Grant Agreement No.: 101080718
Call: HORIZON-HLTH-2022-STAYHLTH-01-two-stage
Topic: HORIZON-HLTH-2022-STAYHLTH-01-05-two-stage
Type of action: HORIZON-RIA

BIO-STREAMS

D4.1 Biobank IMS

Biobank Information Management System

Revision: v.1.1

Work package	WP4
Task	Task 4.1
Due date	31/10/2025
Submission date	31/10/2025
Deliverable lead	Netcompany - Intrasoft
Version	1.1
Authors	Marios Logothetis, Anastasios Gogos, Petros-Sozon Dimitrakopoulos
Reviewers	Eleni Georga (UOI), Konstantinos Lentzos (UNI)

Abstract	The BIO-STREAMS Information Management System (IMS) is a FastAPI-based microservice that enables secure, standardized access to distributed clinical and synthetic data across multiple European hospital nodes. Integrated with seven active hospital node bundles via a dedicated gateway, the IMS provides unified data aggregation and querying capabilities while ensuring robust logging and environment-specific configurability. Its successful containerized deployment and validated end-to-end data flow establish a foundational technical component for the BIO-STREAMS biobank infrastructure.

Keywords	IMS, FastAPI, OpenAPI, Swagger, query-processing, node-bundles, node-gateway, CI/CD, testing, Docker containers

DOCUMENT REVISION HISTORY

Version	Date	Description of change	List of contributor(s)
1.1	31/10/2025	Final version	Anastasios Gogos, Marios Logothetis, Petros-Sozon Dimitrakopoulos
1.0	31/10/2025	Internal Review	Eleni Georga, Konstantinos Lentzos
0.9	30/10/2025	References, minor corrections	Anastasios Gogos, Marios Logothetis, Petros-Sozon Dimitrakopoulos
0.8	29/10/2025	Curation, references, abbreviations	Anastasios Gogos, Marios Logothetis, Petros-Sozon Dimitrakopoulos
0.7	27/10/2025	Curation, review, minor fixes	Anastasios Gogos, Marios Logothetis, Petros-Sozon Dimitrakopoulos
0.6	24/10/2025	Add more sections and figures to Ch. 1	Anastasios Gogos, Marios Logothetis, Petros-Sozon Dimitrakopoulos
0.5	23/10/2025	Abstract, minor corrections, table updates	Anastasios Gogos
0.4	22/10/2025	Review, curation, figures	Anastasios Gogos, Marios Logothetis
0.3	21/10/2025	Curation, enhancements, figures	Anastasios Gogos
0.2	20/10/2025	Basic content added (structure and text)	Anastasios Gogos
0.1	01/10/2025	1st version of the document	Anastasios Gogos

Disclaimer

Co-funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the other granting authorities. Neither the European Union nor the granting authority can be held responsible for them.

Copyright notice

Project co-funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	DEM	
Dissemination Level		
PU	*Public, fully open, e.g. web*	**X**
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* *R: Document, report (excluding the periodic and final reports)*
DEM: Demonstrator, pilot, prototype, plan designs
DEC: Websites, patents filing, press & media actions, videos, etc.
DATA: Data sets, microdata, etc
DMP: Data management plan
ETHICS: Deliverables related to ethics issues.
SECURITY: Deliverables related to security issues
OTHER: Software, technical diagram, algorithms, models, etc.

Executive summary

As part of Task 4.1, the BIO-STREAMS Information Management System (IMS) represents a critical runtime component of BIO-STREAMS which serves as a central data aggregation and query-processing service that enables access to clinical and synthetic data across participating hospital *Node Bundles*.

The IMS has been successfully implemented as a FastAPI-based [1] microservice that provides standardized query interfaces to hospital node bundles deployed across multiple European healthcare institutions. Through the *Node Gateway* component, the IMS currently integrates with 7 active hospital nodes, enabling dashboard users to query and aggregate clinical data through a unified API, while maintaining proper logging.

Key achievements include the successful containerization and deployment of the IMS to the project's development environment through automated CI/CD pipelines, establishing network connectivity between the IMS and hospital node bundles via the node-gateway infrastructure. Real-world integration testing has validated the complete data flow from IMS through node-gateway to individual hospital node bundles, demonstrating the system's capability to process clinical queries across distributed healthcare environments. The implementation employs environment-specific configuration management allowing seamless deployment across different environments (for example: development / production).

This first release establishes a core infrastructure / network component for the BIO-STREAMS biobank, contributing to the technical foundation of the querying mechanism used within the project.

Table of contents

List of figures

List of tables

Abbreviations

API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
IMS	Information Management System
IP	Internet Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

1 System Architecture

The IMS operates as a central data aggregation and query-processing component within the broader BIO-STREAMS platform architecture. As illustrated in Figure 1, the IMS serves as a critical component that facilitates data querying and data flow between hospital Node Bundles and the main user-facing application of the project, the Dashboard.

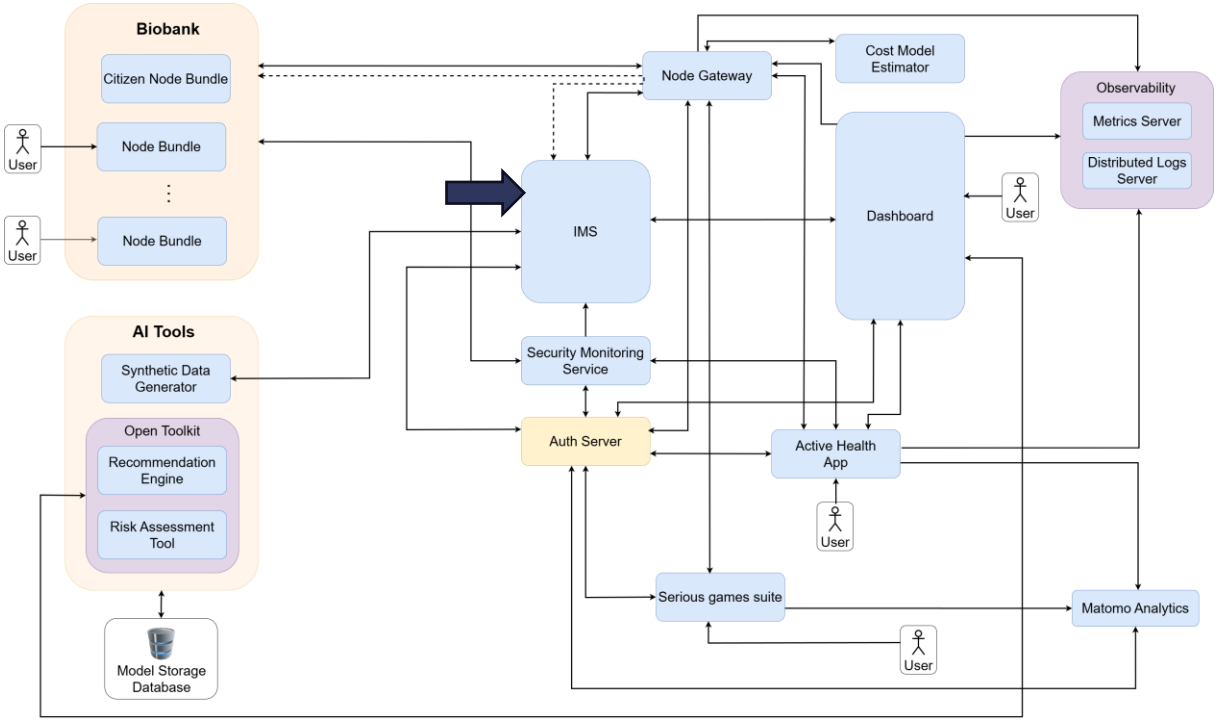


Figure 1 – BIOSTREAMS Architecture from D5.1

1.1 Integration Points

The IMS interfaces with multiple system components through well-defined APIs:

- **Node Gateway Integration**: The IMS receives clinical data from hospital node bundles through the node-gateway component, which aggregates data from the distributed hospital environments.
- **User Interface Connectivity**: The IMS provides data to the Dashboard, enabling project users to query and download clinical information through standardized interfaces.
- **Synthetic Data**: The IMS also requests data created by the Synthetic Data Generator component, supporting the platform's AI-driven capabilities.
- **Security Monitoring Service**: The IMS integrates with the Security Monitoring Service for comprehensive security oversight.

1.2 Data Flow Architecture

The system implements a distributed data processing model where:

1. Hospital *Node Bundles* collect and store clinical data locally

2. The *Node Gateway* establishes and maintains secure SSH connections to the Node Bundles and provides the only way to request data from them

3. The IMS processes queries coming from the *Dashboard*, routes relative queries to the hospital *Node Bundles* via the Node Gateway defined in the initial query, gathers the individual results and responds with a unified collection of data.

4. *Dashboard* consumes the aggregated data from IMS.

1.3 Source Code

The source code of the IMS component can be found under the GitHub [6] Organization of the BIO-STREAMS project in the following link: https://github.com/bio-streams-eu-project/ims (currently as a "private" repository) together with the relative guidelines for setting it up in a local environment for development purposes.

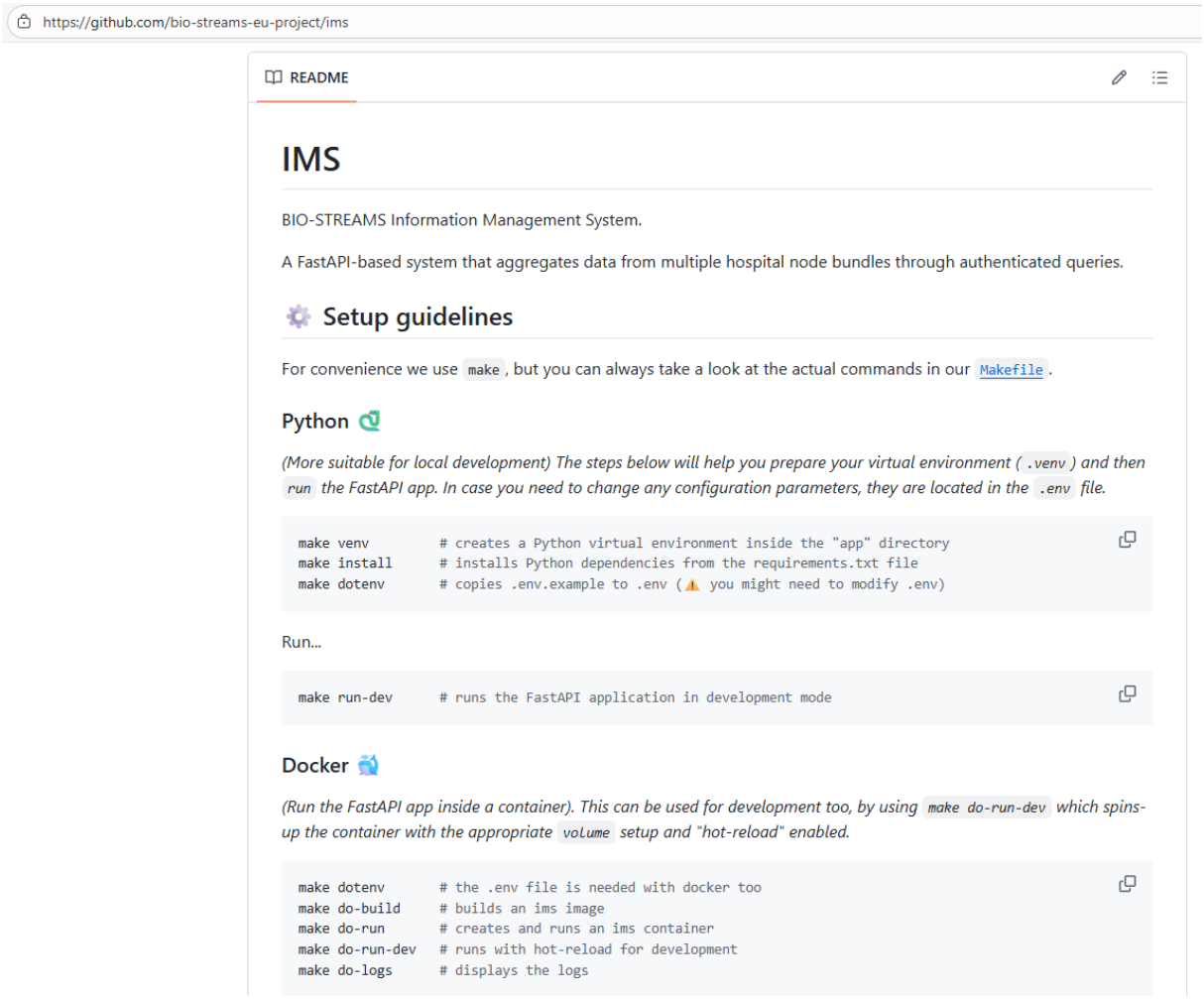


Figure 2 – README file of the IMS repository

1.4 Deployment Architecture

The IMS is deployed as a containerized microservice within the BIO-STREAMS platform, currently operating and being tested in the development environment with planned expansion to production deployment. In Figure 3, part of our Portainer [7] dashboard is being shown, depicting the development environment and some of the deployed services.

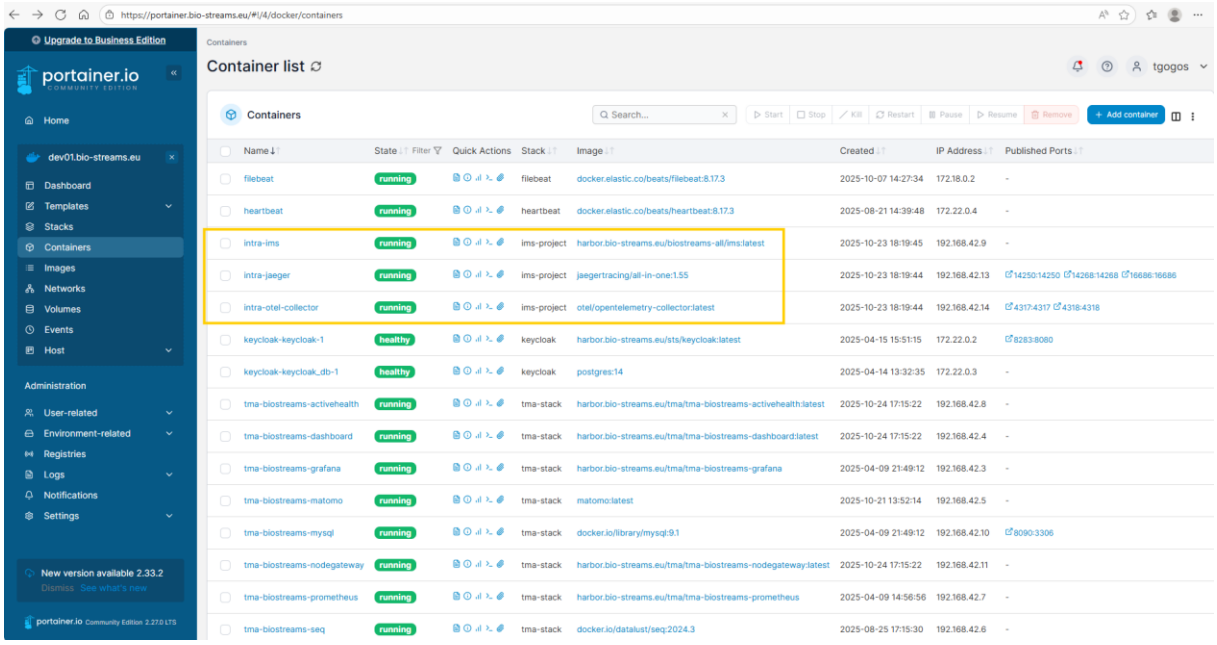


Figure 3 – running IMS containers inside the Development environment

1.5 Node Bundles configuration

The listed Node Bundles below are pre-configured to a file that is being loaded every time the IMS service is bootstrapping:

Node Bundle ID	Hospital Name	Country	Base URL
UKCM	University Clinical Centre Maribor	Slovenia	http://nodegateway.dev.bio-streams.eu:10001/
NKUA	National and Kapodistrian University of Athens	Greece	http://nodegateway.dev.bio-streams.eu:10003/
KI	Karolinska Institute	Sweden	http://nodegateway.dev.bio-streams.eu:10005/

BLOCKS	Blocks Health and Social Care EOOD	Bulgaria	http://nodegateway.dev.bio-streams.eu:10007/
VHIR	Val d'Hebron Research Institute	Spain	http://nodegateway.dev.bio-streams.eu:10009/
CHUL	University Hospital of Liege	Belgium	http://nodegateway.dev.bio-streams.eu:10011/
PENTELI	Penteli General Children's Hospital	Greece	http://nodegateway.dev.bio-streams.eu:10013/
CITIZEN	Citizen Node Bundle	-	http://nodegateway.dev.bio-streams.eu:10015/
test	Node Bundle used for development / testing	-	http://193.2.20.29:8050

Table 1 – List of Node Bundles configured within the IMS service

2 Technical implementation

2.1 FastAPI Application

The IMS is built on FastAPI [1], providing a modern, high-performance web framework for Python [4]. The application implements a centralized query processing mechanism that aggregates data from multiple hospital Node Bundles through authenticated API calls.

2.1.1 Core Functionality and Query Mechanism

The system implements a distributed query processing model where:

- **Query Distribution:** The IMS receives query requests and distributes them to active hospital node bundles based on a configuration parameter included in the query
- **Data Aggregation:** Responses from individual hospitals are collected, merged, and returned as a unified dataset
- **Asynchronous Processing:** Queries are processed concurrently using Python's asyncio framework, enabling efficient handling of multiple hospital requests simultaneously
- **Error Handling:** The system implements comprehensive error handling for network failures, authentication issues, and data validation errors

2.1.2 Request/Response Handling and Logging

The application implements structured logging throughout the request lifecycle:

- **Request Logging:** Each incoming query is logged with a unique request ID, requester information, and query parameters
- **Hospital Response Tracking:** Individual hospital responses are logged with success/failure status and response times
- **Performance Metrics:** Query execution times and data volume metrics are captured for monitoring and optimization
- **Error Logging:** Detailed error information is logged for troubleshooting and system maintenance

2.2 API Endpoints

2.2.1 Basic Endpoints

The system provides fundamental endpoints designed for microservice integration and health monitoring. The root endpoint (`GET /`) returns a simple "Hello" message, serving as a basic connectivity test for other microservices that need to verify IMS availability. The health endpoint (`GET /health`) provides essential system status information including status and uptime in seconds, with plans for future expansion to include additional operational metrics such as hospital connectivity system performance indicators.

HTTP verb /endpoint	Functionality
`GET /`	Returns a “Hello” message
`GET /health`	Returns app status and uptime

Table 2 – IMS Basic endpoints

2.2.2 Hospital Information Endpoints:

The hospital management endpoints are designed to support other services that need to interact with the IMS for data querying. In BIO-STREAMS this is particularly about the Dashboard component. Since the IMS bootstraps with a configuration that includes all available hospitals, any service or component that wants to send queries to the IMS must first retrieve the list of available hospitals and their identifiers. The `GET /hospitals` endpoint provides this essential information, returning the list of configured hospitals with their public details (no sensitive information is returned), enabling client services to determine which hospital IDs they can include in their query requests.

The `GET /hospitals/{hospital_id}` endpoint provides detailed information about a specific hospital, included for API completeness and to support services that need detailed hospital metadata for their query planning.

HTTP verb /endpoint	Functionality
`GET /hospitals`	Returns list of configured hospitals in IMS
`GET /hospitals/{hospital_id}`	Returns specific hospital details

Table 3 – IMS Hospital information endpoints

2.2.3 Query Processing Endpoints:

The `POST /queries/` endpoint serves as the primary interface for data retrieval, enabling client services to query clinical and synthetic data across multiple hospital Node Bundles. The endpoint accepts query requests that specify target hospitals, query parameters, and data requirements, then processes these requests by distributing them to the appropriate hospitals and aggregating the responses. Current performance testing has shown query response times for requesting all the available data at around 15 seconds, making the current synchronous approach acceptable for most use cases.

The `POST /queries/jobs` and `GET /queries/jobs/{job_id}` endpoints represent a work-in-progress backup approach designed for scenarios where query processing times exceed acceptable limits for synchronous requests. This asynchronous job processing system would enable the IMS to handle complex queries that require extended processing times, returning results minutes after the initial request rather than requiring clients to wait for the

entire processing duration. However, given the current performance characteristics, the system continues to rely on the simpler synchronous approach for query processing.

HTTP verb /endpoint	Functionality
`POST /queries/`	Main query endpoint for clinical and synthetic data retrieval
`POST /queries/jobs`	Asynchronous job creation (future implementation)
`GET /queries/jobs/{job_id}`	Job status checking to be used for “polling” (future implementation)

Table 4 – IMS Query-processing endpoints

Example query payload:

The query payload combines requirements from both the IMS and the underlying Node Bundle abstraction. The `query` and `fields` parameters are required by the Node Bundles and define the specific data retrieval criteria and response format. The IMS requires the `hospital_ids` array to determine which hospital nodes should receive the query request, while the `data_type` field specifies whether to request `"clinical"` or `"synthetic"` data (with synthetic data generation currently in development). Additional helper fields include `requester_id` and `requester_name` for logging and audit purposes, enabling the IMS to track which services are making requests and maintain comprehensive request logs.

```
{
  "hospital_ids": ["UKCM", "NKUA"],
  "query": {
    "VS": {
      "$elemMatch": {
        "VSTEST": "Systolic Blood Pressure",
        "VSORRES": { "$gte": 130 }
      }
    }
  },
  "fields": {"USUBJID": 1, "VS": 1},
  "data_type": "clinical",
  "requester_id": "dashboard-service",
  "requester_name": "Main Dashboard"
}
```

2.2.4 API documentation:

FastAPI [1] automatically generates comprehensive OpenAPI [8] documentation for all endpoints, providing interactive Swagger UI [9] pages (see Figure 4) that enable developers to explore API capabilities, test endpoints directly, and understand request/response schemas.

The generated documentation includes detailed parameter descriptions, example payloads, and response models, facilitating integration with client services and providing a self-documenting API interface.

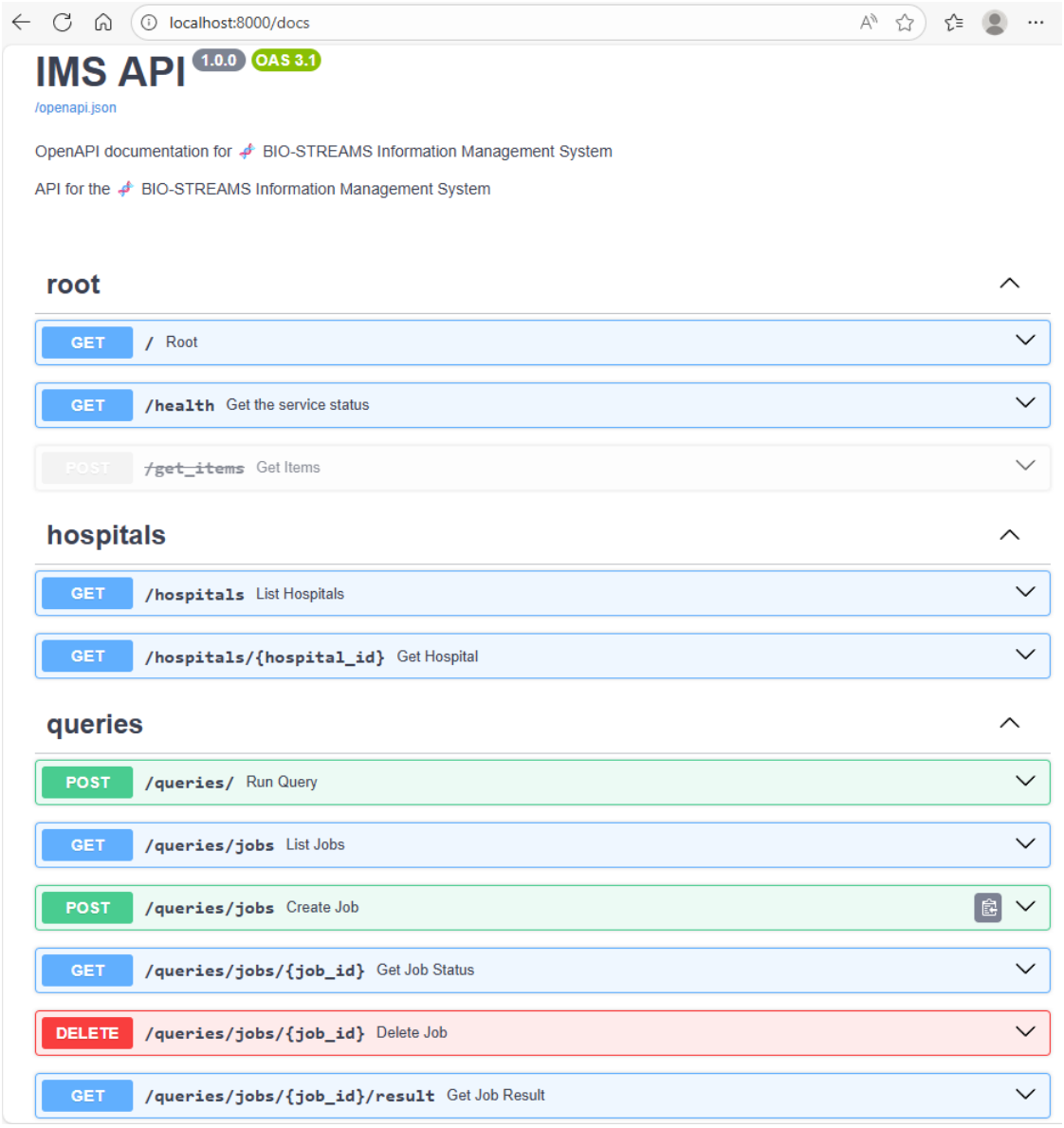


Figure 4 – IMS OpenAPI documentation (Swagger UI)

2.2.5 Authentication and Security Implementation

The system implements a multi-layered security approach:

- **Network Isolation:** Node Bundles are not exposed to the outside world. The Node Gateway maintains persistent SSH connections to each Node Bundle, creating an SSH-tunnel approach where the IMS can only access Node Bundles through the Node Gateway. Since the IMS and Node Gateway are co-hosted containers within the same trusted environment, their communication is considered secure and reliable.
- **Hospital Authentication:** On top of the Network isolation, each hospital Node Bundle requires username/password authentication for API access and IMS requests data through protected endpoints.

- **Token Management:** To optimize performance authentication tokens are generated and cached and if they expire IMS automatically requests new ones
- **Request Validation**: All incoming requests are validated against defined schemas with Pydantic [2] to prevent malformed queries

2.3 Configuration Management

2.3.1 Modern Container Configuration Strategy

In today's containerized microservices architecture, flexible configuration management is critical for successful deployment and operations. The IMS implements a hybrid configuration approach that combines multiple strategies to address different operational needs:

Environment Variables and CI/CD Integration:

- **Jenkins Pipeline Integration**: Environment variables are set within Jenkins [**Error! Reference source not found.**] pipelines and injected into containers during deployment, enabling automated environment-specific configuration
- **Container Runtime Configuration**: The system leverages Docker [10] environment variables to configure application behaviour at runtime, eliminating the need for container rebuilds when configuration changes
- **Developer Workflow Support**: Local development is facilitated through `.env` files that developers can customize without affecting shared configuration

File-Based Configuration:

- **Hospital Configuration Files**: The system uses JSON configuration files (`hospitals.local.json` & `hospitals.dev.json`) to preload hospital-specific settings, credentials, and connection parameters during its bootstrap process.
- **Environment-Specific Files**: Different configuration files are maintained for different environments, ensuring proper separation between development and production settings

2.3.2 Bootstrap Process and Startup Logging

The application implements comprehensive startup logging for debugging and monitoring:

- **Environment Detection:** Startup logs display the detected environment and configuration file path
- **Hospital Loading:** The system logs the loading process for each hospital configuration
- **Validation Results:** Startup logs confirm successful loading of all hospital configurations
- **Error Handling:** Configuration errors are logged with detailed error messages to facilitate troubleshooting

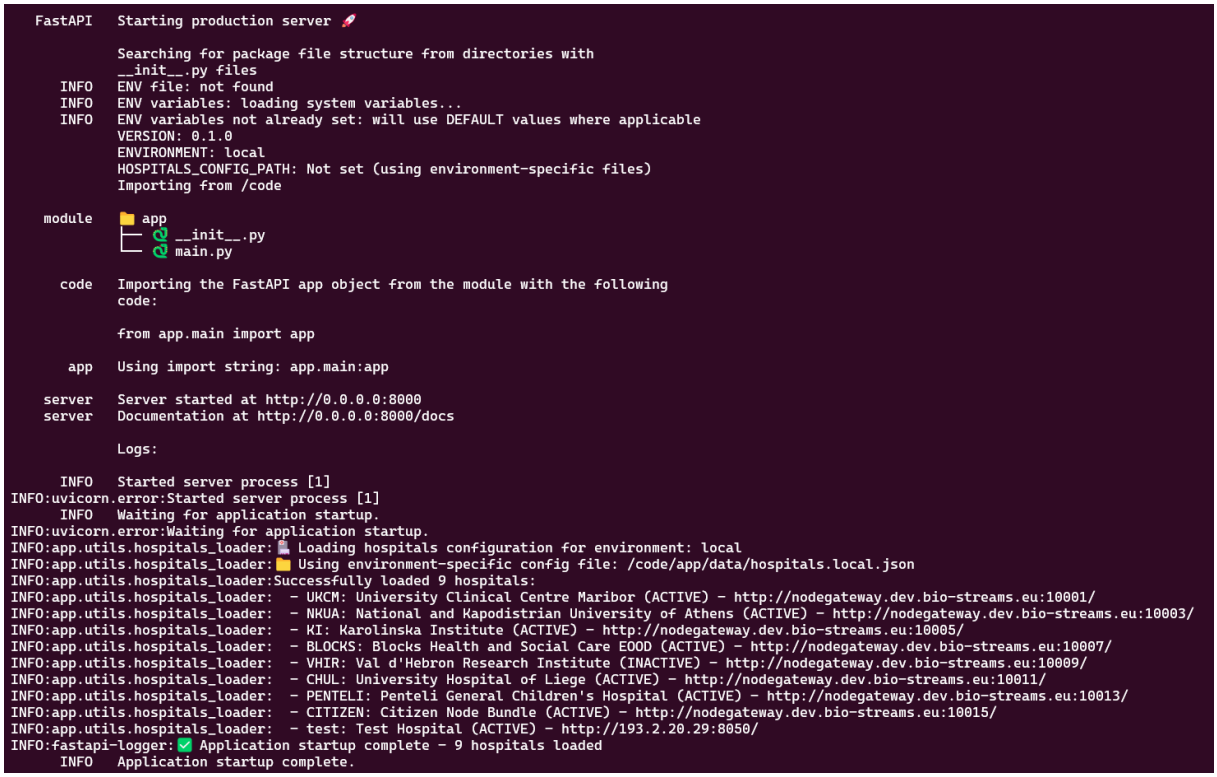


Figure 5 – Console logs of IMS bootstrap process

2.3.3 Environment-Specific Configuration Files

The system employs environment-specific configuration management:

- **Local Development Environment:** Uses `hospitals.local.json` with development URLs and test credentials
- **Development Environment:** Uses `hospitals.dev.json` with container-based URLs as accessing the Node Gateway takes place directly on the same host
- **Environment Detection:** The system automatically detects the deployment environment through the `ENVIRONMENT` variable
- **Configuration Validation:** All hospital configurations are validated against Pydantic [22] models to ensure data integrity

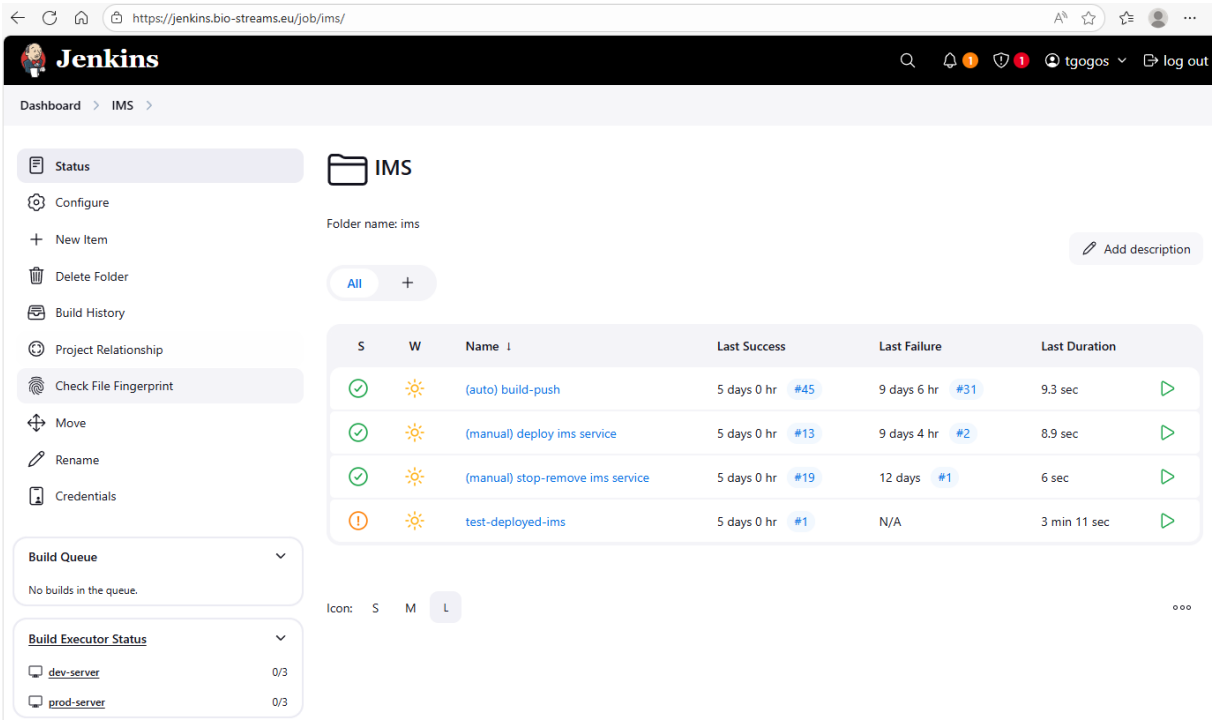
3 Deployment and Operations

3.1 Repository and Version Control

The IMS source code is maintained in a Git [5] repository following standard version control practices. The repository structure encompasses the core FastAPI [1] application code organized in a modular structure within the `app/` directory, environment-specific configuration files stored in `app/data/` and the current test suite located in `app/tests/` for validation and integration testing. Additionally, the repository includes Jenkins [**Error! Reference source not found.**] pipeline definitions in the `jenkins/` directory for automated deployment and technical documentation for deployment guides.

3.2 CI/CD Pipelines

The system implements a comprehensive CI/CD pipeline architecture consisting of four distinct Jenkins pipelines that manage the complete lifecycle of the IMS deployment:



3.2.1 Build and Push Pipeline:

The build pipeline compiles the Docker [10] image from source code and pushes it to the Harbor registry. This pipeline handles the complete build process, including dependency installation, application packaging, and image optimization, while ensuring that the built image is properly tagged and stored in the Harbor registry for subsequent deployments.

3.2.2 Deploy Pipeline:

The deployment pipeline retrieves the latest image from the Harbor registry and deploys the complete IMS container composition to the target environment. This pipeline orchestrates the

multi-service deployment using Docker Compose, configures environment variables, and verifies successful deployment through container status checks and health monitoring.

3.2.3 Kill Deployment Pipeline:

The kill deployment pipeline provides clean removal of the IMS container composition, ensuring proper cleanup of resources and preventing orphaned containers. This pipeline is essential for maintenance operations, environment resets, and ensuring clean deployment states.

3.2.4 Test Pipeline:

The testing pipeline runs comprehensive validation tests against an already running IMS instance, providing validation of system functionality. This pipeline verifies running container status, validates actual hospital connectivity and authentication though Node Gateway, tests query processing across multiple hospitals, and validates the environment configurations to ensure system reliability.

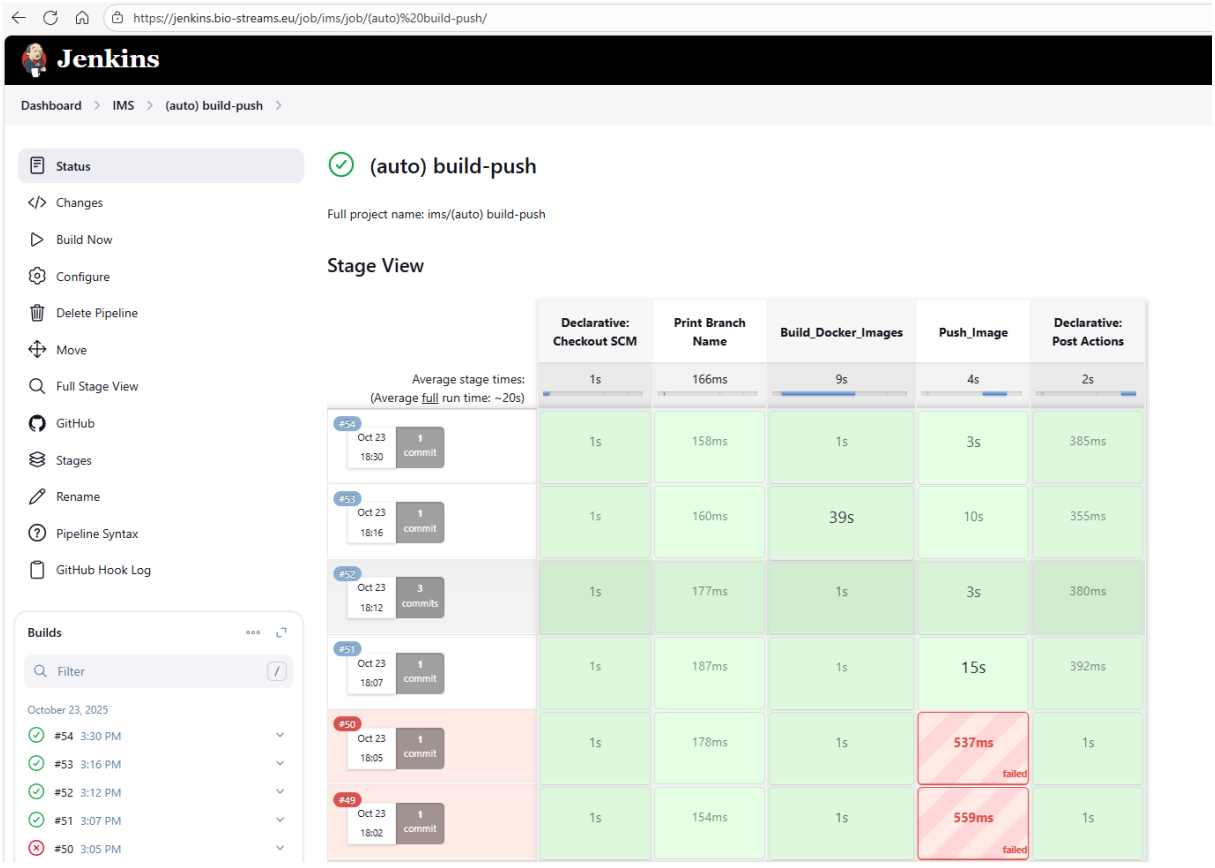


Figure 6 – Example of a Jenkins page for the "Build & Push" pipeline

3.3 Container Deployment

3.3.1 Docker Configuration

The IMS is containerized using Docker [10] with a configuration optimized for performance and security. The Dockerfile utilizes `Python 3.12-slim` as the base image for optimized

container size and includes essential system dependencies such as `curl`, `jq`, and `bc` for testing and monitoring capabilities. The configuration implements automated Python dependency installation and application setup while maintaining security through non-root user execution and minimal attack surface.

The Docker Compose configuration enables multi-service deployment including the IMS, OpenTelemetry [12] collector, and Jaeger for comprehensive observability. The configuration integrates with the Traefik [11] network for external access, implements proper volume management for configuration file mounting and log persistence, and includes container health monitoring with restart policies for reliable operation.

3.3.2 Environment Variable Management

The container deployment implements comprehensive environment variable management to provide flexible and secure configuration. The system uses the `ENVIRONMENT` variable to determine configuration file selection, while managing service configuration including database connections, API endpoints, and service discovery. Security settings encompass authentication tokens, encryption keys, and access controls, while monitoring configuration covers logging levels, metrics collection, and observability settings.

This approach provides deployment flexibility through environment-specific variable sets for development, staging, and production environments. The system implements secure handling of sensitive configuration data and supports runtime configuration changes without container rebuilds, while maintaining environment variable validation and error handling for operational reliability.

4 Testing and Validation

4.1 Test Suite

The IMS implements a comprehensive test suite designed to validate system functionality across multiple dimensions. The testing framework encompasses authentication validation, query performance assessment, and environment-specific testing to ensure reliable operation in different deployment scenarios.

Authentication Testing: forms a critical component of the test suite, validating the connectivity and authentication mechanisms with hospital node bundles. The authentication tests verify that the IMS can successfully establish connections with all active hospital nodes, authenticate using the configured credentials, and retrieve bearer tokens for subsequent API calls. These tests are designed to fail gracefully when hospital nodes are unavailable, providing clear feedback on connectivity issues and enabling troubleshooting of authentication problems.

```
root@46163f86ec13:/code/app/tests# ./test_auth.sh
Testing authentication with active hospitals...
================================================
Testing: University Clinical Centre Maribor (UKCM)
✓ SUCCESS
Testing: National and Kapodistrian University of Athens (NKUA)
✓ SUCCESS
Testing: Karolinska Institute (KI)
✓ SUCCESS
Testing: Blocks Health and Social Care EOOD (BLOCKS)
✓ SUCCESS
⏭  Skipping inactive: Val d'Hebron Research Institute (VHIR)
Testing: University Hospital of Liege (CHUL)
✓ SUCCESS
Testing: Penteli General Children's Hospital (PENTELI)
✓ SUCCESS
Testing: Citizen Node Bundle (CITIZEN)
✓ SUCCESS
Testing: Test Hospital (test)
✓ SUCCESS

Authentication test completed!
```

Figure 7 – Authentication testing of active hospitals

Query Performance Testing: evaluates the system's ability to process queries efficiently across multiple hospital nodes. The test suite includes progressive testing scenarios that begin with single hospital queries and gradually increase complexity by testing combinations of multiple hospitals. This approach provides insights into query performance characteristics, response times, and system scalability under varying load conditions. The performance tests measure query execution times, data volume processing capabilities, and system responsiveness to help identify potential bottlenecks and optimization opportunities.

```
root@46163f86ec13:/code/app/tests# ./test_queries_all_data.sh
Testing query performance with progressive hospital combinations...
================================================================
Found 8 active hospitals: UKCM NKUA KI BLOCKS CHUL PENTELI CITIZEN test

=== Testing Single Hospitals ===
✓ [UKCM] 30.449497878s
✓ [NKUA] 10.732194508s
✓ [KI] 14.896289572s
✓ [BLOCKS] 2.434681483s
✓ [CHUL] 5.289106720s
✓ [PENTELI] 3.009619301s
✓ [CITIZEN] 1.477433940s
✓ [test] 2.582793969s

=== Testing Hospital Combinations ===
✓ [UKCM,NKUA] 10.060458095s
✓ [UKCM,NKUA,KI] 17.556915209s
✓ [UKCM,NKUA,KI,BLOCKS] 16.865629779s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL] 17.806831827s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI] 18.591565689s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI,CITIZEN] 20.467841197s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI,CITIZEN,test] 18.756688987s

Query performance test completed!
root@46163f86ec13:/code/app/tests#
```

Figure 8 – Requesting all available data for Hospital Node Bundles (big response times)

```
root@46163f86ec13:/code/app/tests# ./test_queries_sbp.sh
Testing query performance with progressive hospital combinations...
================================================================
Found 8 active hospitals: UKCM NKUA KI BLOCKS CHUL PENTELI CITIZEN test

=== Testing Single Hospitals ===
✓ [UKCM] 1.323888408s
✓ [NKUA] .134063457s
✓ [KI] .530539472s
✓ [BLOCKS] .409930742s
✓ [CHUL] .491829909s
✓ [PENTELI] .137771491s
✓ [CITIZEN] .570955350s
✓ [test] .294133556s

=== Testing Hospital Combinations ===
✓ [UKCM,NKUA] .809354785s
✓ [UKCM,NKUA,KI] .982335406s
✓ [UKCM,NKUA,KI,BLOCKS] .798349081s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL] 1.191032761s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI] .990822084s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI,CITIZEN] 1.246856383s
✓ [UKCM,NKUA,KI,BLOCKS,CHUL,PENTELI,CITIZEN,test] 1.466606841s

Query performance test completed!
root@46163f86ec13:/code/app/tests#
```

Figure 9 – Requesting records with Blood Pressure above specific threshold (small response times)

Environment-Specific Testing: ensures that the system operates correctly across different deployment environments. The test suite is planned to be used to validate both local development configurations and production environment settings, testing the system's ability to automatically select appropriate configuration files based on environment variables. This testing approach verifies that the IMS can seamlessly transition between local, development and production environments while maintaining consistent functionality and performance characteristics.

4.2 Integration Testing

The previously presented Test Suite practically provides a comprehensive testing of the IMS's ability to communicate over the distributed network of the various hospital Node Bundles through the node-gateway component. The integration tests validate the complete data flow from the IMS through the node-gateway to individual hospital Node Bundles, ensuring that network connectivity, authentication, and data transmission function correctly across this distributed system architecture. These tests verify that the IMS can successfully establish connections with hospital nodes, authenticate requests, and receive responses within acceptable timeframes.

Furthermore, this end-to-end query-processing testing validates the complete query lifecycle from initial request to final response delivery. The integration tests simulate real-world query scenarios, testing the system's ability to receive query requests, distribute them to appropriate hospital nodes, aggregate responses, and deliver unified results to the Dashboard. This testing approach helps us ensure that the IMS can handle complex queries involving multiple hospitals, manage concurrent requests efficiently, and provide reliable data aggregation services for the broader BIO-STREAMS platform.

The integration testing framework also validates error handling and recovery mechanisms, ensuring that the system can gracefully handle network failures, authentication errors, and data validation issues while maintaining service availability and providing meaningful error responses to client applications.

5 Current Status and Future Development

5.1 Implemented Features

The IMS has successfully achieved its core objectives as a central data aggregation and query processing platform for the BIO-STREAMS project. The system currently provides a fully functional FastAPI-based microservice that enables seamless querying of clinical data across multiple hospital node bundles. The implementation includes comprehensive hospital management capabilities, allowing administrators to configure and monitor hospital connections through standardized API endpoints.

The system has successfully implemented environment-specific configuration management, enabling seamless deployment across development and production environments. The containerized deployment architecture provides reliable, scalable operations with automated CI/CD pipelines that ensure consistent deployment processes. The testing framework provides comprehensive validation of system functionality, including authentication testing, query performance assessment, and integration testing across different deployment scenarios.

The IMS currently integrates with seven active hospital nodes across European healthcare institutions, demonstrating the system's ability to handle distributed data sources effectively. The implementation includes robust error handling, comprehensive logging, and performance monitoring capabilities that support operational reliability and troubleshooting.

5.2 Long-running queries

While the current implementation provides a solid foundation for the BIO-STREAMS biobank, limitations might arise as development, testing and actual use of the platform takes place. The system currently focuses on clinical data querying (with synthetic data generation capabilities planned for future implementation) and copes well with the type of queries / amount of data being exchanged. If these parameters change, the asynchronous job processing system for long-running queries (which is not yet implemented) will come into play, enabling the system to handle complex queries that require extended processing times.

5.3 Planned Improvements

Future development will focus on addressing the identified limitations and expanding the system's capabilities based on project requirements and user feedback. The next phase will include implementation of synthetic data generation endpoints, enabling the system to provide both clinical and synthetic data through unified query interfaces. Asynchronous job processing capabilities will be developed to support complex queries and long-running data processing tasks.

The system will be enhanced with improved data transformation capabilities to support more flexible data processing and integration. The planned improvements will also include enhanced monitoring and observability capabilities, providing better insights into system performance and operational metrics. These enhancements will support the system's evolution from a development prototype to a more production-ready platform capable of supporting the needs of the relative BIO-STREAMS users.

Co-funded by the European Union

References

[1] S. Ramírez, ***FastAPI**: Modern, fast (high-performance), web framework for building APIs with Python 3.7*+, GitHub, 2018. [Online]. Available: https://github.com/tiangolo/fastapi

[2] S. Colvin *et al.*, ***Pydantic**: Data validation and settings management using Python type annotations*, version 2.x. GitHub, 2023. [Online]. Available: https://github.com/pydantic/pydantic

[3] K. Kawaguchi *et al.*, ***Jenkins**: An open-source automation server*, Continuous Delivery Foundation, 2011. [Online]. Available: https://www.jenkins.io

[4] ***Python*** is a programming language that lets you work quickly and integrate systems more effectively. https://www.python.org/

[5] ***Git*** is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. https://git-scm.com/

[6] ***GitHub***: a collaboration platform built on top of Git: https://github.com/git-guides

[7] ***Portainer*** delivers an enterprise-grade container management platform, simplified and engineered for everyone. From multi-cluster agility to remote-site efficiency, it empowers IT and OT teams with visibility, control, and ROI, all without complexity or vendor lock-in. https://www.portainer.io/

[8] The ***OpenAPI*** Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. https://swagger.io/specification/

[9] ***Swagger UI*** allows anyone to visualize and interact with the API's resources without having any of the implementation logic in place. It's automatically generated from your OpenAPI (formerly known as Swagger) Specification, with the visual documentation making it easy for back-end implementation and client-side consumption. https://swagger.io/tools/swagger-ui/

[10] ***Docker*** is a platform designed to help developers build, share, and run container applications. https://www.docker.com/

[11] ***Traefik*** is the leading open-source reverse proxy and load balancer for HTTP and TCP-based applications that is easy, dynamic and full-featured. https://traefik.io/traefik

[12] ***OpenTelemetry*** is a collection of APIs, SDKs, and tools. Use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) https://opentelemetry.io/