AuthN**) and authorization (**AuthZ**) are handled through the **Gen3 Arborist** service, ensuring that all data interactions comply with the Gen3 security model.

Authentication

- Clients authenticate using JWT bearer tokens obtained from Gen3.
- The proxy verifies these tokens using Arborist's `/auth/resources` endpoint.
- Tokens include user identity and roles, enabling resource-level authorization decisions.

Authorization

- Once authenticated, requests are filtered based on Arborist policies, which define which users or groups can access specific FHIR resources or data domains.
- The proxy dynamically applies security tags (e.g., `gen3-security:resource_id`) to enforce governance on both ingestion and retrieval operations.
- Unauthorized requests are rejected with standard HTTP responses (`401 Unauthorized` or `403 Forbidden`).

This model ensures fine-grained control over data access while maintaining interoperability with FHIR-compliant systems.

Alternatives Considered

FHIR server choices:

In evaluating options for the FHIR server, the following alternatives were explored and excluded for specific reasons:

- **IBM FHIR Server** – Excluded due to its commercial licensing model, which limits open-source or flexible deployment.

- **Microsoft FHIR Server** – Excluded because of its tight dependency on Azure, making it unsuitable for cloud-agnostic or on-premise deployments.

- **Firely Server** – Excluded since key advanced features require a paid license.

- **Medplum** – Excluded as it is relatively new and lacks a long track record in production use.

- **Blaze** – Excluded because it is Haskell-based, resulting in a niche option with a steeper adoption curve.

- **Python SMART** – Excluded as it only provides a FHIR client, not a server implementation.

- **LinuxForHealth** – Excluded because it functions mainly as an ETL and integration tool, not a full FHIR server.

After this assessment, focus was placed on two primary options:

HAPI FHIR and **FHIR Starter**.

Feature	HAPI FHIR	FHIR Starter
Language	Java	Python
FHIR Compliance	Fully supports FHIR specification (high compliance); all standard operations included.	Simpler core FHIR only; lacks advanced features and requires manual profile validation.
Performance	~1,000+ requests/sec	~500 requests/sec
GraphQL Support	Built-in FHIR GraphQL	Requires Apollo Server integration
Elasticsearch Integration	Native module	Custom ETL needed
Authorization	Java interceptors (complex)	Python middleware (simpler)
Deployment	Heavyweight, slow startup; Helm charts available; stronger for production.	Lightweight, fast startup; custom Helm charts needed; easier setup.
Data Ingestion	Requires Java pipelines	Direct HTTP requests, Python scripts
Community & Maintenance	Mature, 10+ years	Newer, smaller community (opportunity to grow Python FHIR community)
Gen3 Alignment	Better features, worse stack fit (Java-based)	Worse features, better stack fit (Python-native)

Decision: We selected HAPI FHIR as the foundation for the Gen3 FHIR server due to its mature codebase, full compliance with FHIR standards, robust performance, native support for GraphQL and Elasticsearch, and production-ready deployment options.

Database choices

Database	Status	Hibernate Dialect Class	Notes
MS SQL Server	**Supported**	`ca.uhn.fhir.jpa.model.dialect.HapiFhirH2Dialect`	
PostgreSQL	**Supported**	`ca.uhn.fhir.jpa.model.dialect.HapiFhirPostgresDialect`	
Oracle	**Supported**	`ca.uhn.fhir.jpa.model.dialect.HapiFhirOracleDialect`	
Cockroach DB	Experimental	`ca.uhn.fhir.jpa.model.dialect.HapiFhirCockroachDialect`	A CockroachDB dialect was contributed by a HAPI FHIR community member. This dialect is not regularly tested, use with caution.
MySQL	Deprecated	`ca.uhn.fhir.jpa.model.dialect.HapiFhirMySQLDialect`	MySQL and MariaDB exhibit poor performance with HAPI FHIR and have therefore been deprecated. These databases should not be used.
MariaDB	Deprecated	`ca.uhn.fhir.jpa.model.dialect.HapiFhirMariaDBDialect`	MySQL and MariaDB exhibit poor performance with HAPI FHIR and have therefore been deprecated. These databases should not be used.

Decision: We chose PostgreSQL to be compatible with the current infrastructure of Gen3