Detailed Software Specification Request (SSR)

# 1. Project Overview

This project involves developing a sophisticated reactive microservices framework that integrates diverse applications, services, and backends via Semantic Web technologies (RDF Triples) and reactive functional stream programming. The system facilitates the discovery and execution of interactions across integrated applications through inference-based APIs and user interfaces.

# 2. Objectives

- Semantic integration of heterogeneous data sources using RDF triples (W3C RDF standard).

- Implementation of a dynamic inference engine to derive types, states, and interaction contexts using Semantic Web inference methods.

- Providing discoverable and navigable APIs and frontends to access inferred use cases (Contextual API design, Domain-Driven Design (DDD) principles).

- Reactive and incremental data processing using Functional Reactive Programming (FRP) with Spring Reactive (Spring WebFlux, Project Reactor).

References:

- RDF standard: https://www.w3.org/TR/rdf11-concepts/

- Spring Reactive: https://spring.io/reactive

# 3. Scope and Use Cases

The scope covers the full data lifecycle: ETL processing, inference-based aggregation, alignment, activation, and API exposure. Example use cases include:

- Retail scenarios (greengrocer supply-chain management).

- BI reporting and data analytics integration.

References:

- Enterprise Application Integration: https://www.ibm.com/cloud/learn/enterprise-application-integration

# 4. System Architecture

Microservice architecture consists of:

- Datasource Service (ETL to RDF)

- Aggregation Service (Classification-based inference)

- Alignment Service (Ontology matching and clustering)

- Activation Service (Regression-based inference)

- Consumer API Service (REST/HATEOAS frontend)

- Augmentation Service (orchestrating service)

References:

- Microservices: https://martinfowler.com/articles/microservices.html

- Reactive Streams: https://www.reactive-streams.org/

# 5. Component Specifications

Detailed descriptions of each component:

- Datasource Service: Handles ETL from diverse sources to RDF (SPO triples), synchronizing data with backend sources.

- Aggregation Service: Infers types, states, and orders using classification algorithms (ML).

- Alignment Service: Matches entities using ontology alignment (OWL), performs clustering and linking inference.

- Activation Service: Infers context interactions and actor roles, exposing metadata for API navigation.

- API/Frontend Service: Implements discoverable interfaces following REST principles and hypermedia controls (HATEOAS, HAL).

References:

- OWL Web Ontology Language: https://www.w3.org/OWL/

- HATEOAS: https://restfulapi.net/hateoas/

# 6. Data Flow and Inference Layers

- Aggregation: Processes RDF triples to infer entity types (classification), states, and hierarchical orders.

- Alignment: Semantic matching of data entities using ontology matching methods, clustering algorithms, and upper ontology alignment.

- Activation: Infers interaction contexts, actor roles, and interactions via regression techniques.

References:

- Ontology Matching: https://ontologymatching.org/

- Semantic Web Inference: https://www.w3.org/standards/semanticweb/inference

# 7. Technology Stack

- Spring Boot with Reactive Extensions (Spring WebFlux, Project Reactor)

- RDF4J / Neo4j graph databases for semantic data storage

- Machine Learning: Classification, Clustering, Regression (TensorFlow/PyTorch integration for inference)

- NLP and LLM support for semantic processing (spaCy, OpenAI API integration)

- Web3/DID identifiers for decentralized semantic interoperability

References:

- Spring WebFlux: https://docs.spring.io/spring-framework/reference/web-reactive.html

- RDF4J: https://rdf4j.org/

- Neo4j: https://neo4j.com/

- DID Identifiers: https://www.w3.org/TR/did-core/

# 8. Integration and Interfaces

System integration achieved through APIs exposing context-aware interactions, navigable using hypermedia controls. Frontend services dynamically adapt to inferred contexts.

References:

- REST API design: https://restfulapi.net/

- Domain-Driven Design (DDD): https://domainlanguage.com/ddd/

- Hypermedia (HAL): https://stateless.group/hal\_specification.html

# 9. Administration & Helper Services

Detailed administration interface for each microservice, backed by helper services:

- Registry Service: URI-based CRUD and provenance tracking.

- Naming Service: NLP/NER-based semantic resolution.

- Indexing Service: Embedding-based similarity resolution for contexts/interactions.

References:

- NLP (spaCy): https://spacy.io/

- Embeddings (OpenAI Embeddings): https://platform.openai.com/docs/guides/embeddings

# 10. Next Steps / To-Do

- Formalize RDF schema definitions for each component input/output.

- Define algorithms for incremental graph parsing and inference.

- Establish synchronization patterns between microservices.

- Develop detailed administration UIs leveraging modern frontend frameworks (React, Vue.js).

References:

- RDF Schema (RDFS): https://www.w3.org/TR/rdf-schema/

- Reactive programming patterns: https://projectreactor.io/docs