POLITECNICO DI TORINO

MASTER's Degree in COMPUTER ENGINEERING

MASTER's Degree Thesis

DMP Evaluation Service Design

Supervisors

Dott. Alessandro FIORI

Dr.techn. Tomasz MIKSA

Candidate

Andres TABIMA

December 2025

DMP Evaluation Service Design

Andres Mauricio Tabima Romero

Abstract

Research data management (RDM) is a key aspect of the research lifecycle, enabling organizations and researchers to plan, control, and track research data through living documents known as Data Management Plans (DMPs). DMPs are crucial for reproducibility and scientific progress. However, their assessments are usually done manually, which involves time-consuming and subjective processes. In this work, we build upon previous work to redesign and implement an improved DMP Evaluation Service that automates the assessment of machine-actionable DMPs (maDMPs). The service improves the modularity, extensibility and interoperability in the results by automating the evaluation process through a modular design that integrates metrics, benchmarks, and tests. The outcome is an extendable API service that can be integrated with external DMP platforms and that ensures automated, transparent, and interoperable DMP assessment at scale for stakeholders.

ACKNOWLEDGMENTS

To my parents Yasmin and Alejandro, for their unconditional love, for always believing in me, and for teaching me the values that shaped who I am. To my family in Colombian for always being there for me.
To my girlfriend, whose love, patience, and incredible support carried me through every step of this thesis.
To my Colombian friends in Turin who were my family and support during all the process of my master.
To my supervisors, whose guidance, trust, and insightful direction made this work possible. A special thank to Tomasz Miksa for his valuable feedback and constant support and mentorship.
This thesis is dedicated to all of you, with deep gratitude.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In recent years, research data management (RDM) has become an essential part of the research lifecycle. Funding agencies, institutions, and researchers are required to plan, assess, and track the management of research data through structured documentation, commonly known as Data Management Plans (DMPs). These plans are crucial for ensuring data FAIRness (Findability, Accessibility, Interoperability, and Reusability), compliance with institutional and legal requirements, and for the overall sustainability of research outputs.
Despite their importance, DMPs are often reviewed manually, making the assessment process time-consuming, subjective, and inconsistent. Moreover, actors like researchers and reviewers who write and asses DMPs frequently lack timely or actionable feedback to improve the quality and completeness of the plans. This creates a gap in the RDM ecosystem: There is currently no widely adopted automated and standarized mechanism to evaluate DMPs in a structured, objective, and interoperable way.
This thesis addresses this gap by reviewing previous research, improving designs, and implementing a software service that automates the evaluation of some aspects of machine-actionable DMPs (maDMPs). The system is aligned with the objectives of the OSTrails project, an initiative aimed at planning, tracking, and assessing RDM practices. Specifically, it implements the core concepts from the Assessment Framework, which defines how digital objects, including DMPs, can be evaluated through interoperable benchmarks, metrics, and tests.

1.2 Motivation

The topic of this thesis originated from the importance of the DMP in the ecosystem of data management and the need to check this document to evaluate the strategies and the consistency of the projects that declare the information and data into the data management plans, however, in the current workflow and the use of the DMPs, the manual work and specific knowledge to get a revision are still present.

Despite recent research and developments in DMP practice, there is still a gap in how the evaluation of a DMP can be automated.
The motivation of this thesis is to reduce the manual workload in evaluating DMPs by providing an automated way to assess specific aspects of the DMPs and generating reusable formats of the results evaluation that can be used across tools. By making the assessment process machine-actionable and interoperable, the implemented system contributes to a more efficient and transparent research data practice, facilitating the evaluation of DMPs.

1.3 Problem Statement

Although DMPs are increasingly required across research domains, their evaluation remains largely manual, subjective, and lacks standardization. Researchers often submit plans without clear guidance on their quality, and reviewers spend valuable time checking compliance without automated support. Moreover, existing tools do not offer integrated or extensible frameworks for formal assessment, especially in a way that is compatible with the maDMP concept.
This thesis seeks to solve the problem of non-automated, non-standardized DMP evaluation by creating a configurable, extensible, and interoperable evaluation service.

1.4 Objectives

The main objective of this thesis is to evaluate the design and the implementation of a web-based evaluation service for maDMPs. The system allows the definition of benchmarks, composed of metrics and tests, which can then be used to assess the contents of a machine-actionable DMP. The evaluation results are produced in structured formats (e.g., JSON, JSON-LD) that follow the interoperability principles defined in the OSTrails Assessment Framework [1].

1.5 Research Questions

To address the identified problem and achieve the stated objectives, this thesis is guided by the following research questions. They aim to explore how an automated evaluation service can be designed, implemented, and integrated within the existing Research Data Management (RDM) ecosystem. The questions address the alignment between conceptual models and technical implementation, the efficiency of an semi-automated assessments, interoperability with external systems, and the design of an API that enables integration with other DMP-related platforms.

1. In what way the architecture of the DMP service needs to be revised to better reflect the real world requirements of production ready systems?

2. In what way the data model of the DMP service needs to be revised to align with FTR and to support the architectural changes?

3. How to integrate the Assessment Framework with the DMP Evaluation Service in order to provide standardized evaluation results ?

1.6 Methodology

The methodology of this thesis follows a design science research approach, which is well-suited for studies aiming to create and evaluate technological artifacts that address practical problems. In this context, the artifact is a web-based DMP Evaluation Service that automates the assessment of machine-actionable Data Management Plans (maDMPs).
The research process was organized into four main stages:

1. **Conceptual analysis and requirements definition:** A review of the state of the art in Data Management Plans, machine-actionable DMPs, and assessment frameworks was conducted to identify the existing challenges and requirements for automation and interoperability. This analysis established the functional and quality requirements that guided the system's design.

2. **Architectural design:** Based on the identified requirements, the system architecture was specified using the arc42 documentation template and the C4 model. These methods provided a structured way to describe the system context, containers, components, and runtime interactions, ensuring traceability between requirements and design decisions.

3. **Implementation:** The architecture was realized in Kotlin using the Spring Boot framework. The implementation includes a RESTful API, a plugin mechanism for extensibility, and a coroutine-based parallel execution model for scalability. MongoDB was used as the persistence layer to store benchmarks, metrics, tests, evaluations, and results.

4. **Evaluation and validation:** The implemented prototype was evaluated against the functional requirements and research questions. Functional validation was performed through end-to-end workflow testing and API-level verification, while non-functional aspects such as scalability, robustness, and extensibility were assessed through targeted experiments. The evaluation demonstrates that the proposed system effectively automates and contribute to standardizes the maDMP assessment process.

The methodology process ensures that the thesis not only contributes a functional prototype but also provides a reproducible and well-documented output justifying its design and implementation choices.

1.7 Contribution

The key contributions of this thesis are as follows:

- Design and development of a configurable evaluation service for maDMPs, implemented in Kotlin using the Spring Boot framework.
- Integration of a benchmark-metric-test model from OSTrails project that enables transparent, reusable, and modular assessment configurations.
- Implementation of an interoperable API for evaluating maDMPs and returning structured results aligned with OSTrails' assessment interoperable framework specifications.
- Extension of prior work conducted at Lukas Arnold thesis at TU Wien, improving the initial prototype's design and functionality.

1.8 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** reviews the background of research data management, DMPs, and related evaluation tools and frameworks.
- **Chapter 3** identifies requirements for the development of the DMP Evaluation Service.
- **Chapter 4** presents the architecture and design of the proposed evaluation service.
- **Chapter 5** details the implementation of the system, including its components, technologies, and API structure.
- **Chapter 6** discusses the evaluation of the system using example maDMPs and analyzes its strengths and limitations.
- **Chapter 7** concludes the thesis and outlines potential directions for future work.

Chapter 2

Related work

2.1 Data Management Plan (DMP)

Data Management Plans (DMPs) are structured documents that outline how research data will be generated, described, stored, shared, and preserved throughout and beyond a project's lifecycle. They formally address the what, how, who, and where of data management, ensuring alignment with FAIR principles (Findability, Accessibility, Interoperability, and Reusability). Traditionally, DMPs are text-based and manually reviewed, which makes evaluation subjective and time-consuming. Studies show that while manual review provides detailed insights, automated analyses can efficiently detect whether key elements such as availability, metadata, and sharing practices are addressed [2].
DMPs are living documents in research projects. Usually, they are developed six months after a project starts and updated during the project and at the end. Different stakeholders are involved in each stage, and information changes according to the progress.

2.2 Machine-Actionable DMP (maDMP)

Even with the growing adoption of DMPs, many challenges remain when ensuring that DMPs are actionable, consistent, and interoperable across tools and institutions. Therefore, there have been ongoing efforts to clarify maDMPs requirements [3]. In general, maDMPs encode the content of traditional plans in a structured, interoperable format to enable automated exchange, validation, and reuse across research systems. The RDA DMP Common Standard and its Application Profile define the core elements and JSON serializations that make data management plans computable [4]. Unlike traditional DMPs, maDMPs require a data model that captures information in a structured way and can integrate with other services to retrieve details for the DMP (e.g., institutional affiliation from the ORCID).

2.3 RDA DMP Common Standard

The RDA DMP Common Standard (RDCS)[1] is a metadata application profile designed to express the contents of a Data Management Plan in a machine-actionable, interoperable format. It was developed by the DMP Common Standards Working Group under the Research Data Alliance to address the limitations of free-text, tool-specific DMPs and facilitate automated exchange, validation, and integration of DMP data.
At its core, the RDCS defines a set of classes, properties, and relationships for representing DMPs, such as contact, project, dataset, distribution, license, and security/privacy. Some fields are mandatory (e.g. title, dmp_id, language, ethical_issues_exist) in its JSON schema (maDMP-schema-1.1.json), while others are optional and may be extended in specific deployments.

2.4 DMP Tools

A Data Management Plan (DMP) tool is a software application that helps researchers and institutions in the creation, management, and review of Data Management Plans. These tools usually provide guided templates, structured metadata fields, and support for complying with institutional or funder requirements. Modern DMP tools adopt *machine-actionable* DMPs (maDMPs), where the DMP is represented in a structured, machine-readable format that enables automation and interoperability with external systems.
In the context of this thesis, DMP tools play a complementary role to the DMP Evaluation Service. A typical workflow is the following:

1. A researcher or institution creates a DMP using a DMP tool.
2. The resulting machine-actionable DMP (maDMP) is submitted to the DMP Evaluation Service as input.
3. The Evaluation Service processes the maDMP, executes benchmarked tests and metrics, and produces structured evaluation results.
4. These results can be returned to the DMP tool or consumed by other systems, enabling iterative improvement of the DMP or integration into institutional and funder workflows.

By positioning DMP tools as producers of machine-actionable DMPs and the DMP Evaluation Service as the automated assessment component, this thesis establishes an end-to-end workflow: from *plan creation* to *automated evaluation*. This integration supports one of the key goals of the thesis: enabling seamless, interoperable, and machine-actionable research data management practices that enhance the quality, transparency of DMPs.

[1] https://github.com/RDA-DMP-Common/RDA-DMP-Common-Standard

2.5 Assessment for maDMP

The practices used in research data management vary significantly across disciplines, making the assessment process complex for reviewers and heavily dependent on their expertise [5]. Several studies have proposed methods for automating the assessment process to address this challenge.
Tomek et al. [6] Miksa et al. [6] proposed a group of methods that can be integrated into an automated toolbox for evaluating maDMPs. The methods covered are the following:

- **RDF-Based Validation:** The idea is to leverage Shapes Constraint Language (SHACL), an approved standard language for validating RDFs [citation], to enforce constraints in the maDMPs, for instance, in the data types or the vocabularies for property values. This can be adapted to the requirements of a specific funding agency.
- **Identifier Analysis:** The idea is to verify that objects in the form of URL (e.g., DOIs) point to reachable links using the HTTP response and to check if the resource is accessible or forbidden, and map the results to the requirements from the funder or organization.
- **Using information from link resources**: A similar idea to the previous method, but apart from checking the validity of a resource, it also considers fetching additional details of the specific resource. For example, metadata registries can be used to check whether the metadata standard fits the domain.
- **Using scientific knowledge graphs**: query additional information (e.g., through existing knowledge graphs as OpenAIRE[citation or footnote]) to get a broader domain context and check for inconsistencies.
- **Fairness Assessment:** consider FAIR metrics to evaluate aspects of the maDMP and contrast the results with requirements from funders. This can be achieved by instantiating existing FAIRness evaluator tools and integrating results as a checking criterion that needs to be fulfilled by the maDMP.

The main limitation stressed out by authors is the unstructured nature of data provided by maDMPs. Here, techniques from natural language processing (NLP) field can be helpful.

2.6 maDMPpy Library

Ballesteros-Rodriguez et al. [7] recently developed madmpy, a Python library for creating and validating DMPs. The library validates DMPs using the RDA DMP standard and provides functionality to create, modify, and update DMPs. Additionally, different versions of the RDA DMP standard can be chosen, and the validated DMP can be exported in JSON format.

2.7 Assessment Framework and Prototype by Lukas Arnold

The work of Lukas Arnold [8] established the foundation for the DMP Evaluation Service by introducing a conceptual and prototype framework for the automated evaluation of machine-actionable Data Management Plans (maDMPs). His thesis, proposed a comprehensive architecture that connects research data management practices with automated quality assessment mechanisms.

2.7.1 General Description

In his conceptual design, Arnold followed a structured enterprise and software architecture methodology combining TOGAF and the arc42 framework. The goal was to describe a reference architecture capable of supporting automated evaluation workflows for maDMPs. The system aimed to provide measurable quality indicators derived from DMPs, supporting reviewers and other stakeholders in assessing compliance with data management standards. The framework emphasized interoperability and reusability by modeling its core entities—*metrics*, *dimensions*, and *categories*—using established standards such as the Data Quality Vocabulary (DQV), Dublin Core (DC), and PROV vocabularies.

2.7.2 Component Architecture

The proposed architecture is composed of modular components organized into two primary services: the **DMP Harvester Service** and the **DMP Indicator Service**.

- **DMP Harvester Service:** This service is responsible for collecting and normalizing DMP data and related contextual information. It is composed of several subcomponents:
  - *DMP Loader* – retrieves and normalizes maDMPs into the standardized DCS format.
  - *Context Loader* – gathers external contextual data from systems such as SKGs and repositories.
  - *Inference Engine* – derives additional information from linked data by applying reasoning rules.
  - *Data Provider* – aggregates the retrieved information and provides unified access to extended DMP data.
- **DMP Indicator Service:** This service coordinates the actual evaluation process. It comprises:
  - *Evaluation Manager* – orchestrates evaluation workflows and manages communication between components.

- *Evaluation Provider* – connects to individual *Evaluator* components that compute quality indicators for specific dimensions.
- *Measurement Aggregator* – aggregates metric values and computes averages to produce reports.

Additional components include the *Data Store*, which persists maDMPs, contextual data, and evaluation results.

Each component exposes standardized interfaces, such as `Load DMP`, `Load Context`, and `Evaluate DMP`, enabling clear separation of responsibilities and extensibility.

2.7.3 Data Architecture

Arnold's data architecture was built upon the **Data Quality Vocabulary (DQV)**[2], a W3C recommendation that provides a standardized model for representing data quality information in RDF. The DQV enables expressing quality measurements, metrics, and their relationships to data resources in a machine-readable way.
In his thesis, the **DMP Quality Vocabulary (DMPQV)** was proposed as an extension of DQV, adapting its core concepts to the evaluation of Data Management Plans. The main components of DQV and their use in the DMPQV model are summarized as follows:

- **dqv:QualityDimension** – Represents a high-level aspect or dimension of quality, such as completeness, accessibility, or reusability. In the DMPQV, this class was used to group metrics according to conceptual categories relevant for DMP assessment.
- **dqv:Metric** – Defines a measurable criterion for evaluating a specific aspect of a data object. In the DMPQV, each metric corresponds to a concrete check or rule that can be applied to a DMP, such as verifying the presence of licensing information or the availability of persistent identifiers.
- **dqv:Measurement** – Represents the result of applying a metric to a resource. In the DMPQV, measurements link to specific DMP statements and record the computed value, provenance, and timestamp.
- **dqv:QualityMeasurementDataset** – A collection of quality measurements produced in an evaluation process. This allows the aggregation of results across metrics, supporting comparative and reproducible assessments.
- **dqv:ComputedOn** and **dqv:Value** – Properties connecting measurements to evaluated DMP entities and storing the measured values (numeric, boolean, or textual).
- **dqv:hasQualityMeasurement** – Used to associate a DMP or related resource with one or more quality measurements.

[2] `https://www.w3.org/TR/vocab-dqv/`

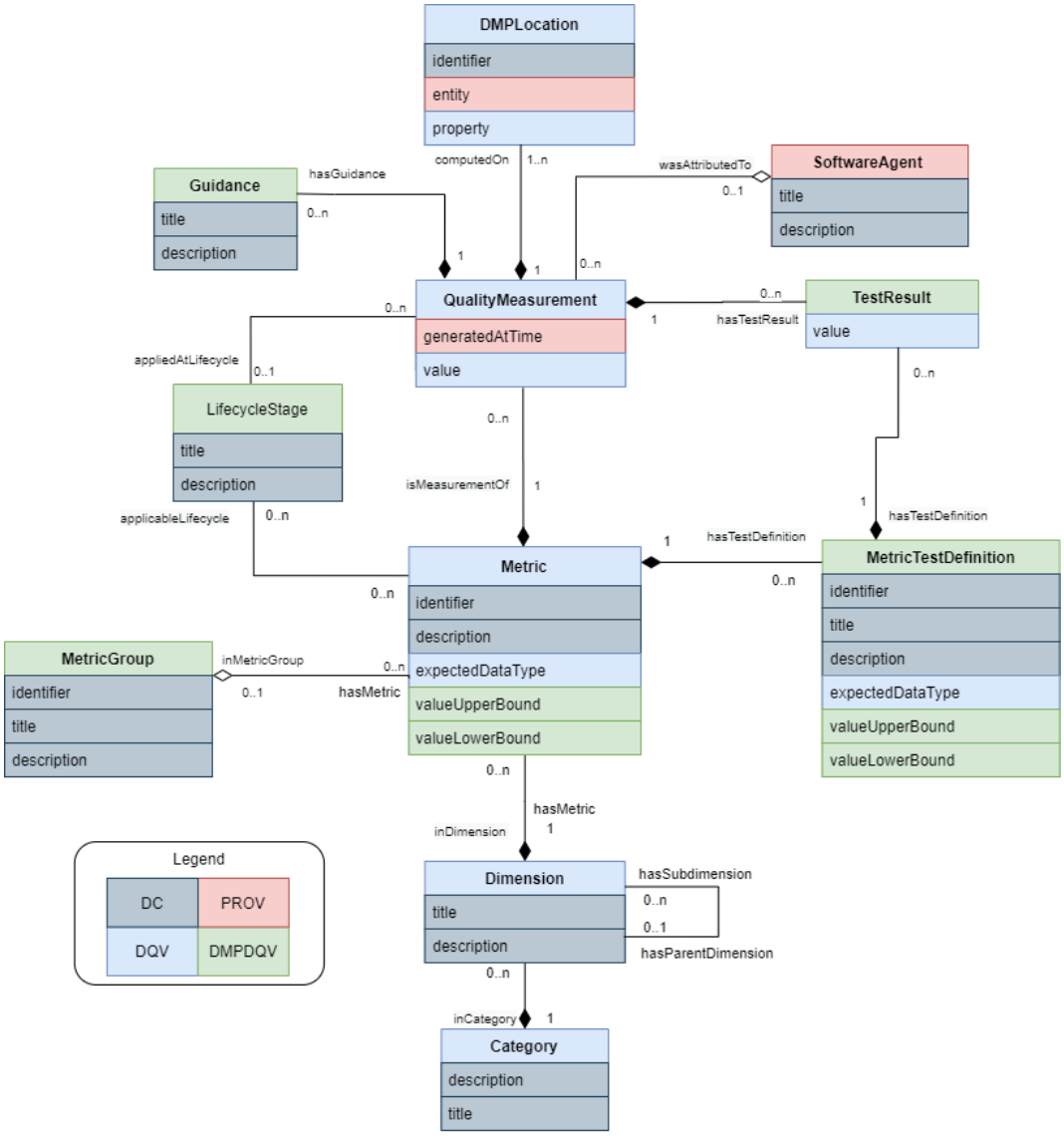

Figure 2.1: Depiction of the data structure used to specify metrics (DMPQV) and measurements based on Data Quality Vocabulary (DQV)

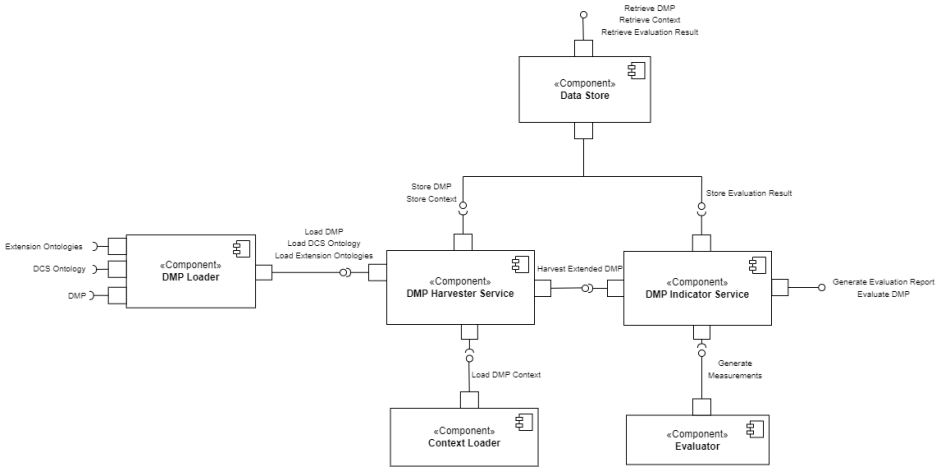

Figure 2.2: Enter Caption

To ensure interoperability, the model also reused terms from other vocabularies, such as `prov:wasGeneratedBy` (PROV-O) to describe provenance, and `dcterms:creator` (Dublin Core) for metadata attribution. This RDF-based representation allowed the system to export evaluation results as Linked Data, enabling reuse and integration with other FAIR assessment tools.

2.7.4 Application Architecture

The application architecture presented in Arnold's work defines how the DMP Evaluation Service interacts with both internal components and external systems. It follows a service-oriented structure that separates data harvesting, contextual enrichment, and evaluation logic into independent services.
At a high level, the architecture consists of two main application layers: the **DMP Harvester Service** and the **DMP Indicator Service**. These communicate through standardized data models based on RDF.

- **DMP Harvester Service:** Responsible for retrieving, normalizing, and enriching DMP data. It includes the following components:
  - *DMP Loader* – imports machine-actionable DMPs (maDMPs) and transforms them into a normalized internal representation.
  - *Context Loader* – retrieves complementary contextual information from external sources such as repositories, funders, or research organization databases.
  - *Inference Engine* – infers additional facts or relationships using semantic reasoning, extending the available DMP data.
  - *Data Provider* – exposes the aggregated and enriched DMP data to other services through a unified interface.
- **DMP Indicator Service:** Performs the actual evaluation of DMPs and computation of quality indicators. It is composed of:

  - *Evaluation Manager* – orchestrates the evaluation workflow by coordinating communication between components.
  - *Evaluation Provider* – connects to one or more *Evaluator* components that implement concrete assessment logic for specific metrics or dimensions.
  - *Measurement Aggregator* – collects and aggregates the computed quality measurements to produce reports and overall quality indicators.

- **Data Store:** A persistence layer used to store DMPs, contextual data, computed measurements, and aggregated reports. The data store supports RDF serialization to maintain compatibility with Linked Data standards.

The system interacts with several external actors, including:

- **DMP Sources**, such as maDMP repositories and authoring tools that provide the input data.
- **FAIR Evaluators** and **Knowledge Graphs**, which contribute external validation and contextual information.
- **Data Repositories** and **External APIs**, serving as additional information sources for context retrieval.

The architecture was documented using the **arc42 template** and represented through **C4 model diagrams**. Runtime views described key operational workflows, such as loading DMPs, enriching them with contextual information, performing evaluations, and generating quality reports. Together, these views form a complete blueprint for implementing an extensible and interoperable DMP evaluation ecosystem.

2.8 FAIR Testing Resource Vocabulary (FTR)

The **FAIR Testing Resource Vocabulary (FTR)** was developed as part of the *Open Science Trails (OSTrails)* [1] initiative to provide a standardized, machine-readable model for representing tests, metrics, benchmarks, and evaluation results used in FAIR assessments [1]. The goal of the FTR is to enable interoperability among assessment tools, support benchmarking of FAIRness and other quality dimensions for digital objects, and facilitate the aggregation and comparison of results across platforms.
The vocabulary defines the core concepts involved in an evaluation workflow and how they are related through well-defined RDF and PROV-O properties. It builds on existing W3C standards such as the *Data Quality Vocabulary (DQV)*, *PROV-O*, and *Dublin Core*, ensuring compatibility with the Semantic Web ecosystem and other FAIR maturity assessment frameworks.

2.8.1 Main Components

The principal classes and properties of the FTR are summarized below:

- **ftr:Metric** – Describes a measurable criterion or rule that a test should verify. Metrics represent the conceptual layer of assessment and are typically human-defined and domain-specific.
- **ftr:Test** – Represents an implementation of a Metric. Tests correspond to executable services or functions that evaluate whether a resource meets a specific criterion. The relation between a Test and its Metric is expressed using the property `sio:isImplementationOf`.
- **ftr:TestResult** – Captures the outcome of executing a Test. It includes the evaluation result (e.g., *Pass*, *Fail*, or *Indeterminate*), provenance metadata, timestamps, and references to the resource under assessment. TestResults are modeled as extensions of `prov:Entity` and are linked to the corresponding Test via `ftr:outputFromTest`.
- **ftr:TestResultSet** – Groups multiple TestResults that were produced during a single evaluation session or activity. This class supports traceability and shared metadata across result sets.
- **ftr:Benchmark** – Defines a community-specific collection of Metrics that together describe a higher-level evaluation objective, such as assessing FAIRness, completeness, or accessibility. Benchmarks are used to organize and interpret groups of Tests and TestResults.
- **ftr:ScoringAlgorithm** or **ftr:Algorithm** – Specifies a computational procedure for aggregating multiple TestResults within a Benchmark to produce a quantitative score or summary indicator.
- **ftr:BenchmarkScore** – Represents the computed result of applying a ScoringAlgorithm to a Benchmark. It includes a numeric or qualitative score, optional explanatory notes, and links to the underlying TestResults.
- **ftr:TestExecutionActivity** – A `prov:Activity` that denotes the execution event of a Test or set of Tests, producing one or more TestResults or TestResultSets. This allows detailed provenance tracking of evaluation processes.

2.8.2 Interoperability and Standardization

The FTR reuses and extends established vocabularies such as DQV, PROV-O, and DCAT, ensuring semantic interoperability with other data quality and FAIRness assessment efforts. By representing Tests, Metrics, Benchmarks, and Results as Linked Data, FTR enables transparent publication, exchange, and aggregation of evaluation information across heterogeneous systems. This standardization facilitates automated FAIR assessments, reproducibility of results, and meta-analysis of evaluations across tools and platforms.

2.8.3 Relevance to This Thesis

In the context of this thesis, the FTR serves as the reference model for representing evaluation components and outputs generated by the DMP Evaluation Service. By aligning the system's data model with FTR, the service ensures that evaluation results are interoperable and can be exchanged with other tools and registries following the same standard. This alignment directly supports the thesis objective of providing standardized and machine-actionable evaluation results, as well as the research question concerning the integration of the Assessment Framework with the DMP Evaluation Service.

2.9 Discussion

The work of Lukas Arnold [8] established the conceptual and architectural foundations for the DMP Evaluation Service. His prototype successfully demonstrated how Data Management Plans (DMPs) can be assessed automatically through a structured model of metrics, dimensions, and categories. The architecture introduced a modular approach, separating data harvesting, context enrichment, and evaluation processes into independent components. Furthermore, the use of the Data Quality Vocabulary (DQV) provided a formal basis for representing evaluation results and quality indicators as Linked Data.
However, the system was designed primarily as a proof of concept and therefore some limitations need to be addressed to transform it into a production-ready and interoperable service. First, the architecture lacked a mechanism for scalable execution and extensibility. Evaluator components were statically defined, and no mapping mechanism was available to allow developers to easily integrate new evaluation logic. Second, the data model, while conceptually aligned with DQV, was not fully integrated with existing community standards for assessment tools and did not provide a standardized way to express test results, provenance, and benchmark relationships. Third, the overall service communication relied on static data exchange, which limited its capacity for dynamic integration with other RDM or FAIRness assessment platforms.
To overcome these limitations, this thesis proposes improvements to the architecture and data model:

- The service architecture is redesigned to include a **plugin mechanism** that supports dynamic registration and execution of evaluation logic, improving modularity and maintainability.
- The data model is revised and extended to align with the **FAIR Testing Resource (FTR) Vocabulary**, ensuring that benchmarks, metrics, tests, and test results follow a standardized and interoperable representation.
- The system exposes an **API-first design**, allowing external DMP tools, FAIR evaluators, or other research data management systems to interact with the

evaluation service programmatically.

In addition, the integration of the FTR model provides a semantic foundation for representing and sharing evaluation results. While Lukas Arnold's prototype used the DQV as the main conceptual basis, the FTR vocabulary extends this by introducing classes and properties specifically designed for FAIR testing, such as Test, Metric, Benchmark, and TestResult. By adopting this vocabulary, the DMP Evaluation Service ensures that its evaluation outputs are machine-actionable, interoperable with other tools in the FAIR assessment ecosystem, and reusable for meta-evaluation and benchmarking studies.
Therefore, the present thesis builds upon the original architectural vision of Lukas Arnold, refining its design for robustness, scalability, and interoperability, while aligning the system's data representation with the FAIR Testing Resource Vocabulary to support standardized and transparent automated assessment of machine-actionable Data Management Plans.

Chapter 3

Requirements

This chapter describes the identified requirements for the development of the DMP Evaluation Service, providing the conceptual design of the solution. The section presents the stakeholders, followed by the listing of use cases, then describes the functional requirements, and the quality goals for the system. The collected requirements represent an interpretation of the results found in previous work on the topic of maDMPs and from discussions with the consortium of the OSTrails project.

The DMP Evaluator service checks specific aspects of different dimensions of a maDMP and returns the result of the evaluation align with the assessment framework designed in OSTrails project. This has been created to support interoperability between stakeholders who need to assess DMPS.

3.1 Roles of Stakeholders

In the publication of [5] "Ten principles for machine-actionable data management plans" nine roles in the research community that can be benefit from the maDMPS were described. See the figure 3.1 for the list of roles.
In Lukas Arnholds' thesis [8] he grouped the stakeholders described in [5] "Ten principles for machine-actionable data management plans" into three groups: DMP Maintainer, Reviewer, and Review System Facilitator. See table 3.1
In this thesis we agree with the grouping from Lukas thesis Arnhold [8], but we

Figure 3.1: Stakeholders from [5] "Ten principles for machine-actionable data management plans"

Role Name in "Ten principles paper"	Role Name in Lukas´Thesis
Funder	Reviewer
Ethics review	Reviewer
Legal Expert	Review System Facilitator
Research	DMP Maintainer
Publisher	Review System Facilitator
Repository operator	DMP Maintainer
Infrastructure provider	Review System Facilitator
Research support staff	DMP Maintainer
Institutional administrator	Review System Facilitator

Table 3.1: Mapping for Miksa [5] paper and Lukas' thesis [8] roles

Role name	Description	Expectations
DMP Maintainer	The Principal Investigator and collaborators who write the DMP	Evaluate the DMP in order to improve its quality based
Review System Facilitator	An expert in a specific field who reviews the DMP within institutions	Evaluate the DMP to check specific institutional or domain-specific field requirements
Reviewer	Funding agencies and foundations that specify requirements for DMPs and monitor compliance	Evaluate the DMP to verify if compliance with the funders' requirements.
DMP tool	Tool that provides researchers with guidance and formatting for writing the DMP	Evaluate the DMP while its users are writing it n order to highlight missing or misaligned information.
DMP Evaluation Service Maintainer	A developer who is in charge of maintaining and implementing additional tests for the service	maintain the codebase and implement new tests for the DMP Evaluator Service.

Table 3.2: Roles of Stakeholders

identified some other stakeholders who can benefit from the use of a DMP Evaluation Service when implemented as a software service, such a DMP tools.
Additionally, for the service, we identified that a maintainer of the proposal system is also a stakeholder due to the interaction with the service. The criterion for selecting these roles was to answer the following question. Which of these roles are involved in the process of assessing the maDMPs in an automated or semi-automated way?

The stakeholder roles that the system considered in defining its functionalities are described in the table 3.2.

3.2 Use Cases

After analyzing the stakeholders identified for the system and considering the recommendations made in the assessment framework defined in the OsTrails project, we described the use cases that the system is designed to support.

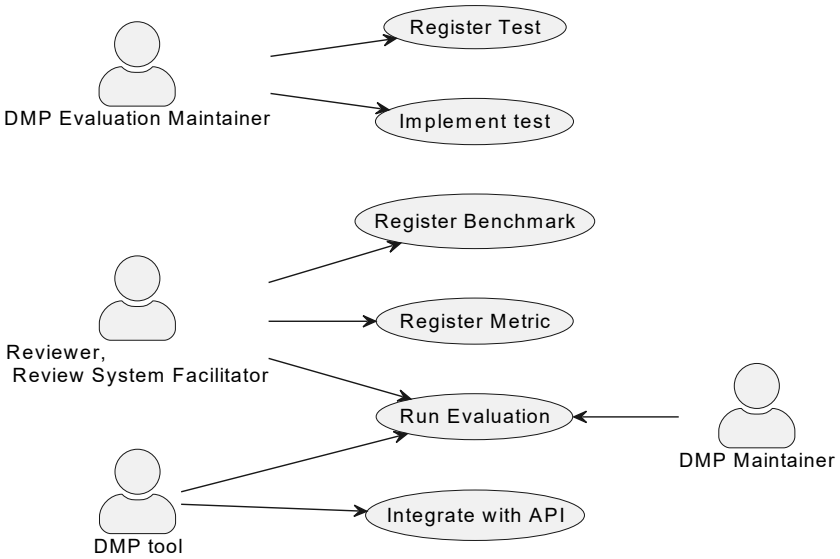

Figure 3.2: Diagram use cases

1. Use Case 1: Register a Benchmark
   - Actor: Review System Facilitator or Reviewer.
   - Description: Register a new benchmark (a high level evaluation scenario or goal).
   - Preconditions: No preconditions.
   - Main Flow:
     (a) User provides metadata for the benchmark.
     (b) System validates the input.
     (c) System stores the benchmark in the database.
2. Use Case 2: Register a Metric
   - Actor: Review System Facilitator or Reviewer.
   - Description: Register a new metric to evaluate a particular aspect of a DMP.
   - Preconditions: It is possible that the metric is created specifically to be added to a benchmark, but is not mandatory to have a benchmark before the metric.
   - Main Flow:
     (a) User provides metadata for the metric.

(b) System validates the input.

(c) System stores the metric in the database.

3. Use Case 3: Register a test
   - Actor: DMP Evaluation Developer.
   - Description: Register a new test with its conceptual metadata(e.g., test purpose).
   - Preconditions: Metric exists.
   - Main Flow:
     (a) User provides declarative test details (e.g., what to test, criteria).
     (b) System validates the input.
     (c) System stores the metric in the database.
4. Use Case 4: Implement a test
   - Actor: DMP Evaluation Developer.
   - Description: Provides a code implementation for a registered test.
   - Preconditions: Test record exists.
   - Main Flow:
     (a) Developer accesses the list of declared tests.
     (b) Developer submits the code that implements the logic of a test.
     (c) System validates and links the implementation with the tests record.
5. Use Case 5: Run Evaluation
   - Actor: DMP Maintainer - Reviewer - DMP tool - Review system facilitator.
   - Description: Executes a set of test implementations on a machine-actionable DMP..
   - Preconditions: Test implementations exist.
   - Main Flow:
     (a) User select the maDMP to assess and the test or benchmark to execute.
     (b) System run each test implementation.
     (c) System provide the evaluation results .

3.3 Functional Requirements

Based on the selected stakeholders and the identified use cases from sections 3.1 and 3.2, we established the functional requirements for the system design proposal in this thesis. Table 3.3 lists these functional requirements, which are described in detail later in this section.
Some of the functional requirements listed in the table are defined based on the recommendation from the assessment framework developed within the OSTrails project *OSTrails/FAIR_testing_resource_vocabulary* [1].

Id	Requirement
FR-01	The system shall allow Review System Facilitators or Reviewers to register a new benchmark with metadata.
FR-02	The system shall allow Review System Facilitators or Reviewers to register a metric with metadata.
FR-03	The system shall allow DMP Evaluation Developers to register a test with conceptual metadata.
FR-04	The system shall allow DMP Evaluation users to access declared benchmarks, metrics, and tests registered in the system.
FR-05	The system shall allow developers to add source code implementing a test.
FR-06	The system shall allow linking between the test implementation and its corresponding test metadata record.
FR-07	The system shall allow users to select a machine-actionable DMP and the tests or benchmarks to execute.
FR-08	The system shall execute the selected test implementations.
FR-09	The system shall return evaluation results and possible failure explanations.

Table 3.3: Functional requirements

3.3.1 FR-01 - Register a Benchmark

Description: The system shall allow Review System Facilitators or Reviewers to register a new benchmark, including the required metadata such as name, description, digital object, and evaluation criteria.
Justification: Benchmarks define high-level evaluation scenarios or goals against which Data Management Plans (DMPs) can be assessed. Without registered benchmarks, evaluation processes cannot be standardized or compared across different DMPs. This requirement was derived from the OSTrails assessment framework.

3.3.2 FR-02 - Register a Metric

Description: The system shall allow Review System Facilitators or Reviewers to register a new metric, including all required metadata such as name, description, digital object, and evaluation criteria.
Justification: Metric define a specific granularity level of evaluation scenarios or goals against which Data Management Plans (DMPs) can be assessed. Without registered metrics, evaluation processes cannot be standardized or compared across different DMPs. This requirement was derived from the assessment framework.

3.3.3 FR-03 - Register a Test

Description: The system shall allow developers to register a new test, including all required metadata such as name, description, digital object, and evaluation criteria.

Justification: Tests represent the implementation of a metric used to assess specific aspects of a Data Management Plan (DMP). Without registered tests, evaluations cannot be automated or systematically repeated. This requirement is derived from the OSTrails assessment framework.

3.3.4 FR-04 - Access Declared Benchmarks, Metrics, and Tests

Description: The system shall allow users to access and review information about all registered benchmarks, metrics, and tests. This includes metadata such as the name, description, associated digital object, evaluation criteria, creation date, and the actor who registered the item. The system shall present this information in a clear and organized format, enabling users to easily browse, search, and filter the available evaluation components.
Justification: Providing users with access to the complete list of registered benchmarks, metrics, and tests ensures that the evaluation process is transparent. This functionality also promotes reusability, as users can identify existing evaluation components that may be relevant to their own assessments. This requirement is aligned with best practices in research data management and contributes to openness, a principle emphasized in the OSTrails assessment framework.

3.3.5 FR-05 - Add Test Implementation Source Code

Description: The system shall allow developers to provide source code that implements a registered test. This functionality must support associating the implementation with its corresponding test metadata, ensuring that the code can be executed as part of the evaluation process.
Justification: Tests are not operational without their corresponding implementations. Allowing developers to contribute and maintain source code for tests ensures that evaluation criteria can be applied in a computational and reproducible way. This functionality supports modularity, enabling multiple implementations for different environments or use cases.This requirement allows the system to be flexible, maintainable, and open to contributions.

3.3.6 FR-06 - Link test Implementation to metadata

Description: The system shall allow linking between a test implementation and its corresponding test metadata record. This functionality ensures that every piece of executable code is associated with the correct descriptive information, such as the test name, description, and intended evaluation criteria. The link should be established when a test implementation is uploaded or updated and maintained consistently to ensure accurate retrieval and execution.
Justification: Linking the test implementation to its metadata guarantees that the evaluation process is transparent, traceable, and reproducible. Without this association, there is a risk of executing the wrong implementation or misinterpreting test results due to mismatched documentation. This requirement supports the

recommendations of the OSTrails assessment frameworkework for maintaining clear traceabibetween thetween the test logic and its descriptive context.

3.3.7 FR-07 - Select the DMP and Evaluation Component

Description: The system shall allow users to select a machine-actionable Data Management Plan (maDMP) and choose the tests or benchmarks to execute against it.
Justification: This functionality enables targeted evaluation, allowing users to focus on specific aspects of a DMP based on their needs. For example, a funding agency may wish to run only compliance-related tests, while a data steward may prioritize completeness or interoperability checks. By allowing selective execution, the system becomes more efficient, avoiding unnecessary processing and reducing evaluation time.

3.3.8 FR-08 - Execute Test Implementation

Description: The system shall execute the selected test implementations using the chosen machine-actionable DMP (maDMP) as input. During execution, the system shall capture the results in a consistent and structured format, ensuring that they can be stored, retrieved, and analyzed reliably.
Justification: Automated execution of test implementations is essential for delivering timely, consistent, and repeatable evaluations of DMPs. Manual execution would be inefficient, error-prone, and difficult to scale. Automating this process ensures that results are produced under consistent conditions, contributing to reproducibility and reliability.

3.3.9 FR-09 - Return Evaluation Results and Failure Explanations

Description: The system shall present the results of executed tests to the user, including pass/fail outcomes,and detailed feedback. When a test fails, the system should provide clear failure explanations, indicating the nature of the issue. Results should be displayed in a structured, format and be available for interoperable way.
Justification: Providing clear, actionable evaluation results helps stakeholders understand the current state of a DMP and identify areas for improvement. Failure explanations increase transparency and guide users toward compliance or higher quality standards. This requirement supports the emphasis on transparency, feedback loops, and continuous improvement in research data management practices.

3.4 Quality goals

In the context of software engineering, quality goals define the non-functional characteristics that a system should achieve in order to satisfy user expectations and ensure long-term sustainability. These goals go beyond basic functional requirements.

Instead, they focus on how well the system performs its functions, how easily it can be maintained, and how effectively it can be integrated into different environments. For the DMP Evaluation Service proposed in this thesis, quality goals were selected based on the use cases defined in Section 3.2 and the functional requirements described in Section 3.3. The selected goals: Functional suitability, maintainability, operability, compatibility, and transferability.
Table 3.4 presents these quality goals, their priority, and a brief description. The following subsections describe each goal in detail and explain how the functional requirements and use cases support their achievement.

Priority	Name	Description
1	Funtional suitability	System provides functions that meet stated or implied needs
2	Maintanability	Systemn can be modified, correct, adapted or improved due to changes in environment or requirements
3	Compatibility	Two or more systems can exchange information while sharing the same environment

Table 3.4: Quality goals for the service

3.4.1 Functional suitability

Functional suitability ensures that the system provides the necessary functions to meet the stated or implied needs of its users. In this project, this means supporting all actions required to register, manage, and execute evaluations of machine-actionable DMPs. This goal is directly addressed by the functional requirements FR-01 to FR-09, which collectively define the essential operations of the system, including registering benchmarks (UC-1), registering metrics (UC-2), registering and implementing tests (UC-3 and UC-4), and running evaluations (UC-5).

3.4.2 Maintainability

Maintainability refers to the system's ability to be efficiently modified, corrected, adapted, or improved when requirements or environmental conditions change. This is critical in the context of the DMP evaluation domain, where new assessment criteria, benchmarks, and tests may need to be incorporated over time. Functional requirements such as FR-05 (adding source code for tests) and FR-06 (linking test implementations to metadata) directly support maintainability, as they enable updates to test logic without disrupting existing workflows. Use case UC-4 (Implement a test) is also central to this goal, as it describes the process by which developers can extend or improve the evaluation capabilities of the system.

3.4.3 Compatibility

Compatibility is the system's ability to exchange information with other systems while sharing the same operational environment. For the DMP Evaluation Service, compatibility is essential because evaluation results, benchmarks, and tests may need to be integrated with external tools such as DMP platforms, reporting dashboards, or institutional repositories. Functional requirements FR-09 (Return Evaluation Results and Failure Explanations) supports this goal by requiring results to be available in interoperable formats, facilitating integration into broader research data management ecosystems. UC-5 (Run Evaluation) also contributes, as it defines the execution and output generation process that can be consumed by external systems.

Chapter 4

Architecture design

This chapter presents the architectural design of the proposed DMP Evaluation Service, defining how the system will be structured to fulfill the functional and quality requirements specified in Chapter 3. While the previous chapter focused on identifying the system's required capabilities and use cases, this chapter describes the high-level and detailed architecture that enables their implementation.

The architecture is documented following the principles of the arc42 [9], which provides a structured approach to describe both the static and dynamic aspects of a software system. The design addresses the functional requirements (FR-01 to FR-09) and the use cases (UC-1 to UC-5) by defining the system's main building blocks, their interactions, and the environment in which they operate.
The following sections present the system context, the high-level container view, the main software components, and the runtime interactions for key scenarios. Together, these architectural views illustrate how the proposed system will support the functional requirements and the identified quality goals.

4.1 Conceptual design - System Contexts

The system context diagram in Figure 4.1 illustrates the DMP Evaluation Service in relation to its external environment. The diagram defines the system boundaries, showing which functionalities are provided internally and which actors or systems interact with it from the outside.
The main external actors include the Review System Facilitator and Reviewer, who is responsible for registering benchmarks and metrics; the DMP Evaluation Developer, who implements and uploads test logic; and the DMP Maintainer or DMP Tool, which initiate evaluations of machine-actionable DMPs.
Interactions between the system and its environment are conducted primarily through an API, enabling both human users and external systems to register evaluation components, execute assessments, and retrieve results. The DMP Evaluation Service also exchanges information with external sources, such as FAIR assessment tools or related services to validate or corroborate information.

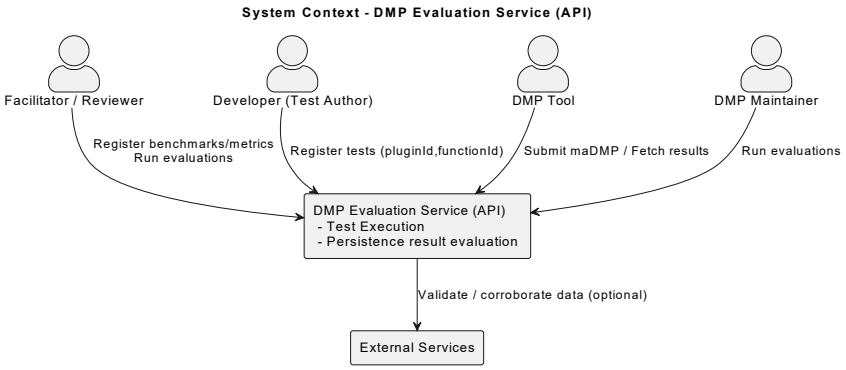

Figure 4.1: System Context

Having defined the external actors, systems, and their interactions with the DMP Evaluation Service in the system context, the next step is to examine the system's internal high-level structure. The container view decomposes the service into its main technical building blocks, referred to as "containers" in the C4 model [10], each representing an application, service, data store, or execution environment. This view illustrates how responsibilities are allocated across containers, how they collaborate to fulfill the functional requirements, and through which interfaces they interact. The following section presents the container diagram of the DMP Evaluation Service, showing the components that collectively implement the evaluation workflow defined in the use cases and requirements of Chapter 3.

4.1.1 Container View

A container diagram in Figure 4.2 illustrates the major building blocks (containers) of the system and how they communicate. Containers are applications, services, databases, or data stores that run independently and together deliver the system's functionality.

The DMP Evaluation Service is exposed exclusively via an application programming interface (API). External clients such as DMP tools, Review System Facilitators/Reviewers, and developers interact with the service programmatically. Internally, the system is organized into three main containers (1) the Application/API responsible for request handling, validation, and orchestration (2) the Plugins, which run test implementations in an isolated runtime and (3) the Persistence layer, which stores benchmarks, metrics, tests, and evaluation results. This organization directly supports FR-01. . . FR-09 and the evaluation workflow in UC-1. . . UC-5.

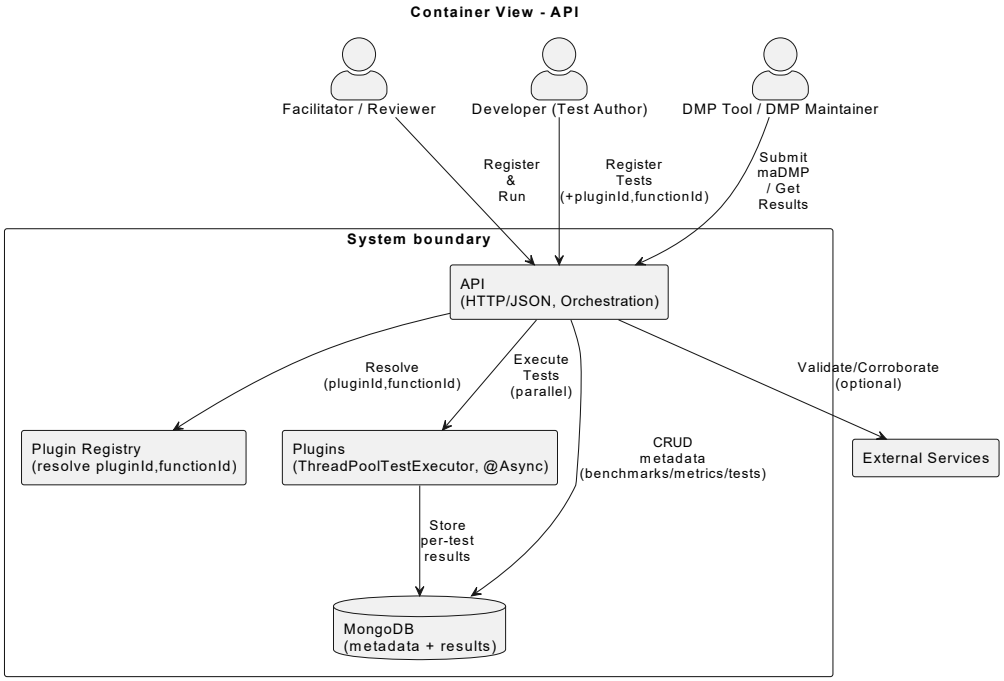

Figure 4.2: Container View DMP Evaluation Service - API

4.1.2 Component View

A component diagram in Figure 4.3 zooms into a container to show the main components, their responsibilities, and relationships. Components group related functionality and encapsulate logic, exposing interfaces to other parts of the system. At the component level, the service separates HTTP controllers (Benchmark, Metric, Test, Assessment) from services that encapsulate business logic. The `AssessmentService` loads the selected `Test` documents, resolves each test's `(pluginId, functionId)` via the `PluginRegistry`, and delegates execution to the corresponding plugin function. Test execution is performed using **Kotlin coroutines** [11], with each test launched as an independent coroutine within a structured concurrency scope. This approach provides lightweight, non-blocking parallelism and ensures that individual `TestResult` records are written incrementally as executions complete. This separation keeps controllers thin, concentrates orchestration logic within services, and isolates execution concerns behind clear boundaries.
The system is structured around the following building blocks:

- Controllers:
  - BenchmarkController, MetricController, TestController, EvaluationController, PluginController
  - Define the REST API endpoints. Each controller is responsible for request handling, validation of input, and delegating logic to the corresponding service layer. The PluginController is a special case: it only provides endpoints for listing available plugins and their functions.

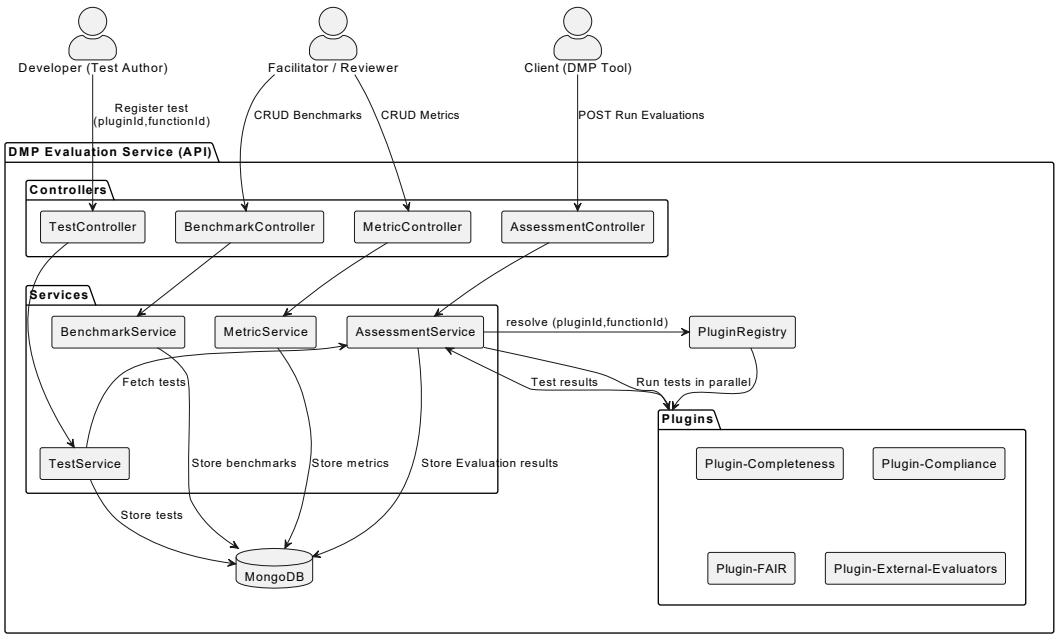

Figure 4.3: Component View – Controllers, Services, Plugins, Parallel Executor

- Services:
  - BenchmarkService, MetricService, TestService, EvaluationService, EvaluationManagerService, PluginManagerService
  - Encapsulate the business logic for each entity. They provide CRUD operations as well as relations between entities, e.g., registering metrics in a benchmark or linking a test to a metric.
  - The EvaluationManagerService orchestrates the execution of evaluations by coordinating which tests must run and how results are aggregated.
- Plugin Executors (Evaluators):
  - Encapsulated units that execute test logic. Each evaluator is mapped to a plugin and function identifier, enabling dynamic test execution.
  - The plugin mechanism allows extending the system with new evaluators without modifying the core code, supporting the quality goal of maintainability.
- Database (MongoDB):
  - Stores benchmarks, metrics, tests, and evaluation results in a flexible schema.
  - Supports entity relationships while allowing heterogeneous structures (e.g., varying test result formats).

This decomposition achieves a clear separation of concerns:

- Controllers remain thin, handling only request/response management.

- Services concentrate domain-specific logic and enforce relations between entities.
- Plugin Executors isolate the execution of test logic, enabling parallel execution and extensibility.
- Database ensures persistence and traceability of all entities and results.

Together, these building blocks implement the functionality required by FR-01 to FR-09 and support the use cases UC-1 to UC-5, while ensuring scalability, transparency, and extensibility.

4.1.3 Sequence View

A sequence diagram describes the dynamic behavior of the system for a specific use case. It shows the order of interactions between components or actors over time to realize the use case.

In UC-5 “Run Evaluation” in Figure 4.4, the client submits a maDMP reference and a benchmark ID. The API validates inputs, fetches the corresponding Benchmark, Metrics, and Test metadata, resolves the associated plugin functions, and then distributes execution in parallel: each test is launched as a separate coroutine within a structured concurrency scope. As coroutines complete, individual `TestResult` documents are persisted incrementally. Once all tests have finished, the system aggregates the outcomes into a single Evaluation record, which is returned to the client through the evaluation endpoint. The response contains aggregated results as well as detailed failure explanations for any unsuccessful tests.

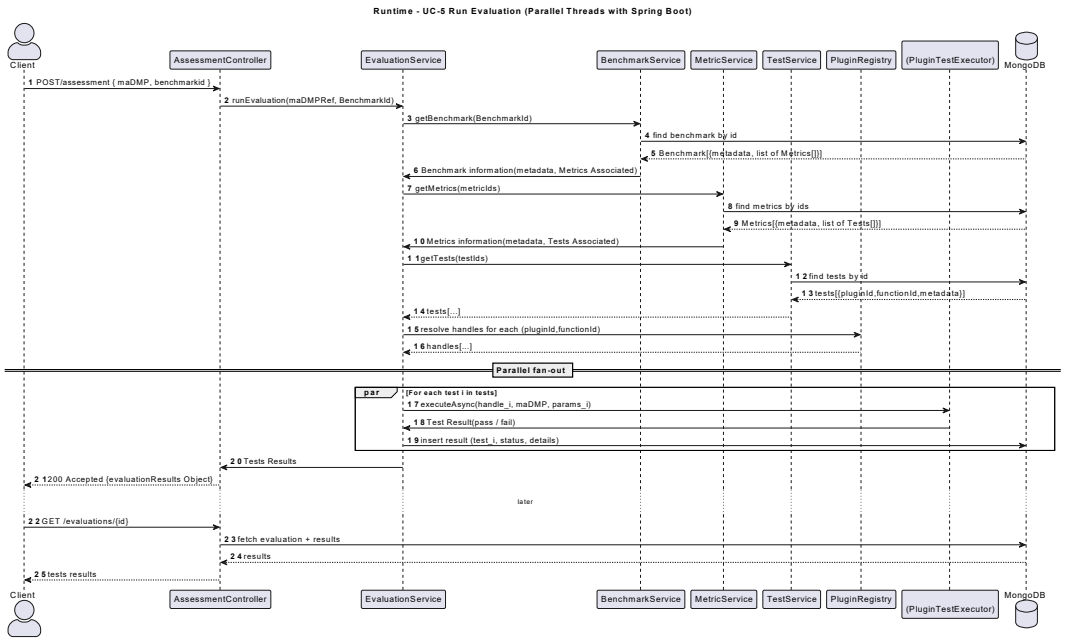

Figure 4.4: Runtime – UC-5 Run Evaluation

4.1.4 Data Model View

The data model view illustrates the core entities of the DMP Evaluation Service and their relationships. It describes how the system organizes, stores, and links information to support evaluation workflows. Figure 4.5 presents the conceptual data model.
This model is aligned with the assessment framework developed in the OSTrails project, which defines benchmarks, metrics, and tests as the key entities for evaluating Data Management Plans. Following this framework ensures that the service adopts established recommendations, enabling interoperability with other tools and supporting standardization of evaluation practices. In addition, the model extends the original framework by introducing two specific fields in the `Test` entity, `pluginId` and `functionId`, which explicitly link each test definition to its executable implementation in a plugin. This design decision enables dynamic resolution and execution of tests, ensuring flexibility and extensibility of the service.
The main entities are:

- **Benchmark**: Defines a high-level evaluation scenario or goal. A benchmark may include multiple metrics.
- **Metric**: Specifies a particular aspect of a DMP to be evaluated. Metrics are grouped under benchmarks and linked to tests.
- **Test**: Represents an executable evaluation procedure for a metric. Each test stores references to a `pluginId` and `functionId`, which bind the test to a concrete implementation.
- **Evaluation**: Represents the execution of one or more tests on a specific maDMP. An evaluation aggregates test executions and provides an overall assessment context.
- **TestResult**: Captures the outcome of a single test execution, including pass/fail status, failure explanations, and additional metadata. Each evaluation contains multiple test results.

The relationships between these entities enable full traceability from a benchmark down to individual test results. For example, a benchmark groups metrics, each metric has associated tests, and tests are linked to their implementations via plugins. Evaluations reference the selected tests and aggregate their corresponding results. This data model ensures transparency, extensibility, and interoperability in managing automated DMP assessments, while adhering to the principles and recommendations of the OSTrails framework [1].

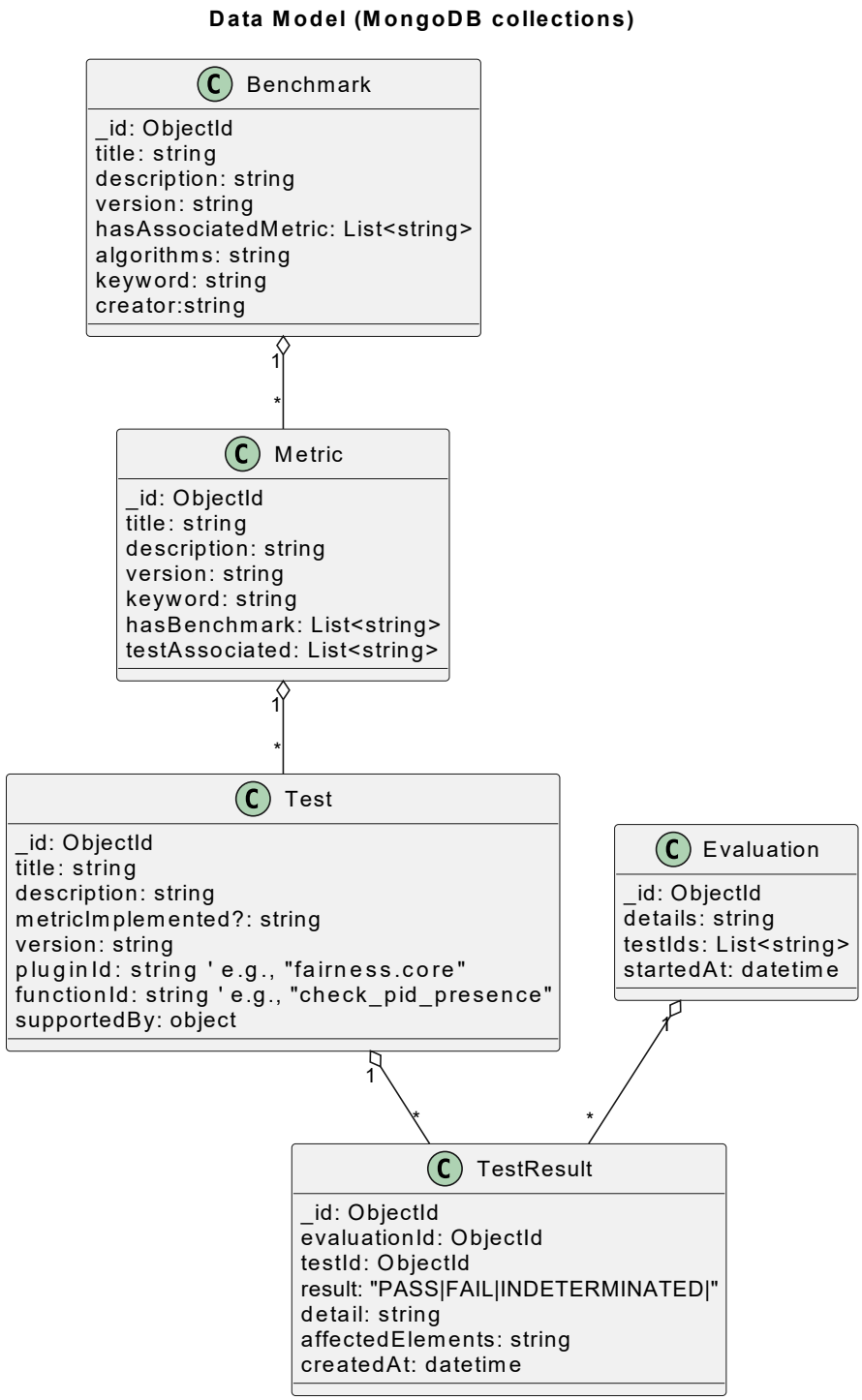

Figure 4.5: Conceptual Data Model of the DMP Evaluation Service

4.2 Description of the workflow

The workflow of the DMP Evaluation Service describes how different users and systems interact with the service to achieve their goals. The system is designed to support both human actors, such as researchers, data stewards, and funding agencies, as well as external tools, such as DMP authoring platforms. By exposing

its functionality through a RESTful API, the service facilitates automation and integration into broader research data management workflows.
The primary goal of users and client systems is to perform an evaluation of a machine-actionable Data Management Plan (maDMP). To enable this, the service provides functionality for registering and managing the necessary evaluation components benchmarks, metrics, and tests, as well as executing these tests and returning the results.
The workflow can be divided into two phases: setup and execution.

1. Setup phase:
    - A benchmark is created to define the high-level evaluation scenario.
    - One or more metrics are registered and linked to the benchmark. Each metric specifies a particular dimension of the evaluation.
    - Tests are defined for metrics. Each test is registered with conceptual metadata (purpose, criteria) and linked to a plugin and function identifier that specify its implementation.
    - Developers implement the test logic inside a plugin, which can contain multiple executable functions. These functions are made available to the service through the plugin registry.
    - The service maintains all benchmarks, metrics, tests, and plugins in its database, ensuring traceability and reusability.
2. Execution phase:
    - A client submits a request to run an evaluation by providing a reference to a maDMP and specifying the benchmark or tests to be executed.
    - The system resolves the required benchmarks, metrics, and tests, and retrieves their associated plugin functions.
    - The AssessmentService distributes the execution of tests in parallel using Kotlin coroutine within a structured concurrency scope . Each test runs independently against the maDMP.
    - As tests complete, the system generates individual TestResult records that capture outcomes and explanations. These are persisted in the database for transparency and later retrieval.
    - Once all tests have finished, the service aggregates the results into an Evaluation object and returns them to the client through the API in a structured and interoperable format.

The workflow also handles error conditions. For example, if a benchmark or metric is not found, the system responds with an appropriate error message without becoming unresponsive. Similarly, if a plugin or function cannot be resolved, the corresponding test is marked as indeterminate and a descriptive explanation is provided.

Finally, the service can interact with external tools and APIs to enrich evaluations. For instance, some tests may query third-party services such as Unpaywall or FAIR assessment tools to verify information beyond what is declared in the maDMP. This ensures that evaluations are not limited to static plan content but can also leverage contextual external data.
Through this workflow, the DMP Evaluation Service ensures that evaluations are machine-actionable, transparent, and extensible, supporting a wide range of stakeholders in improving the quality and compliance of research data management plans.

4.3 Summary

In summary, this chapter has presented the architecture of the DMP Evaluation Service using the arc42 template and C4 model diagrams. The system was described from multiple perspectives, including its external context, high-level containers, internal components, runtime interactions, and data model. Together, these views illustrate how the service is structured to fulfill the functional requirements (FR-01 to FR-09) and use cases (UC-1 to UC-5), while addressing the identified quality goals such as functional suitability, maintainability, and interoperability. The described workflow demonstrates how benchmarks, metrics, and tests are managed and executed in practice, ensuring that evaluations are transparent, reproducible, and extensible. Having established the architectural foundations, the next chapter details the implementation choices and design decisions that realize this architecture in software.

Chapter 5

Implementation

5.1 Introduction

This chapter presents how the architectural design described in Chapter 4 was implemented. It explains the concrete technologies, components, and implementation choices used to fulfill the functional requirements (FR-01 to FR-09) and to support the quality goals defined in Chapter 3.
The implementation follows the architecture: REST controllers expose the service API, services encapsulate business logic, a plugin mechanism binds tests to executable functions via `pluginId` and `functionId`, and a parallel execution layer runs tests concurrently against a machine-actionable DMP (maDMP). MongoDB provides persistent storage for benchmarks, metrics, tests, evaluations, and per-test results.
The remainder of this chapter introduces the technology stack (Section **??**), maps functional requirements to concrete endpoints and components (Section 5.3), details the plugin mechanism and parallel execution model (Section 5.4), describes the persistence layer and data model realization (Section 5.5), discusses error handling and robustness (Section 5.6), and concludes with an end-to-end example workflow (Section 5.7).

5.2 Solution Strategy

First, the system is implemented as a Spring Boot application in Kotlin. Kotlin provides modern language features, seamless Java interoperability, and strong support for concurrency via coroutines. This choice directly supports the quality goals of functional suitability and maintainability, since it enables modularization through controllers and services, while allowing future developers to extend the system with minimal effort.

Second, a plugin-based architecture was adopted to achieve extensibility. Each test implementation is encapsulated within a plugin, identified by a `pluginId` and a `functionId`. This approach ensures that new evaluation logic can be integrated without modifying the core system, thereby reducing coupling and supporting the

quality goal of maintainability.

Third, the system uses parallel execution of tests through **Kotlin coroutines and structured concurrency**. This strategy improves scalability and performance by allowing multiple tests to be executed concurrently, while avoiding the complexity of manual thread management. Coroutines also simplify lifecycle control and error propagation, ensuring predictable and reliable execution.

Fourth, MongoDB was selected as the persistence layer. Its flexible schema is well-suited for storing heterogeneous entities such as benchmarks, metrics, tests, and evaluation results. Moreover, its ability to store nested and variable structures supports the need for extensible test results while maintaining efficient query performance.

Finally, the system is exposed exclusively via a RESTful API, ensuring interoperability with external tools such as DMP authoring platforms, and FAIR assessment services. This aligns with the quality goals of compatibility and transferability, as it allows the service to integrate into different environments and workflows.

5.3 Implementation of Functional Requirements

This section describes how the functional requirements specified in Chapter 3 are implemented in the prototype. Each requirement (FR-01 to FR-09) is mapped to its corresponding API endpoint(s), service components, and persistence layer. This mapping demonstrates traceability from requirement to concrete implementation.

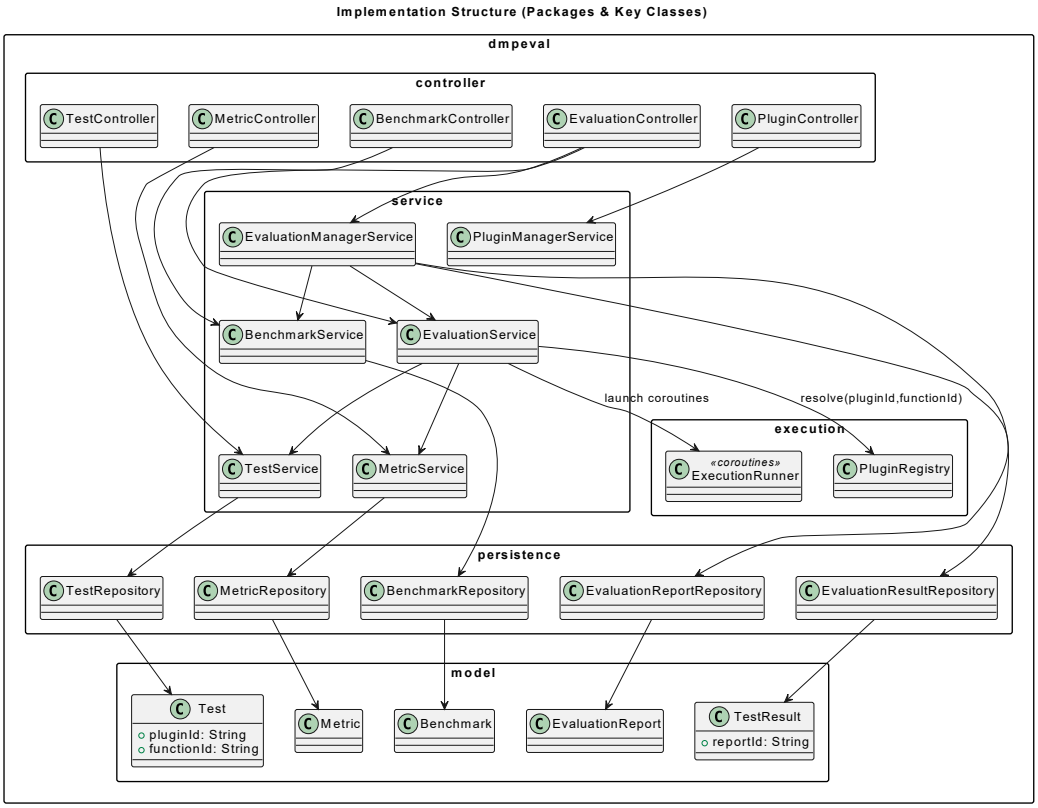

Figure 5.1: Package-level impl structure of the DMP Eva Service

- **FR-01 – Register a Benchmark**
  Endpoint: `POST /benchmarks`
  Controller/Service: `BenchmarkController`, `BenchmarkService`
  Persistence: MongoDB `benchmarks` collection
  Description: Creates and persists a new benchmark with metadata (name, description, criteria). Service ensures validation and uniqueness of benchmark identifiers.

- **FR-02 – Register a Metric**
  Endpoint: `POST /metrics`
  Controller/Service: `MetricController`, `MetricService`
  Persistence: MongoDB `metrics` collection
  Description: Allows registering metrics, linking them to benchmarks. The service enforces consistency of benchmark–metric relationships.

- **FR-03 – Register a Test**
  Endpoint: `POST /tests`
  Controller/Service: `TestController`, `TestService`
  Persistence: MongoDB `tests` collection
  Description: Registers new tests with metadata (purpose, evaluation criteria, pluginId, functionId). Ensures that metadata is complete and references valid metrics.

- **FR-04 – Access Declared Benchmarks, Metrics, and Tests**
  Endpoints: `GET /benchmarks`, `GET /metrics`, `GET /tests`
  Controllers/Services: Corresponding controllers and services
  Persistence: Reads from MongoDB collections
  Description: Exposes API endpoints to retrieve metadata for transparency and integration with external tools.

- **FR-05 – Add Source Code Implementing a Test**
  Mechanism: Plugin system – developers add functions inside plugins
  Controller/Service: `PluginController`, `PluginManagerService`
  Persistence: Plugin registry (in-memory), plugin binaries deployed in execution environment
  Description: Developers provide executable test code inside plugins. The system registers available plugins and their functions for later linking.

- **FR-06 – Link Test Implementation to Metadata Record**
  Endpoint: `PUT /tests/{id}`
  Controller/Service: `TestController`, `TestService`, `PluginManagerService`
  Persistence: Updates MongoDB `tests` collection with `pluginId` and `functionId`
  Description: Ensures that each test is linked to a specific function in a plugin. Validates that identifiers exist in the plugin registry.

- **FR-07 – Select maDMP and Tests for Execution**
  Endpoint: `POST /evaluations`
  Controller/Service: `AssessmentController`, `AssessmentService`
  Persistence: MongoDB `evaluations` collection (initial record)
  Description: Clients submit a maDMP reference and the benchmark or test IDs. Service prepares the evaluation by resolving the selected tests.

- **FR-08 – Execute Test Implementations in Parallel**
  Execution: Kotlin `coroutines` with structured concurrency
  Service: `AssessmentService`
  Persistence: Results stored incrementally in `testResults` collection
  Description: Each test is launched as a coroutine, allowing concurrent execution against the selected maDMP. The coroutine framework provides lightweight, non-blocking parallelism and ensures that results are collected efficiently. Parallel execution improves responsiveness and throughput, particularly for large or complex evaluations.

- **FR-09 – Return Evaluation Results and Failure Explanations**
  Endpoint: `GET /evaluations/{id}`
  Controller/Service: `EvaluationController`, `EvaluationService`
  Persistence: Reads from `evaluations` and `testResults` collections
  Description: Aggregates test outcomes and returns structured results (pass/fail, explanations, metadata). Provides transparency and supports interoperability.

This mapping demonstrates how abstract functional requirements are concretely realized by API endpoints, service components, and persistence structures in the implemented DMP Evaluation Service.

FR	End Point	Controller/Service	Description
FR01	Post /benchmarks	BenchmarkController, BenchmarkService	Creates and persists a new benchmark with metadata (name, description, criteria). Service ensures validation and uniqueness of benchmark identifiers.
FR02	Post /metrics	MetricController, MetricService	Allows registering metrics, linking them to benchmarks. The service enforces consistency of benchmark–metric relationships.
FR03	Post /tests	TestController, TestService	Registers new tests with metadata (purpose, evaluation criteria, pluginId, functionId). Ensures that metadata is complete and references valid metrics.
FR04	Get /benchmarks	Corresponding controllers and services	Exposes API endpoints to retrieve metadata for transparency and integration with external tools.
FR05	Plugin System	PluginController, PluginManagerService	Developers provide executable test code inside plugins. The system registers available plugins and their functions for later linking.
FR06	Put /tests/id	TestController, TestService	Ensures that each test is linked to a specific function in a plugin. Validates that identifiers exist in the plugin registry.
FR07	Post /evaluations	EvaluationController, EvaluationService	Clients submit a maDMP reference and the benchmark or test.
FR08	Kotlin coroutines with structured concurrency	EvaluationService	Each test is launched as a coroutine, allowing concurrent execution against the selected maDMP.
FR09	Get /evaluations/id	EvaluationController, EvaluationService	Aggregates test outcomes and returns structured results (pass/fail, explanations, metadata).

Table 5.1: REST API endpoints and their relation to functional requirements.

5.4 Plugin Mechanism and Parallel Execution

The extensibility of the system relies on a plugin mechanism that allows test logic to be developed and integrated independently of the core service. Each plugin is identified by a `pluginId` and can export one or more functions, each identified by a `functionId`. When a test is registered, its metadata stores both identifiers, ensuring that the system can dynamically resolve and execute the correct function at runtime. The `PluginRegistry` maintains information about available plugins and their exported functions, and acts as the central lookup point for the execution layer.

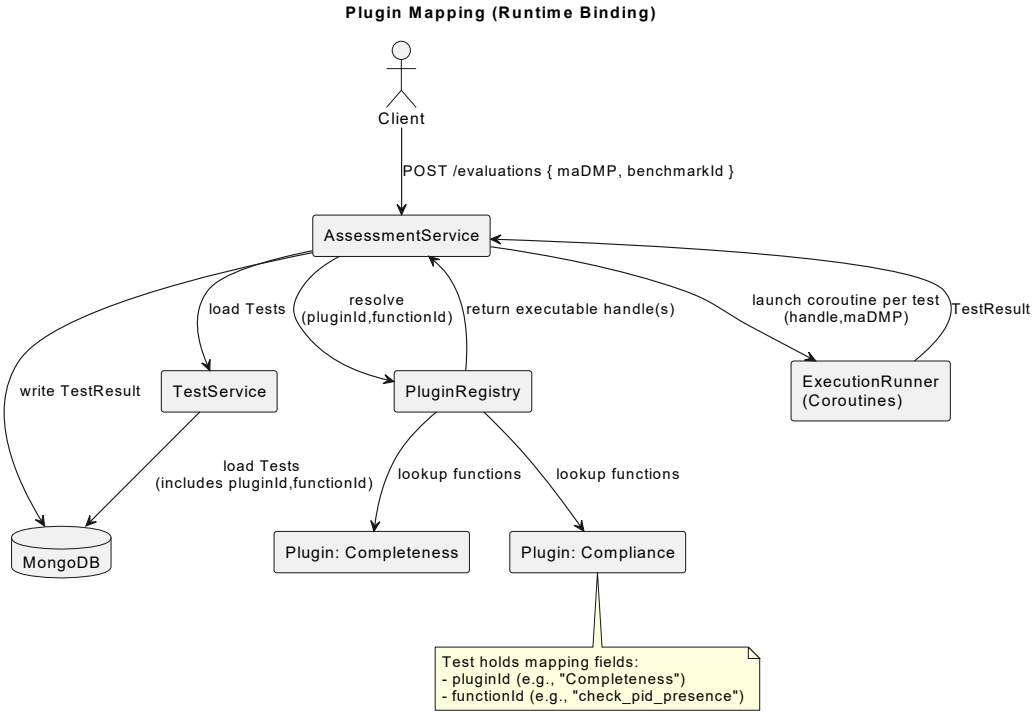

Figure 5.2: Mapping of `Evaluator` and `functionEvaluator`

Test execution is carried out using Kotlin coroutines and structured concurrency. The `AssessmentService` launches one coroutine per selected test, enabling parallel execution while preserving predictable lifecycle management. Results are written incrementally to the database as each coroutine completes. In case of a resolution failure (e.g., plugin or function not found), the test is marked as *indeterminate* with a descriptive explanation. This approach combines flexibility (new plugins can be added without system changes) with scalability (parallel execution of potentially large test sets).

5.5 Data Persistence and Model Realization

The persistence layer is realized using MongoDB, which provides a document-oriented, schema-flexible storage solution. The data model introduced in Chapter 4 is directly mapped to collections:

- `benchmarks`: stores benchmark metadata and references to associated metrics.

Listing 5.1: Example benchmark document in MongoDB

```
{
    benchmarkId: String,
    title: String,
    description: String,
    version: String,
    hasAssociatedMetric: List<String>,
    keyword: String,
    abbreviation: String,
    landingPage: String,
    theme: String,
    status: String,
    creator: List<String>
}

```

- `metrics`: stores metric metadata, each linked to one or more benchmarks.

Listing 5.2: Example metric document in MongoDB

```
{
    id: String,
    title: String,
    description: String,
    version: String,
    testAssociated: List<String>,
    keyword: String,
    abbreviation: String,
    landingPage: String,
    theme: String,
    status: String,
    isApplicableFor: String,
    supportedBy: String,
    hasBenchmark: List<String>
}

```

- `tests`: stores test metadata, including

Listing 5.3: Example test document in MongoDB

```
id: String,
title: String,
description: String,
```

```
    license: String,
    version: String,
    endpointURL: String,
    endpointDescription: String,
    keyword: String,
    abbreviation: String,
    repository: String,
    type: String,
    theme: String,
    versionNotes: String,
    status: String,
    isApplicableFor: String,
    supportedBy: String,
    metricImplemented: String,
    evaluator: String?,
    functionEvaluator: String?,


```

`Evaluator` and `functionEvaluator` that identify the concrete implementation.

- `testResults`: stores the outcome of individual tests, including pass/fail status, explanations, and runtime metadata.

Listing 5.4: Example evaluation document in MongoDB

```
    evaluationId:  String?,
    title: String,
    result: ResultTestEnum,
    details: String,
    timestamp: Instant,
    reportId: String?,
    log: String,
    affectedElements: String?,
    completion: Int?,
    generated: String?,
    outputFromTest: String?

```

- `evaluationReport`: stores the metadata of each evaluation execution, including references to the selected tests and the target maDMP.

Listing 5.5: Example evaluationreport document in MongoDB

```
    reportId: String?
    generatedAt: Instant
```

```
    evaluations: List<String?>
```

This structure provides traceability from high-level benchmarks down to individual test results, supporting transparency and interoperability. MongoDB's flexible schema allows test results to include heterogeneous data structures depending on the nature of the test, while still supporting efficient queries through appropriate indexing (e.g., evaluation IDs, test status).

5.6 Error Handling and Robustness

Robustness is achieved through explicit error handling at both API and execution levels. Typical scenarios include:

- **Entity not found:** If a benchmark, metric, or test does not exist, the API responds with a `404 Not Found` status code and a descriptive error message.
- **External API errors:** If an external service (e.g., Unpaywall) is unavailable, the system records the failure explanation and continues processing the remaining tests without halting the evaluation.
- **Unexpected execution errors:** Exceptions during test execution are caught at the coroutine level. The affected test is marked as failed, while other coroutines continue unaffected.

These mechanisms ensure that evaluations remain resilient, transparent, and informative even under adverse conditions.

5.7 End-to-End Example Workflow

To illustrate the implementation in practice, this section describes an end-to-end evaluation scenario.

1. A user registers a new benchmark by submitting metadata through the `POST /benchmarks` endpoint.
2. Metrics are registered and linked to the benchmark via `POST /metrics`.
3. Tests are registered with metadata, including `pluginId` and `functionId`, via `POST /tests`.
4. A developer implements the test logic in a plugin and makes the plugin available to the system.

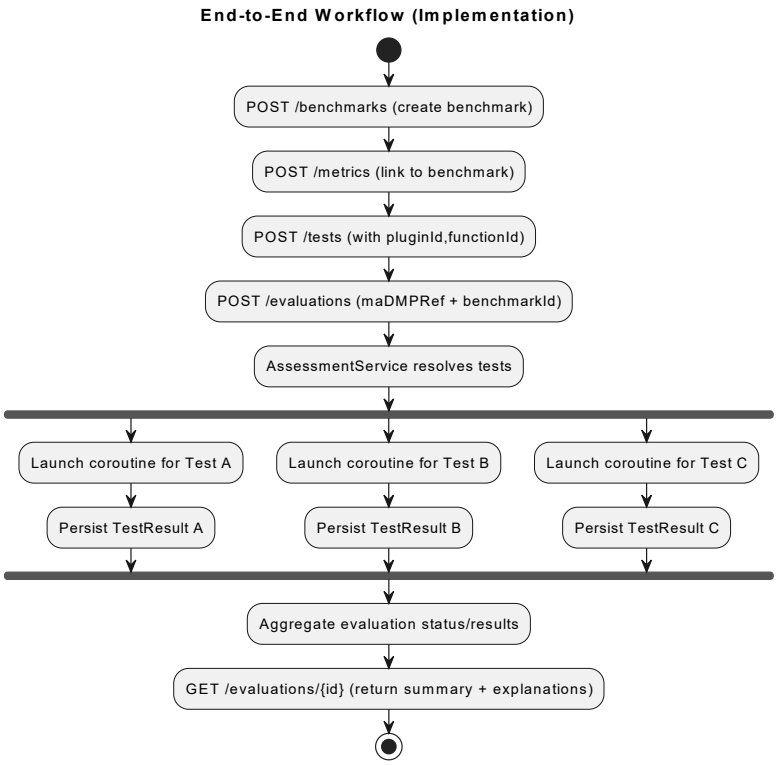

Figure 5.3: End-to-end workflow

5. The user initiates an evaluation by submitting a maDMP and benchmark ID to the `POST /evaluations` endpoint.

6. The `AssessmentService` resolves the relevant tests, launches their execution as coroutines, and stores incremental results in MongoDB.

7. The user retrieves results through `GET /evaluations/{id}`, obtaining grouped outcomes and detailed explanations for each test.

This workflow demonstrates how the architecture and implementation support automation, transparency, and extensibility in DMP evaluation.

5.8 Summary

This chapter has described the implementation of the DMP Evaluation Service, detailing the concrete technologies, components, and design decisions to implement the architecture introduced in Chapter 4. A plugin mechanism enables extensibility by allowing new evaluation logic to be added without changes to the core system, while MongoDB ensures flexible persistence of benchmarks, metrics, tests, evaluations, and test results.
Each functional requirement (FR-01 to FR-09) was mapped to concrete API endpoints, services, and persistence structures, demonstrating full traceability from requirements to implementation. We presented the plugin mechanism and parallel execution model,

the persistence layer, and the handling of error conditions to ensure robustness. An end-to-end example workflow illustrated how the system supports stakeholders in registering evaluation components, executing tests, and retrieving transparent and interoperable results.

These implementation details demostrate that our DMP Evaluation Service not only fulfills the defined functional requirements but also addresses the identified quality goals of suitability, maintainability, and interoperability. The next chapter evaluates the system, analyzing its operation, reliability, and effectiveness in supporting automated and extensible DMP assessments.

Chapter 6

Evaluation

6.1 Introduction

This chapter evaluates the DMP Evaluation Service to determine whether it fulfills the functional requirements (FR-01. . . FR-09) and addresses the quality goals defined in Chapter 3. The evaluation focuses on: (i) functional validation of the API and the execution workflow, (ii) validation of key quality attributes (functional suitability, maintainability, operability, compatibility).

6.2 Methodology

The methodology defines how the evaluation of the DMP Evaluation Service was conducted, ensuring that both functional and non-functional requirements are systematically verified. By structuring the evaluation along clear dimensions—scope, test data, procedure, and environment—the results become reproducible and aligned with the quality goals established in Chapter 3.

Scope

The evaluation covers the complete end-to-end workflow of the service, starting with the registration of evaluation components (`benchmarks`, `metrics`, and `tests`) and continuing through the execution of evaluations on machine-actionable DMPs (maDMPs). Beyond verifying workflow correctness, the evaluation also inspects API behavior (status codes, response payloads, error handling), persistence consistency in MongoDB, and the runtime execution model based on Kotlin coroutines. This dual focus ensures that the system is validated as both a functional service and a robust, scalable software component.
Test data: We used representative maDMP inputs and sample benchmarks/metrics/tests derived from the assessment framework [1]. Synthetic cases were added to exercise error paths (e.g., missing plugins, unreachable external APIs).

Procedure: Each functional requirement was validated via HTTP requests (cURL/-Postman). Quality goals were assessed through targeted experiments (e.g., adding a

new plugin).

6.3 Functional Validation

This section provides evidence that each functional requirement is satisfied by the implementation.

FR Evidence Matrix

FR	What was validated	Evidence (request/response excerpt)	Result
FR-01	Create benchmark	`POST /benchmarks` returns 201 with `_id`; stored in `benchmarks`.	**Pass**
FR-02	Create metric and link to benchmark	`POST /metrics` includes `benchmarkId`; query shows relation.	**Pass**
FR-03	Register test and link to metric	`POST /tests` stores mapping fields; retrieval consistent.	**Pass**
FR-04	List entities	`GET /benchmarks|metrics|tests` return expected sets + filters.	**Pass**
FR-05	Add Test Implementation Source Code	`GET /plugins` lists plugin/function identifiers.	**Pass**
FR-06	Link/Update test→implementation	`PUT /tests/{id}` add the plugin and the function id to a test.	**Pass**
FR-07	Start evaluation	`POST /evaluations` accepts maDMP + selection; creates evaluation.	**Pass**
FR-08	Execute in parallel	Concurrent test execution visible aggregated results.	**Pass**
FR-09	Return results	`GET /evaluations/{id}` returns evaluation results.	**Pass**

Table 6.1: Functional requirements – validation summary

Representative Requests/Responses

Before presenting the individual examples, this subsection introduces how the request/response evidence should be interpreted within the context of the evaluation. Each API interaction shown below corresponds directly to one or more functional requirements (FR-01. . . FR-09) and demonstrates how the implemented DMP Evaluation Service behaves under realistic usage scenarios.
The requests illustrate how clients interact with the REST API—using JSON bodies or multipart form-data—while the responses confirm that the service validates inputs, persists metadata, executes tests, or returns evaluation results in compliance with the system design. Together, these examples provide concrete operational evidence that

the service supports the full evaluation workflow, from registering benchmarks and metrics to executing parallel test evaluations and handling exceptional conditions.

Create a benchmark (FR-01).

This example demonstrates the functionality required by **FR-01**, which states that the system shall allow users to register a new benchmark with descriptive metadata. The request below submits a benchmark definition containing title, description, keywords, and creators. The response confirms successful creation by returning `200` along with the assigned `benchmarkId`, showing that the benchmark was validated and persisted correctly in the `benchmarks` collection.

```
{
    "title": "Comprehensive RDM Activities Coverage",
    "description": "Ensure that the DMP addresses all key components of research data management, including lifecycle coverage, roles and responsibilities, budgeting, description of outputs, provenance, application of best practices, reproducibility, and alignment with relevant policies.",
    "version": "0.0.1",
    "keyword": "DMP, Benchmark, maDMP",
    "theme": "DMP Evaluation COVERAGE",
    "status": "Active",
    "creator": [
        "https://orcid.org/0000-0002-0893-8509",
        "https://orcid.org/0009-0002-4848-5089"
    ],
    "hasAssociatedMetric": []
}
```

Listing 6.1: Request: POST /benchmarks

Response

```
    {
        "benchmarkId": "686ce321dd621c3ebe98d5b8",
        "title": "Comprehensive RDM Activities Coverage",
        "description": "Assesses the extent to which DMPs cover full research data management (RDM) lifecycle, roles, costs, data outputs, and best practices. Helps ensure alignment with funder and institutional policies.",
        "version": "0.0.1",
        "hasAssociatedMetric": [],
        "algorithms": [],
        "keyword": "DMP, Benchmark, maDMP, RDM, policy, lifecycle",
        "abbreviation": "RDM-COVERAGE",
        "landingPage": null,
        "theme": "DMP Evaluation - Coverage",
        "status": "Active",
        "creator": [
            "https://orcid.org/0000-0002-0893-8509",
```

```
            "https://orcid.org/0009-0002-4848-5089"
        ]
    }
```

Listing 6.2: Response: 200 Created

Create metric and link to benchmark (FR-02).

This operation validates **FR-02**, which requires the system to support registering a metric and associating it with an existing benchmark. The following request shows a `POST /metrics` call that includes the `hasBenchmark` field linking the metric to the benchmark created earlier. The response confirms that the metric was stored successfully and that its relationship to the benchmark was recorded as defined in the data model.

```
{
        "title": "Defined Roles for Contributors in RDM",
        "description": "Validates the presence and accuracy of
    contributor roles in the  DMP, supporting RDM responsibility
    transparency.",
        "version": "0.0.1",
        "keyword": "RDM, contributor, roles, responsibility, DMP",
        "abbreviation": "RDM-Roles-Defined",
        "landingPage": null,
        "theme": "Roles in RDM",
        "status": "Active",
        "isApplicableFor": "maDMP",
        "supportedBy": "OSTrails",
        "hasBenchmark": ["686ce321dd621c3ebe98d5b8"]
}
```

Listing 6.3: Request: POST /metrics

Response

```
{
    "id": "686ce730dd621c3ebe98d5b9",
    "title": "Defined Roles for Contributors in RDM",
    "description": "Validates the presence and accuracy of contributor
     roles in the DMP, supporting RDM responsibility transparency.",
    "version": "0.0.1",
    "keyword": "RDM, contributor, roles, responsibility, DMP",
    "abbreviation": "RDM-Roles-Defined",
    "landingPage": null,
    "theme": "Roles in RDM",
    "status": "Active",
    "isApplicableFor": "maDMP",
    "supportedBy": "OSTrails",
    "hasBenchmark": [
        "686ce321dd621c3ebe98d5b8"
    ]
```

```
}
```

Listing 6.4: Response: 200 Created

Register test and link to metric (FR-03).

The functionality shown here satisfies **FR-03**, which mandates the registration of a new test with its conceptual metadata and its association with a metric. The request includes metadata such as title, description, version, and the `metricImplemented` reference. The response verifies that the test has been correctly persisted and linked, making it available for later execution.

```
{
  "title": "Roles in RDM Defined",
  "description": "Verifies that contributor roles related to data
    management are clearly defined in the DMP.",
  "license": "MIT",
  "version": "0.0.1",
  "endpointDescription": "Verifies that contributor roles related to
    data management are clearly defined in the DMP.",
  "keyword": "roles, contributor, RDM, DMP",
  "abbreviation": "RDM-Roles",
  "repository": "https://github.com/OSTrails/DMP-Evaluation-Service",
  "type": "test",
  "theme": "DMP Coverage",
  "versionNotes": "0.0.1",
  "status": "Active",
  "isApplicableFor": "maDMP",
  "supportedBy": null,
  "metricImplemented": "686ce730dd621c3ebe98d5b9"
}
```

Listing 6.5: Request: POST /tests

Response

```
    "id": "686d1088dd621c3ebe98d5ba",
    "title": "Roles in RDM Defined",
    "description": "Verifies that contributor roles related to data
    management are clearly defined in the DMP.",
    "license": "MIT",
    "version": "0.0.1",
    "endpointURL": "http://localhost:8080/tests/686
    d1088dd621c3ebe98d5ba",
    "endpointDescription": "Verifies that contributor roles related to
     data management are clearly defined in the DMP.",
    "keyword": "roles, contributor, RDM, DMP",
    "abbreviation": "RDM-Roles",
    "repository": "https://github.com/OSTrails/DMP-Evaluation-Service
    ",
    "type": "test",
```

```
    "theme": "DMP Coverage",
    "versionNotes": "0.0.1",
    "status": "Active",
    "isApplicableFor": "maDMP",
    "supportedBy": null,
    "metricImplemented": "686ce730dd621c3ebe98d5b9"
```

Listing 6.6: Response: 200 Created

List entities (FR-04).

FR-04 requires the system to provide access to declared benchmarks, metrics, and tests. The following `GET` requests demonstrate that the API can retrieve all registered entities, reflecting the current state of the database. The responses show the expected collections and confirm that entity listing, filtering, and retrieval behave consistently with the service specification.

```
GET http://benchmarks/list
```

Listing 6.7: Request: GET /benchmarks/list

Response

```
[
     {
        "benchmarkId": "686ce321dd621c3ebe98d5b8",
        "title": "Comprehensive RDM Activities Coverage",
        "description": "Assesses the extent to which DMPs cover full
    research data management (RDM) lifecycle, roles, costs, data
    outputs, and best practices. Helps ensure alignment with funder
    and institutional policies.",
        "version": "0.0.1",
        "hasAssociatedMetric": [
            "686ce730dd621c3ebe98d5b9",
            "686d155fdd621c3ebe98d5bb",
            "686d1ae4dd621c3ebe98d5be"
        ],
        "algorithms": [],
        "keyword": "DMP, Benchmark, maDMP, RDM, policy, lifecycle",
        "abbreviation": "RDM-COVERAGE",
        "landingPage": null,
        "theme": "DMP Evaluation - Coverage",
        "status": "Active",
        "creator": [
            "https://orcid.org/0000-0002-0893-8509",
            "https://orcid.org/0009-0002-4848-5089"
        ]
    }

]
```

Listing 6.8: Response: 200 Created

```
GET http: //metrics/list
```

Listing 6.9: Request: GET /metrics/list

Response

```
[
    {
        "id": "683d6f453bf7d46ee332102e",
        "title": "Roles and Responsibilities in RDM",
        "description": "Checks whether data management
    responsibilities are defined and assigned through contributor
    roles in the DMP.",
        "version": "0.0.1",
        "testAssociated": null,
        "keyword": "roles, contributor, responsibility, DMP",
        "abbreviation": "RDM-Roles",
        "landingPage": null,
        "theme": "RDM Governance",
        "status": "Planned",
        "isApplicableFor": "maDMP",
        "supportedBy": "DMP Evaluation Service",
        "hasBenchmark": [
            "http://localhost:8080/benchmarks/6839c19dfde6a54b82304004
    "
        ]
    }
]
```

Listing 6.10: Response: 200 Created

```
GET http:/tests/info
```

Listing 6.11: Request: GET /tests

Response

```
[
        {
        "id": "686d1088dd621c3ebe98d5ba",
        "title": "Roles in RDM Defined",
        "description": "Verifies that contributor roles related to
    data management are clearly defined in the DMP.",
        "license": "MIT",
        "version": "0.0.1",
        "endpointURL": "http://localhost:8080/tests/686
    d1088dd621c3ebe98d5ba",
        "endpointDescription": "Verifies that contributor roles
    related to data management are clearly defined in the DMP.",
        "keyword": "roles, contributor, RDM, DMP",
        "abbreviation": "RDM-Roles",
```

```
        "repository": "https://github.com/OSTrails/DMP-Evaluation-
    Service",
        "type": "test",
        "theme": "DMP Coverage",
        "versionNotes": "0.0.1",
        "status": "Active",
        "isApplicableFor": "maDMP",
        "supportedBy": null,
    }

]
```

Listing 6.12: Response: 200 Created

Add Source Code Implementing a Test (FR-05).

This part validates **FR-05**, which requires the system to support the registration of executable test implementations. Although the code is stored externally in plugins, the service exposes the available plugin identifiers via `GET /plugins`. Displaying the list of plugins confirms that the service successfully discovers and exposes implementations that may later be linked to individual tests.

```
GET http://plugins
```

Listing 6.13: Request: GET /tests

Response

```
[
    {
        "pluginId": "DCSCompletenessEvaluator",
        "description": "Evaluator to perform DCScompleteness tests",
        "functions": [
            "evaluateStructure",
            "evaluateFormats",
            "costEntityPresent",
            "costEntityValuesPresent",
            "contributorValuesPresent",
            "datasetEntityValuesPresent"
        ]
    },
    {
        "pluginId": "ComplianceEvaluator",
        "description": "Evaluator to perform Compliance tests",
        "functions": [
            "evaluateCoherentLicense",
            "checkFormatFile"
        ]
    },
    {
        "pluginId": "FAIR_Champion",
```

```
        "description": "Evaluator to perform External calls for tests
    ",
        "functions": [
            "evaluateStructure",
            "evaluateMetadata",
            "evaluateLicense"
        ]
    },
    {
        "pluginId": "FeasibilityEvaluator",
        "description": "Evaluator to perform Feasibility tests",
        "functions": [
            "evaluateCoherentLicense"
        ]
    },
    {
        "pluginId": "QualityOfActionsEvaluator",
        "description": "Evaluator to perform FAIR tests",
        "functions": [
            "evaluateOpenAccess",
            "dmpIdValid"
        ]
    }
]
```

Listing 6.14: Response: 200 listed

Link Test Implementation to Metadata Record (FR-06).

FR-06 ensures that each test can be connected to a specific executable implementation. The example below shows a `POST /tests/{id}` request that updates a test by attaching a `pluginId` and `functionId`. The system validates the identifiers against the plugin registry and updates the stored test metadata accordingly, enabling correct resolution during evaluation.

```
POST http://tests/686d1088dd621c3ebe98d5ba
```

Listing 6.15: Request: POST /tests/testId

Response

```
{
    "id": "686d1088dd621c3ebe98d5ba",
    "title": "Roles in RDM Defined",
    "description": "Verifies that contributor roles related to data
    management are clearly defined in the DMP.",
    "license": "MIT",
    "version": "0.0.1",
    "endpointURL": "http://localhost:8080/tests/686
    d1088dd621c3ebe98d5ba",
```

```
    "endpointDescription": "Verifies that contributor roles related to
     data management are clearly defined in the DMP.",
    "keyword": "roles, contributor, RDM, DMP",
    "abbreviation": "RDM-Roles",
    "repository": "https://github.com/OSTrails/DMP-Evaluation-Service
    ",
    "type": "test",
    "theme": "DMP Coverage",
    "versionNotes": "0.0.1",
    "status": "Active",
    "isApplicableFor": "maDMP",
    "supportedBy": null,
    "metricImplemented": "686ce730dd621c3ebe98d5b9",
    "evaluator": "DCSCompletenessEvaluator",
    "functionEvaluator": "contributorValuesPresent"
}
```

Listing 6.16: Response: 200 Updated

Select maDMP and Tests for Execution (FR-07).

This example demonstrates compliance with **FR-07**, which requires the system to accept a request that initiates an evaluation. The request includes a reference to the maDMP and the benchmark that should guide the execution. The successful creation of an evaluation object indicates that the system has validated the input and prepared the execution workflow.

To execute a single test, the client sends a multipart `POST` request to the endpoint `/assess/test`. The request includes (i) the maDMP file, (ii) the identifier of the test to execute.

```
POST /assess/test
Content-Type: multipart/form-data; boundary=----form-boundary

------form-boundary
Content-Disposition: form-data; name="maDMP"; filename="madmp.json"
Content-Type: application/json

{
  "dmp": {
    "title": "Sample Research Project DMP",
    "contact": { "mbox": "pi@example.org" },
    "dataset": [
      { "title": "Experimental Results Dataset", "pid": "doi:10.1234/
    foo.bar" }
    ]
  }
}
------form-boundary
Content-Disposition: form-data; name="test"
"686d1088dd621c3ebe98d5ba"
```

Listing 6.17: Request: POST /assess/test (multipart/form-data)

Response

```
{
    "evaluationId": "0d7cc1bb-8e9e-4248-93a1-1d90da6a2e5f",
    "title": "Roles in RDM Defined",
    "result": "PASS",
    "details": "Verifies that contributor roles related to data
    management are clearly defined in the DMP.",
    "timestamp": "2025-11-20T15:54:18.191556300Z",
    "reportId": "691f39aa45f0bc725eeb77c5",
    "log": "contributors: [{\"contributor_id\":{\"identifier
    \":\"0000-0002-4929-7875\",\"type\":\"orc\"},\"name\":\"Tomasz
    Miksa\",\"role\":[\"Supervisor\"]}]\nContributor fields are
    present in the maDMP",
    "affectedElements": "dpm.contributor",
    "completion": 100,
    "generated": "io.github.ostrails.dmpevaluatorservice.evaluators.
    completenessEvaluator.DCSCompletenessEvaluator::
    contributorValuesPresent",
    "outputFromTest": "686d1088dd621c3ebe98d5ba"
}
```

Listing 6.18: Response: POST /assess/test (multipart/form-data)

Execute Test Implementations in Parallel (FR-08).

To demonstrate parallel execution (FR-08), a benchmark with multiple associated metrics and tests was previously created in Sections FR-01 to FR-03. The following snippet shows how the service executes all related tests.

Before running the benchmark evaluation, the necessary evaluation components were already registered in the system. After creating the initial benchmark (Section 6.13), additional metrics were added and linked to it, and each metric was associated with one or more tests. These tests were also linked to concrete plugin-based implementations using their `pluginId` and `functionId`. As a result, the benchmark now defines a complete evaluation configuration composed of multiple metrics and several executable tests.

The following example demonstrates how the DMP Evaluation Service executes all tests associated with this benchmark in parallel using Kotlin coroutines. By submitting a maDMP file together with the benchmark identifier, the service resolves all linked tests, schedules each of them as an independent coroutine, and returns the aggregated evaluation results once the execution has completed.

```
POST /assess/benchmark
Content-Type: multipart/form-data; boundary=----Boundary


Content-Disposition: form-data; name="maDMP"; filename="example-madmp.
    json"
Content-Type: application/json

```

```
{ ... contents of the maDMP JSON document ... }
Content-Disposition: form-data; name="benchmark"
"686ce321dd621c3ebe98d5b8"
```

Listing 6.19: Request: POST /assess/benchmark

Response

The benchmark evaluation returns a list of `Evaluation` objects, one per executed test. Each entry includes status information (`PASS`, `FAIL`, or `INDETERMINATE`), the affected maDMP elements, detailed logs from the evaluator, and a `generated` field indicating the exact plugin function executed. The following response corresponds to the benchmark created in the previous steps.


```
[
  {
    "evaluationId": "7accb7e2-a6dc-45f9-863a-454a7470e325",
    "title": "Roles in RDM Defined",
    "result": "PASS",
    "details": "Verifies that contributor roles related to data
    management are clearly defined in the DMP.",
    "timestamp": "2025-11-24T13:43:14.568563100Z",
    "reportId": "692460f27dab71453683b5b7",
    "log": "contributors: [{\"contributor_id\":{\"identifier
    \":\"0000-0002-4929-7875\",\"type\":\"orc\"},\"name\":\"Tomasz
    Miksa\",\"role\":[\"Supervisor\"]}]\nContributor fields are
    present in the maDMP",
    "affectedElements": "dpm.contributor",
    "completion": 100,
    "generated": "io.github.ostrails.dmpevaluatorservice.evaluators.
    completenessEvaluator.DCSCompletenessEvaluator::
    contributorValuesPresent",
    "outputFromTest": "686d1088dd621c3ebe98d5ba"
  },
  {
    "evaluationId": "b5596774-c1ef-476a-8ada-31f2bb329320",
    "title": "Dataset Entity Completeness",
    "result": "PASS",
    "details": "Checks that dataset entities in the maDMP contain all
    required fields, including distribution, identifiers, and personal
     data flags.",
    "timestamp": "2025-11-24T13:43:14.569092600Z",
    "reportId": "692460f27dab71453683b5b7",
    "log": "Dataset[0] distribution title: ... personal_data: no,
    sensitive_data: no\nDataset[1] distribution title: ...
    personal_data: no, sensitive_data: no",
    "affectedElements": "dpm.dataset",
    "completion": 100,
    "generated": "io.github.ostrails.dmpevaluatorservice.evaluators.
    completenessEvaluator.DCSCompletenessEvaluator::
    datasetEntityValuesPresent",
    "outputFromTest": "686d1761dd621c3ebe98d5bd"
```

```
  },
  {
    "evaluationId": "e732d39b-85b0-4cba-82eb-d0afd5566d82",
    "title": "Cost Entity Completeness",
    "result": "PASS",
    "details": "Checks that cost fields such as title, value,
    description, and currency_code are filled in.",
    "timestamp": "2025-11-24T13:43:14.569092600Z",
    "reportId": "692460f27dab71453683b5b7",
    "log": "Cost entry - title: DAMAP Tutorial, description; Tutorial
    to learn DAMAP, value: 4500, currency: EUR",
    "affectedElements": "dpm.cost.description",
    "completion": 100,
    "generated": "io.github.ostrails.dmpevaluatorservice.evaluators.
    completenessEvaluator.DCSCompletenessEvaluator::costEntityPresent
    ",
    "outputFromTest": "686d1f52dd621c3ebe98d5bf"
  },
  {
    "evaluationId": "53651e1f-6442-43cd-a30e-0963ad3913ab",
    "title": "Presence of Cost Entity",
    "result": "PASS",
    "details": "Ensures the presence of the 'cost' field in the maDMP
    .",
    "timestamp": "2025-11-24T13:43:14.569092600Z",
    "reportId": "692460f27dab71453683b5b7",
    "log": "Cost fields are present in the maDMP",
    "affectedElements": "dpm.cost",
    "completion": 100,
    "generated": "io.github.ostrails.dmpevaluatorservice.evaluators.
    completenessEvaluator.DCSCompletenessEvaluator::costEntityPresent
    ",
    "outputFromTest": "686d201ddd621c3ebe98d5c0"
  }
]
```


Listing 6.20: Response: 200 OK – Benchmark Evaluation

Return Failure Explanations (FR-09).

To validate the robustness of the DMP Evaluation Service, several failure scenarios were tested in addition to successful executions. These tests demonstrate how the service behaves when incorrect parameters are provided, when plugin mappings are missing, or when test logic raises unexpected errors. The goal is to confirm that the system returns clear failure explanations, does not crash during evaluation, and continues processing remaining tests where possible, fulfilling the transparency and operability quality goals.

FAILURE SCENARIO 1 — Invalid Benchmark ID (404 Not Found)

```
POST http://localhost:8080/assess/benchmark

{
  "benchmark": "686ce321dd421c4ebe98s456",
  "maDMP": "examplemaDMP.json"
}
```

Listing 6.21: Request: POST /evaluations with invalid benchmark ID

Response

```
{
    "code": "NOT_FOUND",
    "message": "Was not possible to generate the evaluation due io.
    github.ostrails.dmpevaluatorservice.exceptionHandler.
    ResourceNotFoundException: There is no benchmark with the ID 686
    ce321dd421c4ebe98s456",
    "timestamp": "2025-11-24T15:27:55.929451200Z",
    "path": "/assess/benchmark"
}
```

Listing 6.22: Response: 404 Benchmark not found

FAILURE SCENARIO 2 — Missing plugin/function mapping (Test becomes INDETERMINATE)

```
{
  "maDMP": "examplemaDMP.json",
  "benchmarkId": "686ce321dd621c3ebe98d5b8"
}
```

Listing 6.23: Request: POST /evaluations where a test has no implementation

(

Response)

```
{
  "evaluationId": "e19c2ef5-330c-4ab1-a9dc-9179d508f334",
  "title": "Roles in RDM Defined",
  "result": "INDETERMINATE",
  "details": "Test has no associated plugin or function implementation
    .",
  "affectedElements": null,
  "log": "PluginId=functionEvaluator not resolved",
  "completion": 0
}
```

Listing 6.24: Response: Test marked as INDETERMINATE due to missing plugin mapping

FAILURE SCENARIO 3 — Invalid maDMP (400 Bad Request)

```
POST http://localhost:8080/evaluations/benchmark
Content-Type: multipart/form-data

maDMP = "maDMP.csv"
benchmark = "686ce321dd621c3ebe98d5b8"
```

Listing 6.25: Request: POST /benchmark with malformed maDMP

Response

```
{
  "error": "Invalid maDMP format",
  "message": "Was not possible to generate the evaluation due
    Unexpected JSON token at offset
  "status": 400
}
```

Listing 6.26: Response: 400 Invalid maDMP JSON

These examples collectively confirm that the DMP Evaluation Service correctly processes evaluation components, persists them, retrieves them, and executes benchmark-defined evaluations as specified in the functional requirements.

6.3.1 Quality Goals Validation

We evaluate the most relevant quality attributes from Table 3.4. Each subsection presents the claim associated with the quality goal, the procedure or evidence used to validate it, and the final outcome of the evaluation.

Functional Suitability

The system provides functions that meet the stated needs. **Evidence.** All functional requirements (FR-01 to FR-09) are satisfied according to the validation matrix in Table 6.1. The service correctly performs benchmark & metric registration, test registration, mapping of tests to plugin implementations, and the execution of evaluations with structured outputs. Evaluation results consistently include pass/fail statuses, detailed explanations, and execution metadata. **Outcome.** *Pass.*

Maintainability

New evaluation logic can be added without modifying the core system. **Procedure.** A new plugin function `quality.test/check_license` was implemented and exposed in the plugin registry. A new `Test` entity referencing the new `pluginId` and `functionId` was registered through the API. **Evidence.** No changes were required in controllers, services, or execution infrastructure. The newly added test executed successfully during a benchmark evaluation and produced valid output. **Outcome.** *Pass.*

Operability

The system is easy to operate and its API is simple for users and external platforms to consume. **Evidence.** All endpoints follow predictable REST conventions; error messages include machine-actionable codes and human-readable explanations. CRUD operations for benchmarks, metrics, and tests behave consistently, and multipart evaluation requests can be executed using common tools such as `curl` or Postman without additional configuration. **Outcome.** *Pass.*

Compatibility

The system interoperates correctly with external services and external standards. **Evidence.** Evaluation functions successfully interact with external APIs such as Unpaywall and FAIR evaluation services. The output produced by the service conforms to the FAIR Testing Resource (FTR) data model, enabling reuse in external tools. **Outcome.** *Pass.*

6.3.2 Alignment with the Assessment Framework

The DMP Evaluation Service has been designed to operationalize the Assessment Framework. This framework defines the core concepts (benchmarks, metrics, and tests) and their relationships as the basis for evaluating Data Management Plans (DMPs).

Conceptual Alignment

The service adopts these entities directly in its data model:

- **Benchmarks** represent high-level evaluation scenarios or goals.
- **Metrics** capture specific aspects of a DMP to be assessed and are grouped under benchmarks.
- **Tests** provide concrete and executable evaluation procedures for metrics.

This alignment ensures that evaluations performed by the service follow the same conceptual structure as defined by the Assessment Framework, enabling comparability across different evaluations and use cases.

FAIR Reference Data Model

In addition to conceptual alignment, the service integrates the *FAIR Reference Data Model* produced in [1]. Evaluation results are represented and returned in this model, ensuring that they are machine-actionable and interoperable with other tools. Each `TestResult` includes structured fields for outcomes, explanations, and metadata, which can be exported and reused in other contexts.

Extensibility and Interoperability

The plugin mechanism in the DMP Evaluation Service supports the extensibility principle of the Assessment Framework. New evaluation logic can be added without modifying the core service, while results remain represented in the common FAIR Reference Data Model. This design allows the service to integrate smoothly and continuously with DMP platforms and external FAIR assessment services, supporting interoperability across the data management ecosystem.

6.4 Limitations

Although the DMP Evaluation Service fulfills the functional and architectural goals defined for this thesis, some limitations remain that constrain its current use and applicability:

- There is no UI layer, evaluation is triggered programmatically. The prototype exposes its full functionality exclusively through a REST API. All interactions like registering benchmarks, uploading tests, or evaluating maDMPs must be performed programmatically via HTTP requests or tools like cURL/Postman. This limits accessibility for non-technical users and prevents integration into common DMP workflows without an intermediate client interface. A dedicated UI or integration into existing RDM platforms would significantly increase usability.

  However, this limitation can be addressed once the service is connected with DMP platforms, where the platform itself provides the user interface and hides the API complexity from end-users.

- Dependence on external APIs may introduce instability and unpredictable latency. Some tests rely on external services (e.g., Unpaywall, FAIR assessment endpoints) to validate metadata within the maDMP. These external dependencies introduce variability because:

  - API availability cannot be guaranteed
  - API rate limits or schema changes can break tests

- Security aspects, especially authentication and authorization, are out of scope. The prototype does not implement mechanisms for user authentication (authN) or role-based authorization (authZ). This means all endpoints are publicly accessible, no access control is enforced on creating or modifying evaluation components and no user-specific isolation of evaluation results exists.

6.5 Summary

The evaluation shows that all functional requirements (FR-01...FR-09) are satisfied and the targeted quality goals are met in practice. Coroutine-based parallel execu-

tion provides scalable processing, robustness test worked when plugins are missing or external APIs fail. Identified limitations (no UI, limited scale tests, external dependencies) inform future work.

Chapter 7

Conclusion and Future work

This chapter presents the central contributions of this work including its limitations, summarizes the main results obtained by answering the research questions proposed in Chapter 1, and discusses future work directions.

7.1 Contributions

In this thesis, we propose a DMP Evaluation Service that extends and addresses limitations of previous work [8]. We improved the design using the following approach. First, we collected the requirements of stakeholders from discussions with the consortium of the OSTrails project, and also based on previous work in maDMPs. These requirements cover functional and non-functional characteristics for the design of the DMP Evaluation Service. Based on the requirements, we designed the architecture of the system, meaning we defined the main building blocks, the interactions between components, and the actors involved. Then, we implemented the system in Kotlin using the Spring Boot framework and defined the primary endpoints for executing the service functionalities. Finally, we evaluated the DMP Evaluation Service against the requirements defined and provided practical examples of the responses obtained.

The central contributions of this work can be summarized as follows:

- Design and development of a configurable evaluation service for maDMPs, implemented in Kotlin using the Spring Boot framework.
- Integration of a benchmark-metric-test model from OSTrails project that enables transparent, reusable, and modular assessment configurations.
- Implementation of an interoperable API for evaluating maDMPs and returning structured results aligned with OSTrails' assessment interoperable framework specifications.

7.2 Review of Research Questions

To wrap up the work developed in this thesis, we revisit the research questions defined in section 1.5.

1. In what way the architecture of the DMP service needs to be revised to better reflect the real world requirements of production ready systems?

   The work presented in this thesis demonstrates that the original architecture designed in Lukas Arnold's [8] thesis required modifications to satisfy requirements such as modularity, extensibility, interoperability, and operational robustness.

   First, the service architecture was reorganized into a clear layered structure that separates REST controllers, business logic services, plugin-based execution, coroutine-based parallelism, and database persistence. This redesign eliminates the tightly coupled structure of the original prototype and enables the system to evolve without breaking existing components. In particular, the new modularization avoids the previous situation in which loading or processing a maDMP from a new source required adding custom code directly into the service logic. By isolating data ingestion, execution, and orchestration concerns, the system now provides a stable, extensible integration boundary where new maDMP sources or formats can be supported without modifying core components.

   Second, the modification of the Plugin mechanism, organized in the original design with DMP categories only, allows evaluation logic to be added, replaced, or removed without modifying the core service. This is a critical requirement for real-world deployments where evaluation methods evolve, expand, and must be maintained independently of the backend service.

   Finally, API design, error handling, logging, and the data access layer were all updated to reflect operational needs typical for production systems—improved stability, better diagnostics, meaningful error messages, and resilience against external API failures.

   Overall, the revised architecture is more modular, maintainable, scalable, and operationally realistic, making it appropriate for integration into real research data management infrastructures.

2. In what way the data model of the DMP service needs to be revised to align with FTR and to support the architectural changes?

   The evaluation conducted in this thesis shows that the original data model was too limited to support interoperability and traceability requirements expected from modern evaluation systems. To address this, the thesis revises the data model to align with the FAIR Testing Resource (FTR) vocabulary, ensuring compatibility with the broader FAIR assessment ecosystem.

   First, the entities Benchmark, Metric, Test, Evaluation, and TestResult were adapted to better mirror the conceptual definitions in the Assessment Frame-

work [1] and the FTR specification. This includes the introduction of explicit identifiers, typed relationships, and structured metadata fields.

Second, new fields were added to the Test entity to support runtime binding: pluginId functionI.

These fields link conceptual test definitions to their executable implementations, enabling automated test resolution and execution — a feature not present in the earlier data model.

Third, the adoption of MongoDB aligns naturally with FTR requirements for flexible result schemas. This enables the system to store heterogeneous result payloads originating from different evaluators without sacrificing consistency.

In summary, the revised data model provides semantic alignment with the FTR vocabulary, supports linking of tests to implementations, and ensures that evaluation results are interoperable, machine-actionable, and reusable across tools.

3. How to integrate the Assessment Framework with the DMP Evaluation Service in order to provide standardized evaluation results ?

   The findings of this thesis demonstrate that seamless integration of the Assessment Framework is achieved by embedding its concepts and relationships directly into the core architecture and execution workflow of the DMP Evaluation Service.

   First, the entities defined by the Assessment Framework—Benchmarks, Metrics, and Tests—are implemented as first-class objects in the system and follow the hierarchy prescribed in the framework. This guarantees that every evaluation is grounded in a structured, shared conceptual model.

   Second, the plugin mechanism ensures that the execution of Tests is both standardized and flexible. Because each Test references a pluginId and a functionId, the system can execute heterogeneous evaluation functions while still representing the results in a consistent way defined by the Assessment Framework.

   Third, evaluation output is serialized into a machine-actionable structure that is aligned with the FAIR Reference Data Model and compatible with the FAIR Testing Resource (FTR). This ensures that results produced by the service can be consumed by other FAIR assessment tools, integrated into RDM workflows, and reused for meta-assessment or reporting.

   Fourth, the system supports external validation through optional integration with external APIs such as FAIR assessment services. This extends the Assessment Framework beyond syntactic or structural validations and allows semantic or evidence-based verification, fully in line with the framework's spirit of dynamic and context-aware evaluation.

Through these mechanisms, the DMP Evaluation Service operationalizes the Assessment Framework in a way that maintains conceptual fidelity while enabling interoperability across tools, platforms, and assessment pipelines in the research data management ecosystem.

7.3 Future Work

Several directions can be explored to extend the present work. First, to facilitate stakeholder interaction with the DMP Evaluation service, a possible extension is to provide a UI layer that simplifies the process for end users and enables functionalities depending on the actor's role. Second, assessing the service scalability in a production-ready environment also remains an open direction for future evaluation. Third, in this work, we did not consider security aspects in a production deployment and this is an area that could be explored to improve automatic assessment.

Appendix A

Appendix A: Resources

A.1 Prototype Source Code

The complete implementation of the DMP Evaluation Service is available at:

- **Repository:** `https://github.com/your-org/dmp-evaluator-service`
- **Version evaluated in this thesis:** `v1.0.0-thesis`

The repository is organised as follows:

- `src/main/kotlin/` – Core application modules (controllers, services, plugin registry, evaluation workflow).
- `src/main/resources/` – Configuration files and example data.
- `plugins/` – Example plugin implementations used during evaluation.
- `docker-compose.yml` – Docker configuration for MongoDB.

A.2 Running the Prototype

This section describes how to run the DMP Evaluation Service locally for testing or reproduction of the evaluation results.

1. Clone the Repository

```
git clone https://github.com/your-org/dmp-evaluator-service.git
cd dmp-evaluator-service
```

2. Start MongoDB using Docker Compose

Ensure that Docker is installed and running. Then start the MongoDB backend:

```
docker-compose up -d
```

This launches a MongoDB instance configured for the application.

3. Build the Project

The service is built using Maven:

```
mvn clean install
```

This command compiles the Kotlin/Spring Boot sources and runs all automated tests.

4. Run the Application

Start the service using the Spring Boot Maven plugin:

```
mvn spring-boot:run
```

The application will start on:

- `http://localhost:8080`

5. Access API Documentation (Optional)

Swagger UI is automatically generated and can be accessed at:

- `http://localhost:8080/swagger-ui.html`

This interface provides a complete overview of available endpoints, request schemas, and example executions.

A.3 Additional Resources

A.3.1 Example maDMP Inputs

The following directory contains the maDMP files used during system evaluation:

- `DMP-Evaluation-Service-master-thesis /examplees_maDMP`

A.3.2 Benchmark, Metric, and Test Definitions

The JSON metadata used during the evaluation in Chapter 6 is included in:

- `resources/examples/definitions/`

A.3.3 Plugin Implementations

Example evaluator implementations used during testing are located in:

- dmpevaluatorservice/evaluators/

Bibliography

[1] *OSTrails/FAIR_testing_resource_vocabulary*. original-date: 2024-03-06T10:06:08Z. Aug. 2025. URL: https://github.com/OSTrails/FAIR_testing_resource_vocabulary (visited on 08/25/2025).

[2] Ngoc-Minh Pham, Heather Moulaison-Sandy, Bradley Wade Bishop, and Hannah Gunderman. "Data Management Plans: Implications for Automated Analyses | Data Science Journal". en. In: (Jan. 2023). DOI: 10.5334/dsj-2023-002. URL: https://datascience.codata.org/articles/dsj-2023-002 (visited on 09/29/2025).

[3] Tomasz Miksa, Peter Neish, Paul Walk, and Andreas Rauber. "Defining requirements for machine-actionable data management plans". en-US. In: *iPRES 2018 Conference Proceedings*. Conference Name: iPRES 2018. 2018. URL: https://phaidra.univie.ac.at/o:923628 (visited on 11/26/2025).

[4] Tomasz Miksa, Simon Oblasser, and Andreas Rauber. "Automating Research Data Management Using Machine-Actionable Data Management Plans". en. In: *ACM Transactions on Management Information Systems* 13.2 (June 2022). Publisher: Association for Computing Machinery (ACM), pp. 1–22. ISSN: 2158-656X, 2158-6578. DOI: 10.1145/3490396. URL: https://dl.acm.org/doi/10.1145/3490396 (visited on 07/12/2025).

[5] Tomasz Miksa, Stephanie Simms, Daniel Mietchen, and Sarah Jones. "Ten principles for machine-actionable data management plans". en. In: *PLOS Computational Biology* 15.3 (Mar. 2019). Publisher: Public Library of Science, e1006750. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1006750. URL: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006750 (visited on 08/05/2025).

[6] Tomasz Miksa, Marek Suchánek, Jan Slifka, Vojtech Knaisl, Fajar J. Ekaputra, Filip Kovacevic, Annisa Maulida Ningtyas, Alaa El-Ebshihy, and Robert Pergl. "Towards a Toolbox for Automated Assessment of Machine-Actionable Data Management Plans". en-US. In: *Data Science Journal* 22.1 (Aug. 2023). ISSN: 1683-1470. DOI: 10.5334/dsj-2023-028. URL: https://datascience.codata.org/articles/10.5334/dsj-2023-028 (visited on 07/12/2025).

[7] Alberto Ballesteros-Rodríguez, Miguel-Ángel Sicilia, and Elena García-Barriocanal. "madmpy: A Python library for creating and validating Data Management Plans". In: *SoftwareX* 31 (Sept. 2025), p. 102215. ISSN: 2352-7110. DOI: 10.

1016/j.softx.2025.102215. URL: https://www.sciencedirect.com/science/article/pii/S2352711025001827 (visited on 08/26/2025).

[8] Lukas Arnhold. "Automated Quality Indicators for Machine-actionable Data Management Plans". en. Accepted: 2024-09-11T10:46:26Z Journal Abbreviation: Automatisierte Qualitätsindikatoren für Maschinell Verarbeitbare Datenmanagementpläne. Thesis. Technische Universität Wien, 2024. DOI: 10.34726/hss.2024.117145. URL: https://repositum.tuwien.at/handle/20.500.12708/200466 (visited on 02/07/2025).

[9] Gernot Starke and Peter Hruschka. *arc43-web*. en. URL: %5Curl%7Bhttps://arc42.org/%7D (visited on 08/25/2025).

[10] Simon Brown. *The C4 Model for Visualising Software Architecture*. en-US. URL: https://c4model.com/ (visited on 08/25/2025).

[11] JetBrains. *Coroutines | Kotlin*. en-US. URL: https://kotlinlang.org/docs/coroutines-overview.html (visited on 08/25/2025).