JOHANNES KEPLER
UNIVERSITY LINZ

Author
Mario Lanz, BSc

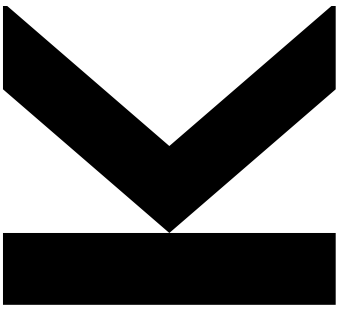
Submission
Institute of Business Informatics - Data & Knowledge Engineering

Thesis Supervisor
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Assistant Thesis Supervisor
Simon Staudinger, MSc

October 2026

Design and Implementation of an Extensible Knowledge Base for the Reliability Assessment of Predictive Analytic Results



Master's Thesis

to confer the academic degree of

Master of Science

in the Master's Program

Business Informatics

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

Abstract

Data mining plays an increasingly important role in helping organizations draw conclusions from large datasets and make informed decisions. However, a recurring problem is the evaluation of prediction reliability. In many practical settings, unreliable predictions can have serious implications, such as financial misjudgments or incorrect clinical decisions. This thesis addresses the issue of how to support a user to assess the reliability of new predictions by selecting and applying suitable approaches for assessing the reliability of analytic results from predictive models. Building on the reference process proposed by Staudinger et al. (2024), the system integrates assessment approaches, methodologies, and implementations into a unified semantic framework. A knowledge graph based on RDF and the PROV Ontology forms the core data layer and links entities such as assessment approaches, methodologies, and implementations across several abstraction levels, namely generic, method-specific, and instance-specific. The proposed knowledge base consists of three sections: the Guided Section, which helps users narrow down suitable approaches through structured filtering; the Browsing Section, which provides an overview of all stored elements; and the Results Section, where recorded assessment cases can be viewed and compared. The implemented knowledge base shows how the integration of assessment knowledge in a knowledge graph can support users in selecting suitable assessment approaches for evaluating the reliability of analytic results.

Kurzfassung

Data Minig spielt eine wichtige Rolle um Organisationen dabei zu unterstützen große Datenmengen zu analysieren und fundierte Entscheidungen darüber zu treffen. Ein wiederkehrendes Problem dabei besteht jedoch in der Bewertung der Zuverlässigkeit solcher Entscheidungen, da unzuverlässige Modelle zu schwerwiegenden Folgen wie finanziellen Fehlentscheidungen oder fehlerhaften klinischen Diagnosen führen können. Daher befasst sich diese Masterarbeit mit der Entwicklung einer erweiterbaren Wissensbasis um Anwenderinnen und Anwender bei der Auswahl von geeigneten Ansätzen für die Beurteilung der Zuverlässigkeit von Modellen zu unterstützen. Aufbauend auf dem von Staudinger et al., 2024 entwickeltem Referenzprozess integriert das System Bewertungsansätze, methodische Konzepte sowie Implementierungen in einen einheitlichen Rahmen. Ein auf RDF und PROV-O basierender Knowledge Graph bildet dabei die zentrale Datenbasis, die die unterschiedlichen Entitäten über mehrere Abstraktionslevels hinweg miteinander verbindet. Die Wissensbasis besteht aus drei Bereichen: die Guided Section, die durch strukturierte Vorgaben und Filter Optionen bei der Auswahl geeigneter Ansätze unterstützt; die Browsing Section, in der ein Überblick über alle gespeicherten Elemente geboten wird; und die Results Section, in der gespeicherte Bewertungsergebnisse angezeigt werden. Das implementierte System zeigt, wie die Integration von Bewertungswissen in einem Knowledge Graph Benutzerinnen und Benutzer bei der Auswahl von geeigneten Methoden unterstützt, um analytische Ergebnisse zu evaluieren und die Ergebnisse dieser Methoden in der Wissensbasis darzustellen. Zukünftige Arbeiten könnten sich darauf konzentrieren mehr Ansätze in die Wissensbasis aufzunehmen, die Empfehlungen der Wissensbasis zu verbessern sowie die Benutzerfreundlichkeit der Wissensbasis anzuheben.

Contents

Listings **vii**

1 Introduction **1**

2 State of the art **3**
2.1 Data Mining . . . 3
2.2 Knowledge Graph . . . 7

3 Related Work **10**
3.1 Reference Process . . . 10
3.2 Assessment Approaches . . . 14
3.3 Similar KB Designs . . . 15

4 Architecture **18**
4.1 System Overview and Component Responsibilities . . . 18
4.2 Description of REST Endpoints . . . 19
4.3 Final Technology Stack . . . 20

5 User Perspective **23**
5.1 Guided Section . . . 23
5.2 Browsing Section . . . 35
5.3 Result Section . . . 41

6 Implementation Perspective **45**
6.1 Knowledge Graph . . . 45
6.2 System Architecture and Implementation Structure . . . 52
6.3 Guided Section and Browsing Section . . . 56
6.4 Results Section . . . 73

7 Conclusion **75**

Bibliography **76**

List of Figures

2.1 CRISP-DM phases 4

3.1 Example activities for enabling assessment of reliability of classification results (Staudinger et al., 2024) 11

4.1 Abstract system overview and component responsibilities of the knowledge base. . . 19
4.2 Final technology stack of the knowledge base. 22

5.1 Guided Section - Stage 1 24
5.2 Guided Section - Stage 2 25
5.3 Guided Section - Example of expanded ADAN Methodology 25
5.4 Guided Section - Filter Sidebar 27
5.5 Guided Section - Result after Filtering 28
5.6 Guided Section - Stage 3 28
5.7 Guided Section - Record Assessment Modal 29
5.8 Add new Filter 30
5.9 Add new Filter Option 31
5.10 Add new Assessment Approach 31
5.11 Guided Section - Result after adding a new approach 31
5.12 Add new Methodology 32
5.13 Add new Methodology - Filter Options and Filter 33
5.14 Guided Section - Result after adding a new methodology 34
5.15 Browsing Section - Overview 36
5.16 Browsing Section - Methodologies 37
5.17 Browsing Section - Implementations and Generic Tools 37
5.18 Browsing Section - Results after filtering 38
5.19 Browsing Section - Results after selecting prediction methods 39
5.20 Browsing Section - Add Methodology: additional Dialog 40
5.21 Browsing Section - Add Implementation: additional Dialog 40
5.22 Results Section - Example assessment case retrieved from the Perturbation Tool . . 42
5.23 Results Section - Example assessment case created with the Perturbation Tool . . . 42
5.24 Results Section - Example assessment cases recorded for the ADAN package 43

6.1 System architecture of the knowledge base 53

6.2 Implementation Structure Backend 54
6.3 Implementation Structure Frontend 55

List of Tables

4.1 Controllers with respective HTTP operations 21

6.1 **BusinessUnderstandingEntity** from the reference process 46
6.2 **DeploymentEntity** from the reference process 46
6.3 Filter 46
6.4 Filter Options 47
6.5 Object property to connect Filter and Filter Options 47
6.6 Generic element: rprov:AssessmentApproach 47
6.7 method-specific element: rprov:Methodology 47
6.8 problem-specific element: rprov:Implementation 48
6.9 subclass element: rprov:Package 48
6.10 subclass element: rprov:Tool 48
6.11 subclass element: rprov:GenericTool 48
6.12 ObjectProperty to connect elements of all abstraction levels 48
6.13 ObjectProperty to connect prediction methods with assessment approaches 49
6.14 ObjectProperty rprov:requires 49
6.15 ObjectProperty rprov:recommends 49
6.16 ObjectProperty rprov:prohibits 49
6.17 Subclasses of *rprov:DeploymentEntity*: **rprov:Assessment** 50
6.18 Subclasses of *rprov:DeploymentEntity*: **rprov:AssessmentCase** 50
6.19 Specific Case Elements 51
6.20 Object Property to link Assessment Cases and Assessments 51
6.21 Object Property to link Assessments to Implementations 51

Listings

2.1 RDF triples . . . 6
6.1 Knowledge Graph example . . . 51
6.2 Example elements for presentation . . . 57
6.3 AssessmentApproachService - getAssessmentApproaches(...) . . . 58
6.4 AssessmentApproachQueryConfig - q_getAssessmentApproach(...) . . . 59
6.5 AssessmentAppraochDto - convertCsvToDtos(...) - Step 1 . . . 60
6.6 AssessmentAppraochDto - convertCsvToDtos(...) - Step 2 . . . 61
6.7 AssessmentAppraochDto - convertCsvToDtos(...) - Step 3 . . . 61
6.8 AssessmentAppraochService - cascadeFilterOptionsUpwards(...) . . . 62
6.9 Request Body for adding a new Implementation . . . 64
6.10 Wrapper for inserting a new implementation . . . 65
6.11 q_insertImplementation(...) - insert new Filters . . . 65
6.12 q_insertImplementation(...) - insert new Filter Options . . . 66
6.13 q_insertImplementation(...) - final INSERT DATA statement . . . 67
6.14 Extended select query - Filtering results . . . 68
6.15 Prediction Method element with Checkbox . . . 69
6.16 Selection Algorithm - onCheckboxChange(...) function . . . 70
6.17 Selection Algorithm - setSelectFlags(...) function . . . 72
6.18 Example elements for Assessment Results . . . 73
6.19 Excerpt of insert new Implementation query to add Assessment . . . 73

1

Introduction

Data mining (DM) addresses the challenge organizations face in extracting valuable insights from large data sets, where traditional analytical methods often fall short. By applying intelligent algorithms, DM converts extensive raw data collections into meaningful information and knowledge, supporting improved decision-making (Pacha et al., 2021). In practice the **CR**oss **I**ndustry **S**tandard **P**rocess for **D**ata **M**ining (CRISP-DM), a standardized framework, is used by many organizations as a guideline to apply data mining methods (Schröer et al., 2021). One of the many challenges when trying to extract information and generate knowledge from datasets lies in the question of the reliability of the predictions made (Staudinger et al., 2024). The importance of reliable predictions is not to be underestimated as it can have huge implications for businesses or patients in medical care. An illustrative example would be a classification model that misclassified a patients risk for heart failure (Myers et al., 2020). Another example outlined by Staudinger et al. (2024) is a classification model that helps a employee of a bank to decide whether or not a customer gets a loan. There are many more examples like these from a wide variety of fields, such as biology, physics, etc., which demonstrate how important the reliability of predictive analytic results is for decision-making in the respective sectors. This is precisely why there is a vast amount of literature available on different approaches to assess the reliability of analytic results for all different kinds of specific prediction models.

Therefore, the design and implementation of a knowledge base are examined in this master's thesis to help users navigate through and find applicable assessment approaches that allow them to assess the reliability of analytic results they got from specific prediction models like classifiers or clustering models. The implemented user interface (UI) of the knowledge base is a web-based application on the client side and features a RESTful web-service on the server side which queries the necessary data from a knowledge graph (KG) by executing queries written in SPARQL, a query language for the Resource Description Framework (RDF). The design of the aforementioned KG is in alignment with the knowledge graph developed by Staudinger et al. (2024) where they propose a reference process, based on CRISP-DM, for assessing the reliability of predictive analytic results. The system presented within this thesis is split into three major parts; the **Browsing Section**, the **Guided Section** and the **Results Section**. The Browsing Section lists all the options available to users for performing reliability assessments, categorized by element type like assessment approach or implementation. The Guided Section allows users to use filter options to narrow down the characteristics of an individual use case step by step in order to find a suitable option for reliability assessment. The Result Section lists all reliability assessments that are stored in the knowledge

graph. The complete project can be found on GitHub[1].

The remainder of this thesis is structured as follows: Chapter 2 describes the state of the art of the concepts and technologies relevant to this thesis. Chapter 3 reviews related work, mainly the assessment approaches integrated in the knowledge base as well as the structure of the underlying knowledge graph. Chapter 4 explains necessary design decisions concerning of the architecture of the web based knowledge base as well as the technology stack used. Chapter 5 provides a documentation of the user's point of view, explaining the front end in detail. Chapter 6 outlines the interaction between the components of the system and provides a detailed view on the implementation of the front- and backend. Lastly, Chapter 7 provides a brief summary of the thesis.

[1] https://github.com/Celthrimon/workbench-masterarbeit

2

State of the art

This chapter provides an insight into the theoretical foundations of the work and the state of the art of technologies that were used to implement the knowledge base. It covers the relevant aspects of CRISP-DM, knowledge graphs and knowledge bases relevant to this thesis.

2.1 Data Mining

The process of data mining focuses on the discovery of interesting patterns, models and various forms of knowledge derived from large data sets (Han et al., 2022). For that reason, data mining has become an essential part of the knowledge discovery used in organizations to support decision-making or gain competitive advantages (Pacha et al., 2021). Furthermore, data mining is also a creative process that demands different skills and knowledge from a particular person or team working on a given project (Wirth and Hipp, 2000). Consequently, the success or failure of such data mining projects is highly dependent on the people working on the project and the framework used as guidance through the data mining process (Wirth and Hipp, 2000). One such framework is called CRISP-DM, a framework independent of the industry sector and the technologies used for different data mining tasks throughout the process (Wirth and Hipp, 2000).

2.1.1 CRISP-DM

CRISP-DM consist of six iterative phases illustrated in Figure 2.1 below. The arrows shown in the figure describe what steps can be taken in any current phase. Note that there are phases where the arrow points backwards, indicating that these phases can be repetitive. The big outer circle of arrows shows that the CRISP-DM process does not end with the deployment of a product but can repeat itself beyond that point. In the following each phase will be described briefly, pointing out important information in the context of this thesis.

Business Understanding

The main focus of this phase is to get an overview of the business situation and the available and required resources needed for the data mining process (Schröer et al., 2021). Furthermore, an understanding of the project objective is necessary as one of the main aspects of the business understanding phase is to derive a data mining problem and a data mining goal (Schröer et al., 2021; Wirth and Hipp, 2000). Wirth and Hipp (2000) additionally points out that a preliminary project plan should be designed to achieve the predefined objectives. Likewise, Schröer et al.

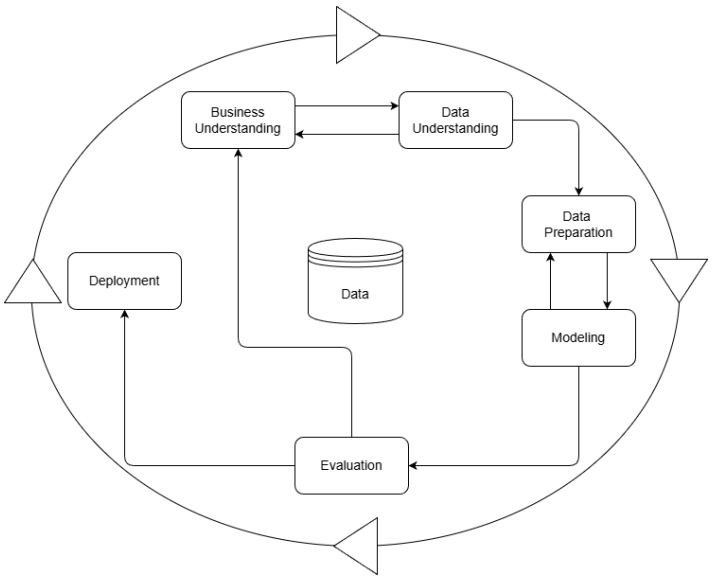


Figure 2.1: CRISP-DM phases

(2021) suggests the creation of a project plan as well as the definition of the data mining type and the success criteria of the data mining (such as precision).

Data Understanding

The data understanding phase is closely linked to the business understanding phase and there will typically be a feedback loop between the two phases (Schröer et al., 2021). At this stage the most important tasks are collecting, exploring and describing the available data starting from the initial data collection (Schröer et al., 2021; Wirth and Hipp, 2000).

Data Preparation

The phase of data preparation encompasses all the procedures needed to create the final dataset, which will be input into the modeling tool(s), starting from the raw data (Wirth and Hipp, 2000). This stage involves tasks that are repeatedly executed and not necessarily in a fixed sequence. Such tasks involve selecting tables, records, and attributes, along with cleaning data, deriving new attributes, and transforming the data for use with modeling tools (Wirth and Hipp, 2000). All these steps can be achieved by different methods and are model dependent (Schröer et al., 2021).

Modeling

During the Modeling phase, appropriate analytical techniques are selected and applied to the prepared dataset. For a given data mining problem, several alternative techniques are usually available, each with distinct methodological requirements (Wirth and Hipp, 2000). The strong link between Data Preparation and Modeling becomes clear at this stage: issues in data quality are often detected only in the course of model development, while the modeling process itself may also generate new requirements or ideas for constructing additional data features (Wirth and Hipp, 2000).

Evaluation

In the evaluation stage one or more models are already built and it is important to evaluate the models and review the steps taken to construct the models until now (Wirth and Hipp, 2000). The results of the models need to be checked against the previously defined business objective and data mining goals as to determine whether any significant business issues have not been sufficiently considered by the models already in place (Schröer et al., 2021; Wirth and Hipp, 2000). The end goal of this phase is to define any further action if necessary and decide on how to move forward with the data mining results provided (Schröer et al., 2021).

Deployment

The deployment phase does not mark the end of a data mining project and the exact tasks in this phase are generally defined in the project plan (Schröer et al., 2021; Wirth and Hipp, 2000). Usually, the knowledge obtained from the CRISP-DM process is organized and presented in a way the organization can benefit from them in day-to-day business and can be structured as a simple report or a complex implementation of a repeating data mining process (Wirth and Hipp, 2000). Additionally, monitoring and maintenance of the knowledge are also part of this phase (Schröer et al., 2021). Lastly, it is important to understand what actions can be taken from a business point of view with the generated models and the resulting knowledge.

2.1.2 Knowledge Representation

In knowledge-based systems, information is stored in a knowledge base in the form of statements about some domain, that contain surrogates for real world domain artifacts known as symbols (Grimm, 2009). The domain stored in such systems can cover various parts about either a hypothetical or real world system, about which one aims to represent knowledge for computational purposes (Grimm, 2009). Knowledge representation aims to design computer systems that can understand and interpret the relationship between the symbols stored in a knowledge base (Grimm, 2009).

To achieve this a common way to represent knowledge in information systems are ontologies. An ontology is a model that makes the knowledge of a domain of interest available to computer systems (Grimm, 2009). Ontologies can either be represented in a graphical or formal way like a semantic network or in some kind of machine-readable serialization format for the purposes of storing or data transfer (Grimm, 2009). A knowledge engineer for example will mostly work with visualizations in the form of semantic networks (Grimm, 2009). However, not all information contained in ontologies can be visualized easily in a graph. Therefore, formal languages such as OWL or RDF are used to display more detailed and complex information that should be part of an ontology (Grimm, 2009).

A specialized application of ontologies is their role in the vision of the Semantic Web. As Grimm (2009) briefly discusses, the general idea of the Semantic Web is to make the Web machine-understandable so that machines can not only use data for display purposes but also to enable automation or reuse across applications on a semantic level (Grimm, 2009). To achieve this, the ontology languages mentioned above, such as OWL and RDF, are used (Grimm, 2009).

RDF(S)

The Resource Description Framework (RDF) and an ontology language RDF Schema (RDFS) emereged as part of an ongoing effort of metadata standardization from the World Wide Web consortium (W3C) (Grimm, 2009). In RDF, resources can be all kinds of objects, such as people, documents, or other physical or logical components that can be exchanged between different applications without losing any information ('RDF 1.1 Primer', n.d.). The aforementioned statements about resource information always follow the same structure in RDF: "<subject> <predicate> <object>" and always describe a relationship between the subject and the object through the predicate, which is read in a unidirectional direction from the subject to the object (RDF 1.1 Primer, n.d.). Since the statements always consist of three elements, as can be seen in Listing 2.1 below, they are called "triples":

Listing 2.1: RDF triples

```
<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.
<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video 'La Joconde a Washington'> <is about> <the Mona Lisa>
```

The RDF framework is defined through several normative specifications, including abstract syntax and model-theoretic semantics, as well as a number of serialization formats such as Turtle, JSON-LD, or RDF/XML ('RDF 1.1 Primer', n.d.). These standards guarantee consistency in both representation and interpretation. In practical terms, RDF is employed in a wide range of scenarios: embedding machine-readable metadata into web pages to improve retrieval by search engines, enriching datasets by linking them to external sources such as Wikidata, connecting data feeds from different APIs, or enabling cross-dataset queries within an organization through SPARQL ('RDF 1.1 Primer', n.d.).

SPARQL

SPARQL 1.1 represents the current W3C standard for querying and manipulating RDF data. It extends the earlier SPARQL 1.0 specification (2008) by introducing both new query features and a broader ecosystem of related protocols. The framework is composed of several components, including the query language itself, update mechanisms, protocols for data exchange, and specifications for service descriptions and testing. Together, these documents define how RDF data can be retrieved, combined, modified, and shared across distributed systems ('SPARQL 1.1 Overview', n.d.).

SPARQL 1.1 extends basic graph pattern matching with more advanced constructs. It allows for nested queries, aggregation functions, or path expressions that capture longer connections in a graph. Besides returning tables of results through SELECT, queries can also generate new RDF graphs with CONSTRUCT or test for the presence of information using ASK. Compared to the first version of the standard, the update introduces grouping, value assignment, and richer navigation options, which make more complex forms of analysis possible while retaining a declarative style ('SPARQL 1.1 Overview', n.d.).

Beyond querying, SPARQL 1.1 also defines an update language, which specifies how RDF graph

stores can be modified through insertions, deletions, and replacements. Update operations may be expressed as standalone requests or chained into more elaborate sequences. Importantly, updates can be conditional on query results, making it possible to define context-sensitive modifications. For example, new information can be inserted into a graph while simultaneously removing outdated or language-specific values ('SPARQL 1.1 Overview', n.d.).
Beyond the query language, SPARQL 1.1 defines ways to send queries and updates over HTTP and to describe available services. It also standardizes several output formats, such as XML, JSON, or simple CSV/TSV tables, so results can be used with common tools. Together, these elements make SPARQL more than a query language: they provide an infrastructure for combining data from different sources, managing RDF graphs, and supporting federated access ('SPARQL 1.1 Overview', n.d.).

2.2 Knowledge Graph

A knowledge graph is a representation of triplets as a directed graph with edges as relationships and nodes as entities (Peng et al., 2023). Triplets are defined as the representation of real-world facts and semantic relations between data in a typical dataset mostly in the form of object-predicate-subject (Peng et al., 2023). Additionally, an ontology that shows the properties of a specific domain and the relationships between those properties, is the definition of the schema of a knowledge graph (Peng et al., 2023). In the context of this thesis, this representation is realized through an RDF-based knowledge graph, where entities such as Assessment Approaches, Methodologies and Implementations are modeled as nodes and linked through object properties. These links reflect the abstraction levels of the reference process, which will be detailed later in Chapter 3.Therefore, Peng et al. (2023) highlight that ontology design is an essential step in the construction of a knowledge graph. As previously mentioned, the typical way to represent a knowledge graph with a formal language is RDF (Peng et al., 2023). Peng et al. (2023) discuss a number of different research lines regarding the technical challenges and opportunities of knowledge graphs, some of which will be highlighted in the following:

1) Knowledge Graph Embedding One of the main research concerns in knowledge graph studies is the problem of embedding. The idea is to represent entities and relations in a compact, low-dimensional vector space, while still keeping as much of the graph's structure and semantics as possible. Once expressed in this form, the information can be handled more easily by machine learning techniques, which are not well suited to working directly with symbolic triples.

While embedding techniques are widely used in machine learning-driven applications, the knowledge base developed in this thesis relies on symbolic RDF representations and explicit SPARQL queries. This design choice prioritizes transparency and traceability of relationships, which is essential for documenting and explaining reliability assessment decisions and for maintaining a clear link between user actions, system recommendations, and stored assessment results.

2) Knowledge Acquisition Another important line of work is knowledge acquisition, which deals with how knowledge graphs are modeled and constructed. At the same time, a large part of research looks at how to extract useful information from unstructured sources like news

articles, research publications, or patents. For these cases, techniques such as relation extraction, entity recognition, or attribute identification are applied to transform text into graph-ready triples.

In the implemented system, knowledge acquisition is realized primarily through user interaction. The knowledge base provides structured input forms and RESTful endpoints that allow users to insert new assessment approaches, methodologies, implementations, and filter options directly into the knowledge graph. This enables controlled and schema-consistent growth of the system without requiring changes to the underlying ontology, thereby supporting the extensibility goal of the thesis.

3) Knowledge Graph Completion Despite the progress in constructing knowledge graphs, it remains unrealistic to cover an entire domain comprehensively. Most existing graphs still miss a considerable number of entities or relations, which limits their usefulness. This is where knowledge graph completion, a set of techniques aimed at filling in the gaps and raising the overall quality of the graph, becomes essential. Typically, two approaches are discussed in the literature. One relies on link prediction, where new triples are generated and then evaluated with plausibility scores to decide whether they should be added. The other approach focuses on entity prediction, which tries to discover and integrate missing entities from external data sources.

Although automated completion techniques are not part of the current implementation, the extensible ontology design of the knowledge base allows missing entities or relations to be added incrementally. This can be achieved either manually by users through the provided interfaces or through future integration of semi-automated completion mechanisms, such as importing assessment approaches from external repositories or literature databases.

4) Knowledge Fusion Knowledge fusion addresses the integration of information from different sources into a unified knowledge graph. Fusion methods play a dual role: they support the initial generation of graphs by bringing together heterogeneous data, and they also contribute to graph completion by adding missing facts from complementary sources. Entity alignment has emerged as the primary method of knowledge fusion tasks.

This principle is reflected in the knowledge base by linking externally developed tools and packages, such as the Perturbation Tool and the ADAN R package, into a unified semantic structure. By modeling these heterogeneous resources within the same ontology, the system allows users to compare and navigate assessment approaches originating from different research contexts through a common interface and a shared conceptual framework.

5) Knowledge Reasoning Knowledge reasoning aims to infer new facts from existing data already contained in the knowledge graph. It can uncover hidden relations between entities, generating additional triples, and also detect false or inconsistent information. Approaches in this area range from logic rule-based methods to distributed representations and neural network models.

In the system presented in this thesis, lightweight forms of reasoning are realized through filter and recommendation mechanisms that exploit explicit semantic relationships in the knowledge graph, such as *requires*, *recommends*, and *prohibits*. These relationships guide users toward suitable assessment approaches based on their selected prediction context, while preserving explainability by making the underlying decision paths visible.

Application of Knowledge Graphs

An important aspect discussed by Peng et al. (2023) is the application of knowledge graphs and the functionality they provide to different kinds of systems. In the context of this work the relevant systems discussed are *information retrieval systems* and *recommender systems*.

Information Retrieval Systems Information retrieval traditionally relies on matching the words in a user's query with words that appear in documents. This works reasonably well for many tasks but struggles when language is ambiguous. For example, the same word can mean different things depending on context, and two different words may refer to the same concept. As a result, keyword-based systems can return either too many irrelevant documents or fail to include some that are actually important (Peng et al., 2023).

When knowledge graphs are added to the process, queries and documents can be interpreted semantically rather than just textually. Instead of looking only for word overlap, the system can recognize that certain entities are related or even identical across different expressions. A search for "coronavirus," for instance, can automatically expand to include documents referring to "COVID-19." This approach not only improves recall and precision but also makes results easier to explain, since the connections between terms and documents are explicit in the graph (Peng et al., 2023).

This concept is directly reflected in the Browsing Section of the knowledge base, where users can explore assessment approaches, methodologies, and implementations through semantically structured filters rather than keyword-based search. This enables more precise and context-aware retrieval of relevant elements across different abstraction levels.

Recommender Systems Recommender systems have become essential for helping users navigate the overwhelming amount of digital content available today. Traditional methods usually rely on two techniques: content-based filtering, which recommends items with similar features to those a user already likes, and collaborative filtering, which compares users with similar behavior patterns. Both approaches, however, face well-known problems. Data is often sparse, with only a small fraction of items receiving ratings, and new users or products create the cold-start problem, since there is little or no history to work with. In addition, systems that rely too heavily on user data may also raise privacy concerns (Peng et al., 2023).

Knowledge graphs offer a way to reduce some of these limitations by providing extra context in the form of structured connections between entities. Instead of relying only on ratings or tags, a KG can link a movie to its director, genre, or leading actors, and then relate these to user preferences. This means a new film can be recommended because it shares a director with another film a user enjoyed, even if no one has rated it yet. Another strength of KG-based recommenders is that they can explain their decisions more clearly: it is possible to trace the reasoning path through the graph, making the suggestion easier to justify to the user (Peng et al., 2023).

Similarly, the Guided Section of the system presented in this thesis adopts a knowledge-graph-driven recommendation principle. By progressively narrowing down suitable assessment approaches based on user-selected prediction methods and contextual constraints, the system guides users through semantic relationships in the graph while maintaining transparency by exposing the conceptual basis for each recommendation.

3

Related Work

This section covers similar work in relation to the knowledge base implemented in this thesis. Not only does its main focus lie on the reference process used for the development of the knowledge base, but also covers aspects of the design of similar work in the context of knowledge bases, workbenches and tool kits. Furthermore, this section highlights work from literature that has already been included in the knowledge base.

3.1 Reference Process

The reference process proposed by Staudinger et al. (2024) aims to solve the problem of assessing the reliability of predictive analytic results. The process is based on the CRISP-DM and additionally defines activities along the entire CRISP-DM lifecycle (Staudinger et al., 2024). Staudinger et al. (2024) argues, that in order to assess reliability of results accurately, the input data, the training data and the data from procedures to collect and prepare data must be considered by the analyst trying to assess the predictions and is thus stored in a knowledge graph. The relevant aspect of the reference process for this thesis is the structure of the knowledge graph and the three abstraction levels, namely, generic, method-specific and instance-specific. However, it is also necessary to introduce the key differences regarding the stages of the reference process compared to the CRISP-DM. Figure 3.1 shows the reference process aligned with the CRISP-DM stages as well as the activities used specifically in the reference process:

Business Understanding

As previously described in Chapter 2.1.1, business understanding is the first step of the CRISP-DM and also of the reference process. In contrast to CRISP-DM however, the main focus does not lie on defining a data mining goal, but to choose an appropriate assessment approach for reliability assessment. That does not mean however, that the data mining goal is not important, as it influences the decision of prediction method, e.g. classification which directly influences the choice of assessment approach.
For classification problems, the reference process introduces two predefined assessment approaches: the local quality measure approach, which requires continued access to training data after deployment, and the perturbation approach, which is useful for identifying sensitive input features. Each option has implications for project planning, such as data availability, privacy restrictions, or

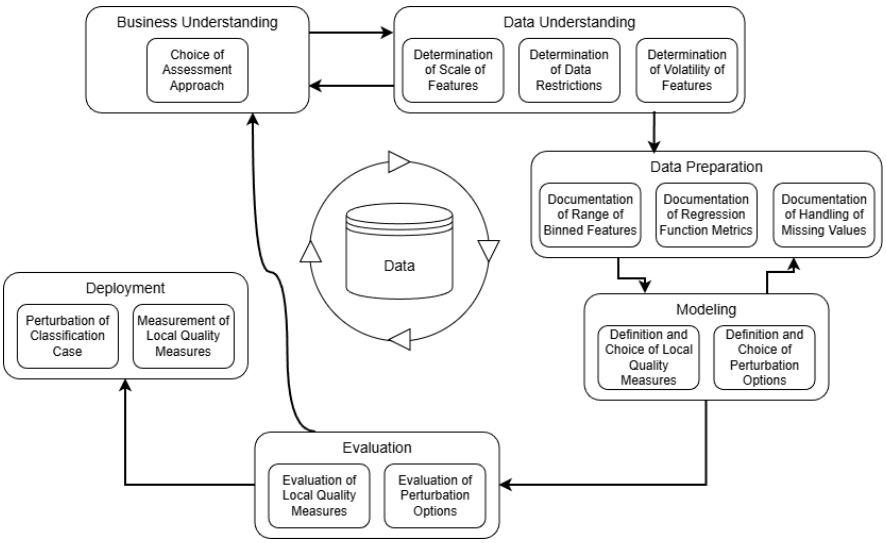


Figure 3.1: Example activities for enabling assessment of reliability of classification results (Staudinger et al., 2024)

additional implementation effort. In the proposed reference process, these decisions are represented at three abstraction levels. On the generic level, broad activities and entities are defined like *BusinessUnderstandingEntity*. On the method-specific level, these are refined into tasks like *ChoiceOfAssessmentApproach* or entities like *AssessmentApproach*. Finally, at the problem-specific level, concrete instances are created, for example choosing perturbation for reliability assessment in a telemarketing use case. These entities and activities are mentioned because they are directly related to entities and activities used in the design of this thesis's knowledge base.

Data Unterstanding

As in CRISP-DM, the data understanding stage comes after the business understanding and is closely related to it. In data mining projects, input can come from structured sources, streams, images, or videos. During the data understanding stage, the relevant data are collected and their quality is assessed.

For structured data, the reference process defines three important aspects of data understanding. First, the level of measurement (nominal, ordinal, cardinal) must be determined, since it influences which reliability checks are possible. Second, the volatility of features needs to be considered, as unstable values such as income or wind speed can make predictions less reliable compared to stable attributes like education level. Third, restrictions and accuracy must be recorded. Some restrictions are general (e.g., age cannot be negative), others are domain-specific (e.g., minimum age for opening a bank account). Accuracy depends on collection methods; rounded values or sensor tolerances should be expressed as intervals or error margins.

In the reference process, these aspects are modeled at three levels: generic (e.g. *Data Understanding Entity*, *Scale*), method-specific (e.g., *Scale of Feature*, *Volatility of Feature*), and problem-specific (concrete cases such as marking a telemarketing balance feature as cardinal). The data understanding aspect of the reference process is relevant for the developed knowledge base, as properties such as scale of feature are included in the filter options used to find suitable options for reliability assessment.

Data Preparation

The next step of the reference process is data preparation. Like in CRISP-DM, once the necessary data have been gathered, the data preparation stage deals with cleaning, transforming, and adapting them so that they can be used to train predictive models. This step may involve improving accuracy, reducing noise, filling in gaps, or resolving inconsistencies. The reference process highlights methods such as binning values into intervals, smoothing features with regression functions, or imputing replacements for missing entries are often applied. While these techniques are useful, they also create approximations that differ from the original values, which has to be taken into account when judging the reliability of later predictions. For that reason, every transformation should be documented. Within the reference process, these tasks are modeled at several levels of abstraction, from general activities to method-specific procedures, down to concrete cases such as recording how missing income values were treated in a telemarketing dataset. This layered view ensures that all preprocessing steps remain transparent and that their impact on reliability can be traced throughout the project. The developed knowledge base does not provide features for data preparation as it is not directly correlated in helping the user find suitable assessment approaches.

Modeling and Evaluation

Like in CRISP-DM, the next stages of modeling and evaluation, the focus lies in the creation of an appropriate model for the data mining goal, and the evaluation of the results of the model, checked against the data mining goals. The reference process describes two reliability assessment approaches, namely, the perturbation approach and the local quality measures approach. The relevance of the perturbation approach for this thesis lies in the integration of a tool that has implemented the reference process, more precisely the perturbation approach. This tool as well as the deep integration level of this tool will be further examined in Section 3.2.1 of this thesis. For a general understanding of how the perturbation of input cases proposed by Staudinger et al. (2024) works, the following explanation will summarize it in a concise manner.

Perturbation Approach

The perturbation method is built around two components: options and modes. Options describe how the value of a feature can be varied in order to create new versions of a case. These modified cases are then sent through the model to see if the prediction changes compared with the prediction of the unaltered original case. Each option is given a level that signals how to interpret the results: red means the prediction should stay stable within the perturbed range, orange means changes are possible when the input lies close to a decision boundary, and green means a change is expected. Designing such options relies heavily on the information gathered earlier during data understanding and preparation. For instance, whether a feature is nominal, ordinal, or cardinal limits what kind of changes are possible. Other aspects, such as how volatile a value is in the real world, whether restrictions apply, or how precise the measurements are, also play a role. Even preparation steps like binning, regression smoothing, or the replacement of missing values influence what perturbations make sense.

Because the number of possible variations can grow very quickly, different modes are used to keep the process manageable. A full mode simply tries out every possible combination of perturbed values. A prioritized mode gives preference to features considered more important, while a selected

mode restricts the exercise to a small set of chosen features. In the process model these activities are described at several levels: a generic layer for the overall concepts, a method-specific layer that defines concrete perturbation options, and a problem-specific layer with actual project cases, such as applying a $\pm 10\%$ change to the balance attribute in a telemarketing dataset. In this way, perturbation is tied back to the characteristics of the data while remaining flexible enough to adapt to different application settings.

Deployment

Deployment is the final step of CRISP-DM, where the finished model moves from the development environment into everyday business use. New input cases are processed by the model to produce predictions, and these outputs serve as a basis for business decisions. At this stage, the reliability checks prepared earlier come into play. Perturbation options, for instance, generate slightly altered versions of the input data to see whether predictions remain stable. When perturbing different features, generally some amount of alternative cases will be created. These cases can sometimes produce different outcomes in the prediction, showing that predictions can shift under small changes in the data and that reliability must be considered before relying on the model's output. Alongside perturbation, local quality measures were used to examine prediction reliability at the level of individual cases. For example, the local accuracy around the input case could reach around 90%, compared with a global accuracy, which could be in the realm of around 80%. This suggests the model works particularly well in that local region of the feature space but also indicates that other regions may perform less reliably. To support later analysis of the data, a knowledge graph captures the design choices, assessment steps, and evaluation results, linking them back to earlier CRISP-DM stages. In practice, this documentation ensures that predictions can be audited and that the applied reliability checks remain transparent and consistent when the model is in use.
In this thesis, results like the evaluation results of the Perturbation Tool are stored in the knowledge graph according to the structure of the reference process and are then displayed in the **Results Section** of the implemented knowledge base, which will be highlighted in later chapters of this work.

3.1.1 Knowledge Graph

At every stage of the proposed reference process, metadata about the performed activities should be captured in a knowledge graph (KG). This graph records which agents carried out which tasks, the entities they produced, and the design decisions that shaped the predictive model and its reliability assessment. In this way, the KG serves as provenance information, building on the W3C PROV-O standard, and provides a structured record of how models were developed and how reliability methods were selected. Three main elements are central: activities, entities, and agents. Activities represent the tasks carried out, entities are the artifacts created, and agents are the people or systems responsible. The KG is organized into three abstraction levels—generic, method-specific, and problem-specific—and for each level it distinguishes between a development view, which documents model creation, and a deployment view, which records how methods are applied in practice.
On the generic level, the KG defines a basic vocabulary for modeling activities and entities across all CRISP-DM stages. It connects entities to the activities that produced them, records when and

by whom they were created, and tracks relationships such as derivations or adjustments (e.g., a perturbation option refined with new parameters). This provides a common foundation for more specific processes. The method-specific level refines these abstractions for particular prediction methods. In the case of classification, it includes entities such as Perturbation Approach, Scale of Feature, Volatility of Feature, or Perturbation Option, along with their connections to relevant features and assessment methods.
By structuring metadata in this way, the KG not only documents the development process but also supports transparency, auditability, and reuse. Although illustrated here with structured classification tasks, the same framework can be adapted to other predictive methods by refining the method-specific layer. Its flexibility stems from using an ontology-based representation rather than a fixed schema, allowing extensions as new processes and assessment methods are introduced.

3.2 Assessment Approaches

This section introduces three relevant assessment approaches with which the knowledge base was initially populated, so an understanding can be reached with what types of theoretical and practical work the knowledge base can be extended, as the design of this knowledge base emphasizes the extensibility of the system. Later in this thesis, different integration levels of assessment approaches will be brought to attention, namely, *loose integration*, *deep integration* and *partial integration*. For this reason, each assessment approach discussed in the following, will correspond to one of these three levels. First, the Perturbation Tool by Badzura (2023) represents a tool that is ready-to-use from within the knowledge base itself, thus falls into the deep integration level. Secondly, the Applicability Domain ANalysis (ADAN) by Carrió et al. (2014) describes a method for assessing the reliability of drug property predictions and comes with an R package that can be accessed via the knowledge base. Lastly, an approach to identify unreliable prediction in clinical risk models by Myers et al. (2020) is the example chosen for a loosely integrated approach in the knowledge base.

3.2.1 Perturbation Tool

The Perturbation Tool developed by Badzura (2023) is based on the reference process introduced earlier in this thesis and outlined by (Staudinger et al., 2024). It tackles the challenge of assessing the reliability of individual predictive analytic results by introducing a streamlined, web-based user interface that guides the user through the various phases of the reference process. While using this tool, users may document the actions taken throughout the reference process in a knowledge graph, information that is used as a basis for generating perturbation options (Badzura, 2023). Finally, the generated knowledge after perturbing sensitive values can be used to efficiently evaluate the reliability of predictive analytic results (Badzura, 2023).
The choice of this tool as the representative of how an assessment approach can be integrated deep into the knowledge base is two-fold. First, the underlying knowledge graph of the perturbation tool and the knowledge graph designed in the context of this thesis are both based on the knowledge graph developed by (Staudinger et al., 2024). Secondly, the knowledge base features direct access to the information stored in the knowledge graph of the Perturbation Tool. Therefore, the tool is accessible from the knowledge base and users can view all the assessments generated by the Perturbation Tool directly in the **Results Section** of the knowledge base.

3.2.2 ADAN

Carrió et al. (2014) introduce an approach, called ADAN (Applicability Domain ANalysis), designed to evaluate the reliability of drug property predictions generated through in silico modeling. The method works by comparing the query compound against the training set using six distinct similarity criteria. The results are aggregated into a score between 0 and 6, indicating how many criteria were not met. This score is then used to assign the compound to one of seven reliability categories, which can be further simplified into broader classes using a traffic-light scheme, approximate confidence intervals, or by labeling certain predictions as unreliable.
The criteria for an approach to be integrated at the partial integration level are that there is not only a theoretical explanation on how to use a certain approach but also a practical option included like tool or a package. The inclusion of ADAN in the knowledge base to represent the medium integration level, stems from the fact that the work published by Carrió et al. (2014) includes an complete implementation in form of an R package. This package can be accessed by from the knowledge base and furthermore, executed outside the scope of the knowledge base. However, in contrast to the Perturbation Tool, the knowledge base does not have unrestricted access to a knowledge graph or a similar method of storing assessment results because the package alone does not provide persistence of assessments.

3.2.3 Identifying unreliable predictions in clinical risk models

The study by Myers et al. (2020) presents a method for pinpointing patient subgroups in which a predictive model is likely to perform poorly, signaling when its output may be misleading and should be treated with caution. The objective was to design a patient-level metric capable of signaling when a prediction should not be trusted. For such a metric to be clinically valuable, Myers et al. (2020) argue that it must satisfy several criteria: (1) it should reliably identify subgroups of patients where the model performs poorly; (2) it should be independent of the specific model being used; (3) it should not rely on retraining or require access to the original training data, allowing clinicians to apply it even without direct knowledge of how the model was built; and (4) it should remain effective in contexts characterized by substantial class imbalance. The approach produces an "unreliability score" that can be applied to any clinical risk model and is particularly well suited for datasets with substantial class imbalance, a common issue in healthcare. Using information from over 40,000 patients in the Global Registry of Acute Coronary Events (GRACE), Myers et al. (2020) show that individuals with high unreliability scores represent a subgroup where the model's accuracy and discriminatory power are both reduced.
This method represents a loosely integrated assessment approach in the knowledge base as no implementation is provided in the context of the work done by (Myers et al., 2020). However, this does not mean that similar work should not be included in the knowledge base, as it is conceivable that an implementation of loosely integrated assessment approaches could be developed in the future and added to the knowledge base afterward.

3.3 Similar KB Designs

The following section highlights similar knowledge base designs from literature focusing on their architectural designs and capabilities. The first toolkit is called Algorithm Development and Mining

(ADaM) proposed by Rushing et al. (2005) and is a data mining toolkit for use with scientific data. It is a toolkit packaged as a suite of independent components with the main focus of providing classification, clustering and association rule mining methods commonly used in many data mining systems (Rushing et al., 2005). Furthermore, it provides feature reduction, image processing, data cleaning and preprocessing capabilities Rushing et al., 2005. The key design issues outlined by Rushing et al. (2005) to develop the toolkit were packaging of the mining capabilities, the flow model used within the toolkit, the data models used and lastly the import/export feature for data. To tackle these issues, ADaM consist of different layers; a layer for data models at the bottom of the architecture and a layer of tools for data mining and image processing purposes on top of the data layer (Rushing et al., 2005). Workflows and applications are constructed from the modules using script languages and can be mixed and matched to construct data mining applications (Rushing et al., 2005).

Next, the NLP Workbench presented by Yao et al. (2023), is a web based platform integrating various text-mining and natural-language-processing (NLP) tools in a unified and extensible framework. The system was developed for non-expert users to simplify access to advanced NLP models and reduce technical barriers (Yao et al., 2023). The system architecture relies on a containerized microservice setup where each analytical function, such as entity recognition, sentiment analysis, or relation extraction, is deployed as an independent Docker service (Yao et al., 2023). These services communicate through REST or RPC interfaces and are coordinated by a task scheduler that organizes dependencies and balances workloads (Yao et al., 2023). Data indexing and result visualization are handled by components such as Elasticsearch and Neo4j, which provide search capabilities and graph views of extracted relations (Yao et al., 2023). By combining modular deployment with a unified interface and shared data layer, the NLP Workbench shows how analytical tools can be organized into a coherent platform that remains flexible and easy to extend (Yao et al., 2023). Its principles of modularity, interoperability, and container-based execution mirror many current trends in the design of integrated analytical systems.

Lastly, El Hachimi et al. (2022) developed the Data Science Toolkit (DST), an open-source Python package that brings together several common steps of the data analysis process in one place. The main idea behind the toolkit is to make everyday data science tasks simpler and faster to set up (El Hachimi et al., 2022). Instead of switching between different libraries and writing the same type of code for every project, users can access predefined functions that already combine elements from libraries like Pandas, NumPy, or Scikit-Learn which makes it easier to handle data preparation, visualization, and model training without spending much time on setup or repetitive implementation (El Hachimi et al., 2022). The toolkit itself is structured in a modular way meaning it introduces classes for the main parts of the workflow, such as DataFrame for transformation and cleaning, Chart for visualization, and Model for building and testing machine learning algorithms (El Hachimi et al., 2022). These modules can be used on their own or connected with each other, so a complete analysis—from raw data to results—can be carried out within the same environment.

All three systems — ADaM, the NLP Workbench, and the Data Science Toolkit — follow a similar goal of bringing together analytical tools within a common environment. They are built to simplify how users interact with complex analytical methods and to reduce the effort needed to combine different components. Each of them organizes independent modules into a single, usable framework that supports a defined workflow. The difference lies mainly in the level of abstraction. ADaM and the NLP Workbench focus on building and running workflows, while the Data Science Toolkit

concentrates on the implementation of algorithms in a unified programming interface. The system presented in this thesis takes another perspective. It does not execute methods but represents them conceptually. The focus is on structuring existing approaches and making their relationships visible through a knowledge graph. In this sense, the system shares the integrative mindset of the other toolkits but applies it to the organization of knowledge rather than computation.

4

Architecture

The following chapter discusses in-depth the design of the system and its corresponding implementation. A description of REST endpoints will be provided to give a comprehensive understanding about the design of the communication between the server and client sides. Finally, this section will conclude by presenting the complete technology stack used for the implementation of the system as well as give detailed explanations about why technologies were chosen.

4.1 System Overview and Component Responsibilities

The system is organized according to a client–server model with three main parts: a web-based frontend, a backend application, and a graph-based persistence layer. Each of these components handles a specific set of responsibilities, and their separation reflects the intention to keep presentation, application logic, and data management clearly distinct. Figure 4.1 shows a simplified view of this structure and the main communication paths between the components. The diagram does not aim to capture every technical detail but instead highlights the roles each part plays in the overall workflow.

The **frontend** provides access to the main functional areas of the knowledge base, including the Guided Section, the Browsing Section, and the Results Section. Actions performed by the user, such as choosing a prediction method, applying filters, or creating new elements, are translated into requests that are sent to the backend. In addition to rendering the interface, the frontend also handles simple checks on user input and keeps track of the current state of the application so that changes are reflected consistently across different views.

The **backend** serves as the connection point between the user interface and the knowledge graph. Its primary role is to handle incoming requests, validate the given data, and carry out the corresponding operations on the stored graph. This includes tasks such as connecting newly created entities to existing elements, managing the propagation of filter options, and enforcing the structural constraints of the knowledge graph.

The **persistence layer** consists of an RDF triple store provided by Apache Jena Fuseki. All information in the knowledge base is stored in the form of RDF triples and accessed through a SPARQL endpoint. This layer maintains the semantic structure of the system, including the different abstraction levels and the relationships between elements, such as *requires*, *recommends*, and *prohibits*. It also stores provenance information for assessment cases, which makes it possible to trace results back to the tools and methods that produced them. Because the data is organized

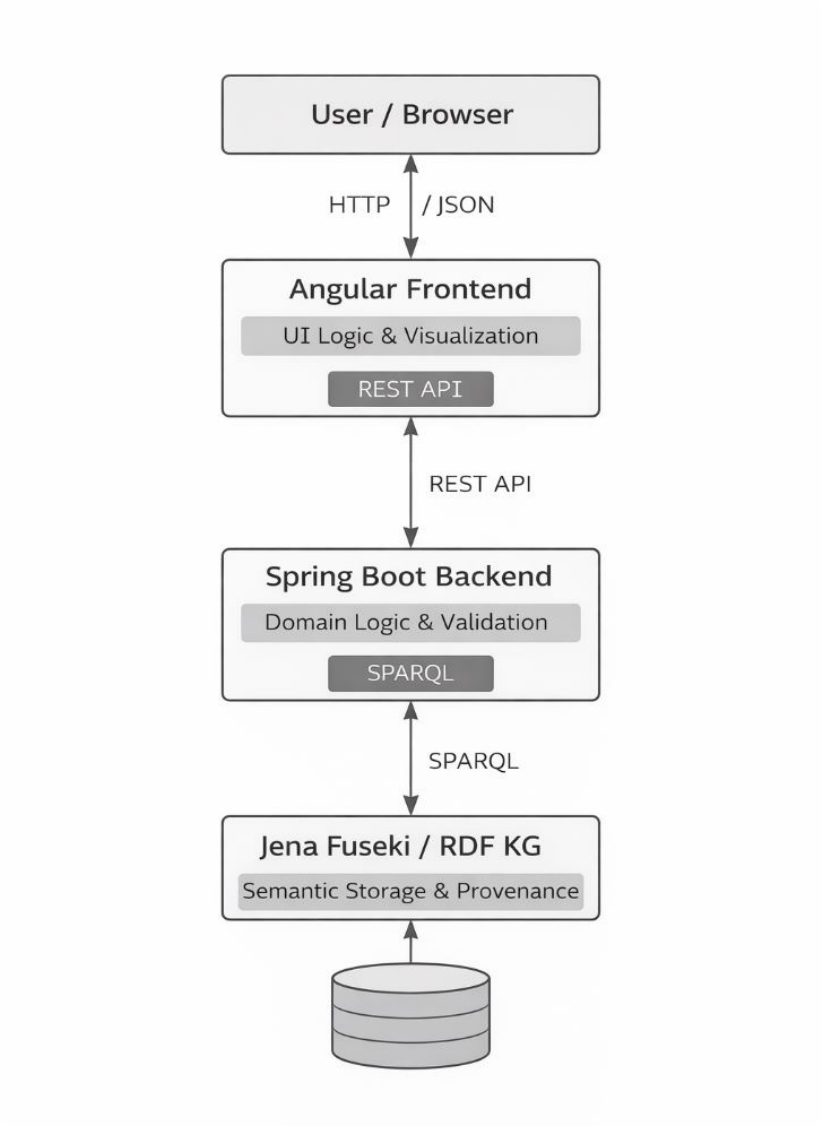


Figure 4.1: Abstract system overview and component responsibilities of the knowledge base.

as a graph rather than in a fixed relational schema, new concepts and relations can be added without restructuring the existing data model. Communication between the components follows a straightforward pattern. The frontend exchanges data with the backend using HTTP requests and JSON-formatted responses. The backend, in turn, interacts with the persistence layer by issuing SPARQL queries and update statements. This setup means that the knowledge graph is not accessed directly by the client, but only through the server, which provides a controlled interface for reading and modifying the stored data. As a result, the system can be extended gradually, for example by adding new assessment approaches or tools, without changing the basic structure of the application.

4.2 Description of REST Endpoints

The REST interface in this project was built with *Java Spring Boot* and acts as the link between server and client. The decision to use a REST-based setup was mainly practical: it is lightweight, easy to extend, and has become the standard approach in most web applications. This makes it well suited for integration with other systems later on.

Data between the components is exchanged in *JSON* format. JSON was chosen because it is both compact and readable, and it can represent structured information such as knowledge graph entities or assessment results without adding unnecessary complexity. Every request must specify the format explicitly, which is done by including the header field `content-type: application/json`. Without this, the server would not be able to correctly process the request.
The interface currently supports two types of operations. *HTTP GET* requests are used whenever information has to be read from the server, for example to load available assessment approaches, fetch prediction methods, or retrieve existing results. *HTTP POST* requests, on the other hand, are used to send data to the server, such as new resources or additional cases for the knowledge base. This clear split between reading and writing helps to avoid confusion and simplifies later maintenance.
Spring Boot offers built-in support for handling these requests, including automatic mapping, data binding, and basic error handling. These features reduce the amount of boilerplate code that would otherwise be required and make it easier to extend the interface if new functionality is needed—for instance, adding further assessment methods or supporting additional data formats.
In the overall architecture, the REST interface takes on the role of middle ware. On one side it connects to the *Angular* frontend, through which users interact with the system. On the other side it communicates with the knowledge graph and the reasoning logic. This ensures that user actions in the frontend, such as uploading a file or triggering an assessment, are consistently reflected in the backend data.
Table 4.1 features a concise overview of all the controllers with their respective HTTP methods and a short description of what each endpoint provides. These endpoints will be discussed in detail during Chapter 6.

4.3 Final Technology Stack

The knowledge base was built on a relatively small set of technologies that complement each other and meet the main requirements of the project. The choice of each tool was not made in isolation but always with the reference process in mind, since this process already makes use of a knowledge graph at its core. In the end, the stack consists of *Java Spring Boot* for the backend, *Apache Jena Fuseki* for managing and querying RDF data, and an *Angular* frontend developed in *TypeScript* with *Bootstrap* for styling.

4.3.1 Backend: Java Spring Boot

The backend was implemented with *Java Spring Boot*. This framework was chosen because it allows REST services to be set up with very little configuration, which fit the needs of the project well. Since the system relies on a REST interface to connect client and server, Spring Boot was a straightforward option. In practice, its large ecosystem and thorough documentation also proved useful during development, as many problems could be solved quickly. The backend was designed according to the Spring Boot documentation on how to build REST services 'Getting Started | Building REST services with Spring', n.d. Another important factor was extensibility: adding new controllers or endpoints later on is relatively simple without breaking the existing structure. Using Java as the base language was also convenient, since it integrates nicely with Jena for RDF handling and is well supported in terms of libraries.

Table 4.1: Controllers with respective HTTP operations

Controller	Path	Description	Method
Filter	/filters	Get all Filters	GET
Filter	/filters/id	Get Filter by uuid	GET
Filter	/filter	Add new Filter	POST
Filter	/filterOptions	Get all FiltersOptions	GET
Filter	/filter/id/options	Get all FiltersOptions by Filter uuid	GET
Filter	/filterOption/filterId	Add new FilterOption to Filter	POST
AssessmentApproach	/assessmentApproaches	Get all AssessmentApproaches	GET
AssessmentApproach	/assessmentApproache	Add new AssessmentApproache	POST
Methodology	/methodology	Add new Methodology	POST
Implementation	/implementation/id	Get Implementation by uuid	GET
Implementation	/implementation	Add new Implementation	POST
GenericTool	/genericTools	Get GenericTools	GET
Assessment	/assessments/id	Get Assessment by uuid	GET
Assessment	/assessments	Add new Assessment	POST
AssessmentCase	/assessment-cases	Get all AssessmentCases	GET
AssessmentCase	/edoc-cases	Get results from PerturbationTool	GET
File	/file-preview?filename	Get File preview	GET
File	/upload	Upload file to Assessment	GET

4.3.2 Database: Apache Jena Fuseki

For persistence, *Apache Jena Fuseki* was used. Fuseki works as a SPARQL server and stores the knowledge graph, which directly matches the reference process where information is also represented as a graph. The choice of a knowledge graph instead of a relational database was made because RDF triples can be queried and extended more flexibly. Adding new entities or relationships does not require schema changes, which is helpful since the system is meant to grow over time with more approaches, tools, and results. Another point worth mentioning is that Fuseki works seamlessly with the Jena libraries in the backend, enabling a smooth integration.

4.3.3 Frontend: Angular with TypeScript and Bootstrap

The frontend was created with *Angular*. One of the main advantages here is its component-based architecture, which made it easier to split the application into smaller, reusable parts such as browsing, filtering, and result views. Angular also integrates well with REST APIs, so connecting it to the backend did not cause major issues. The use of *TypeScript* instead of plain JavaScript gave the project additional type safety and better error detection. For the user interface, *Bootstrap* was chosen because it offers ready-made components and responsive layouts. This saved time compared to building everything manually while still leaving room for adjustments when a custom

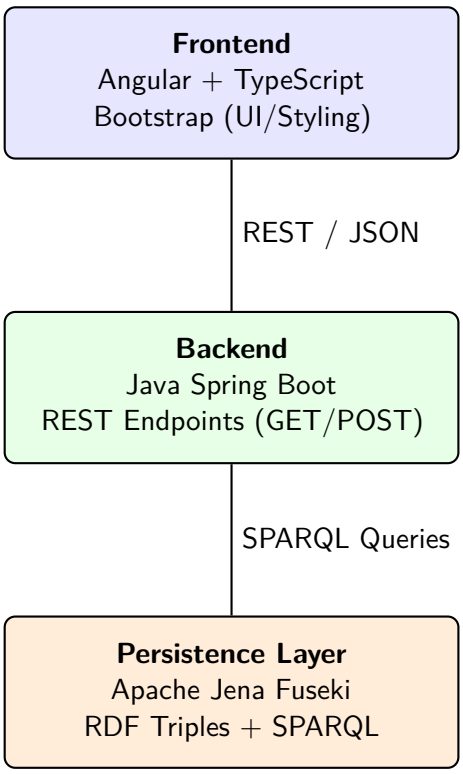


Figure 4.2: Final technology stack of the knowledge base.

look was needed.

4.3.4 Summary

Looking back at the choices made, the mix of *Spring Boot*, *Fuseki*, and *Angular* with *TypeScript* and *Bootstrap* worked well for the purposes of this project. Each part had a clear role to play: Spring Boot was mainly responsible for the REST interface, Fuseki handled the graph-based data storage, and Angular provided a structured way of building the frontend. Some alternatives, such as using a relational database or a different frontend framework, were considered at the start but would have required more effort to adapt to the graph-based approach of the reference process. The final stack is not overly complicated, which helped during development, but still leaves enough flexibility for future extensions when new requirements or features come up.

5

User Perspective

From the perspective of the end user, the knowledge base is organized into several functional sections. The **Guided Section** serves as an entry point where users select a prediction method and are presented with recommended assessment approaches, together with their corresponding methodologies and implementations.
The **Browsing Section** supports free exploration of the knowledge graph, where all available elements—such as approaches, methodologies, tools, and packages—are presented in a structured way. Filters and filter options can be applied to refine the view and focus on specific aspects.
The **Results Section** focuses on the outcomes of executed assessments. Here, users can review stored assessment cases, inspect the corresponding results, and trace them back to the implementation that produced them. In addition to these three main sections, the system also provides pages for populating the knowledge base with new elements. These include interfaces for adding filters and filter options, defining assessment approaches and methodologies, and registering implementations such as packages, tools, or generic tools. Through these pages, users can extend the knowledge base and adapt it to new methods and use cases.
This chapter will provide screenshots of every aspect of the user interface. Each screenshot is accompanied by an explanation of the available functionality, showing how users can interact with the system and contribute to the knowledge base. When a user first accesses the web interface, the system initially displays the **Browsing Section**. Nevertheless, for the purpose of this chapter, the **Guided Section** is introduced first. This order of presentation is more suitable in the context of the user perspective, as it allows the explanation of individual elements to be spread across multiple focused views rather than being presented in the comprehensive "all-at-once" overview that the **Browsing Section** provides.

5.1 Guided Section

In this chapter the three stages of the **Guided Section** will be introduced in detail. Stage 1 will be introduced as an entry point to the knowledge base for the user. Stage 2 represents an overview of all Assessment Approaches and the main page of the guided section will be explained in detail. Stage 3 features the selection of suitable Implementations for reliability assessment of predictions.

5.1.1 Guided Section - Stage 1

The starting point for the **Guided Section** is the selection of a suitable prediction method in order to proceed as can be seen in Figure 5.1. This figure depicts three areas which will be explained in the following. The first area contains the elements of the *Nav bar*. The *Navigation bar* is visible on every page of the user interface and provides the user with guidance in what section of the website they are currently in. As can be seen in the figure the word "Guided" is highlighted in a darker color, indicating that the user is currently viewing a page relating to the **Guided Section**. The second area features a textual explanation of what actions the user can take in stage 1. In this case, the user has the option to select a prediction method applicable to a prediction model in order to proceed. The range of prediction methods available can be extended from within the knowledge base by adding a new Filter Option for the "Prediction Method" filter as described in Chapter 5.1.4. By clicking one of the three available option *"Regression"*, *"Classification"* or *"Clustering"* the next stage will be displayed. Alternatively, the user can proceed to the next step without selecting any of the available prediction methods by pressing the *"Continue"* button highlighted in the third area.

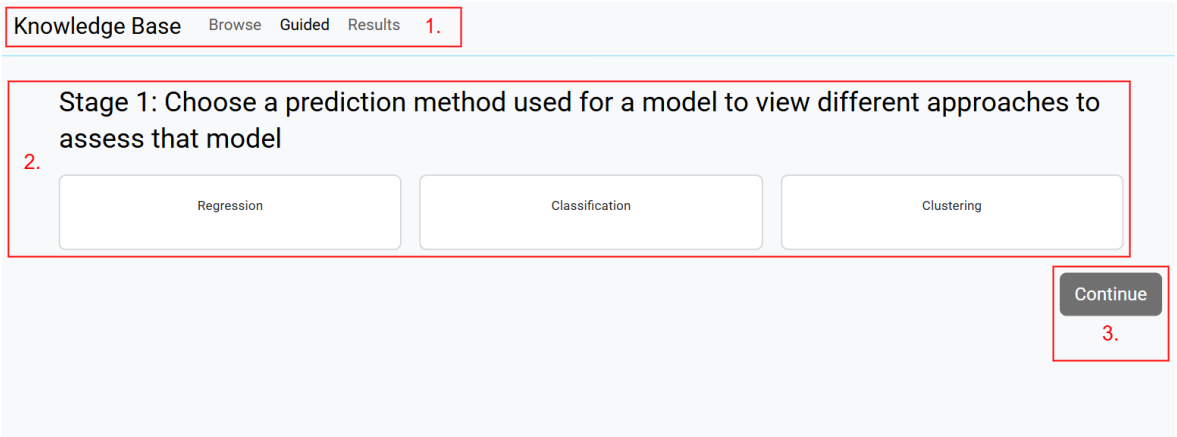


Figure 5.1: Guided Section - Stage 1

5.1.2 Guided Section - Stage 2

After selecting one of the available prediction methods the knowledge base redirects the user to the next stage of the **Guided Section**. The page of stage 2 features an overview of all Assessment Approaches relating to the previously chosen prediction method as well as an icon in the bottom right corner of the page and is highlighted in Figure 5.2.
The first area features a textual explanation telling the user what they can do on this page. The button in the second area enables new Assessment Approaches to be added to the knowledge base. The icon in the third area hides the available Filter Options in a side bar until it is clicked. At first, a detailed look at the first area will be provided in the following. Each rectangle holds information about one Assessment Approach for the selected prediction method; the bold text at the top is the name of such an approach. Within each card, zero or more elements may be included.

Guided Section - Stage 2: Expanded Methodology

Each element contains information about a specific *Methodology*, which is modeled as a subclass of a particular *AssessmentApproach* and is divided into three areas, shown in Figure 5.3. The text

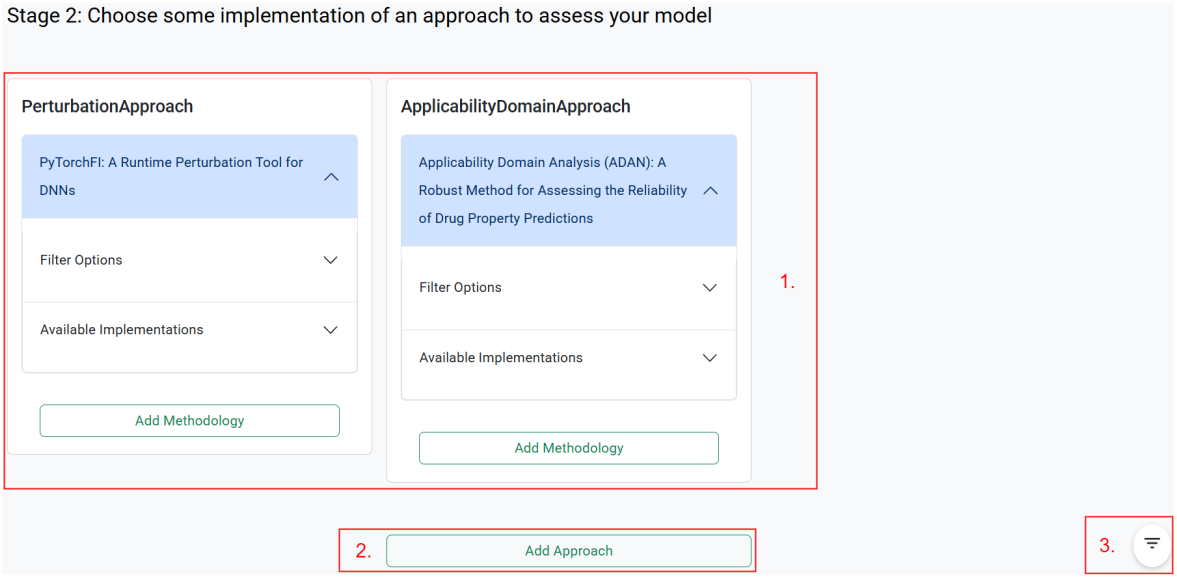


Figure 5.2: Guided Section - Stage 2

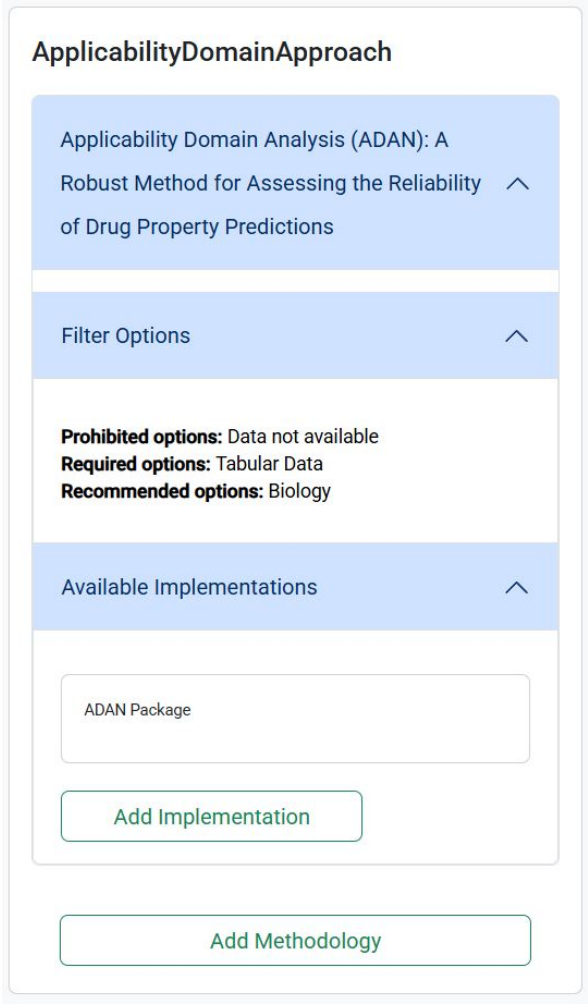


Figure 5.3: Guided Section - Example of expanded ADAN Methodology

displayed in the blue header area corresponds to the title of the methodology as referenced in the literature. By clicking on this title, the user is redirected to the external resource from which the methodology originates. For example, if the Methodology is based on a published approach such as the Applicability Domain Analysis (ADAN), clicking the title will open the corresponding

DOI link of the paper, thereby giving the user direct access to the original publication. The second area, highlighted by the blue text "Filter Options", contains a collapsed item that holds information about the associated *FilterOptions*. This section does not only display the filter options relevant to the selected *Methodology*, but also provides a collection of filter options that apply to all related elements, namely the *AssessmentApproach*, the *Methodology*, and, where available, the *Implementations*. By clicking on this area, the accordion expands and reveals the corresponding filter options in detail. Finally, the third area, highlighted by the blue text "Available implementations", presents information on available *Implementations*. Users can access the details of the implementation by expanding the areas through a click on the label. If a concrete Implementation exists for a given theoretical *Methodology*, it is listed in this section. Furthermore, the button *"Add Implementation"* allows the user to add a new Implementation to a given Methodology. Finally, on the bottom of each Assessment Approach *card* is a button *"Add Methodology"*. By the click of this button the user is redirected to a new page where information about a new Methodology for a particular Assessment Approach can be entered and subsequently added to the knowledge base.

Guided Section - Stage 2: Filtering

Area 3. of Figure 5.2 features the icon to expand a side bar holding the information about all available Filters and Filter Options. The side bar is highlighted in Figure 5.4.
The first element displayed in the sidebar is the label *"Filter"*, which indicates the type of information contained within this area. Directly below, the highlighted button *"Add Filter"* enables the user to navigate to a dedicated page for creating and storing a new filter in the knowledge base. Following this, a set of expandable elements is presented. Each element corresponds to a specific *Filter*, which is identified by the label shown on the element. By default, all elements are collapsed but can be expanded individually by clicking on them. Expanding an element reveals the *FilterOptions* associated with the respective filter. Each filter option is displayed together with a checkbox, allowing the user to actively select one or more options for further use within the knowledge base. In the figure above, the filter *"Data Type"* has already been expanded to illustrate the contents of the *"Data Type"* element and the filter option *"Tabular Data"* has already been selected. Furthermore, within each expandable element, an additional button *"Add Filter Option"* is available. This button provides the functionality to add new filter options to the selected filter, thereby extending the knowledge base with more fine-grained information. The last element in the sidebar is the *"Filter"* button. Clicking it, filters the displayed Assessment Approaches in Figure 5.2 by the selected Filter Options. For a result to show up, the filter has to match across all connected levels: the *Assessment Approach*, its *Methodologies*, and, if they exist, the *Implementations*. This means that the system does not only look at the approach on its own but also at the requirements and recommendations of the linked methods and tools. Only when the combined filter options line up with what the user has chosen the corresponding approaches will be selected for display. The following Figure 5.5 shows the results after filtering when the filter options *"Tabular Data"* from filter *"Data Type"* and *"Biology"* from filter *"Domain"* are selected.

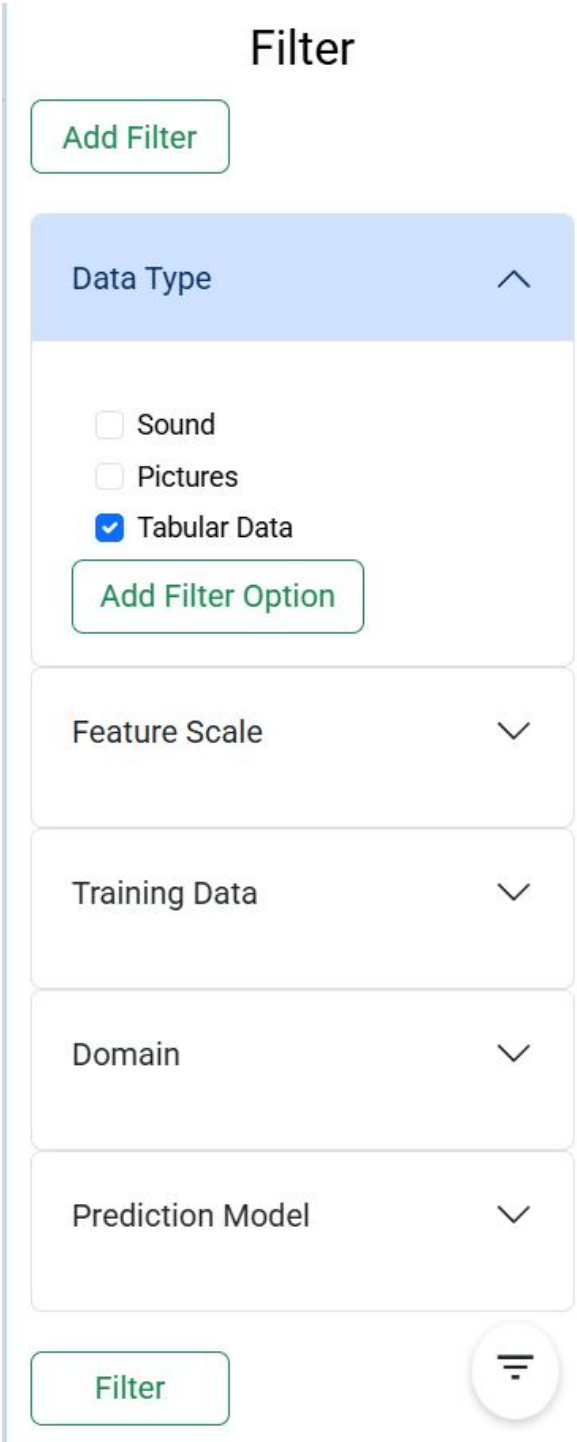


Figure 5.4: Guided Section - Filter Sidebar

5.1.3 Guided Section - Stage 3

The next step, is for the user to select a suitable implementation to assess the reliability of predictions of a particular model. This can be done by expanding the *"Available Implementations"* area of the corresponding expandable element shown in Figure 5.3 and clicking on the implementation to be used. This redirects the user to the last stage of the **Guided Section**, highlighted in Figure 5.6. In this example, the choice was made to use the Perturbation Tool as an example to show the implementation details. The page begins with a short explanation of what the user can do in Stage 3 of the **Guided Section**, presented in the form of a heading at the top. Beneath this, a note informs the user that at this stage it is also possible to make use of a *Generic Tool*. The

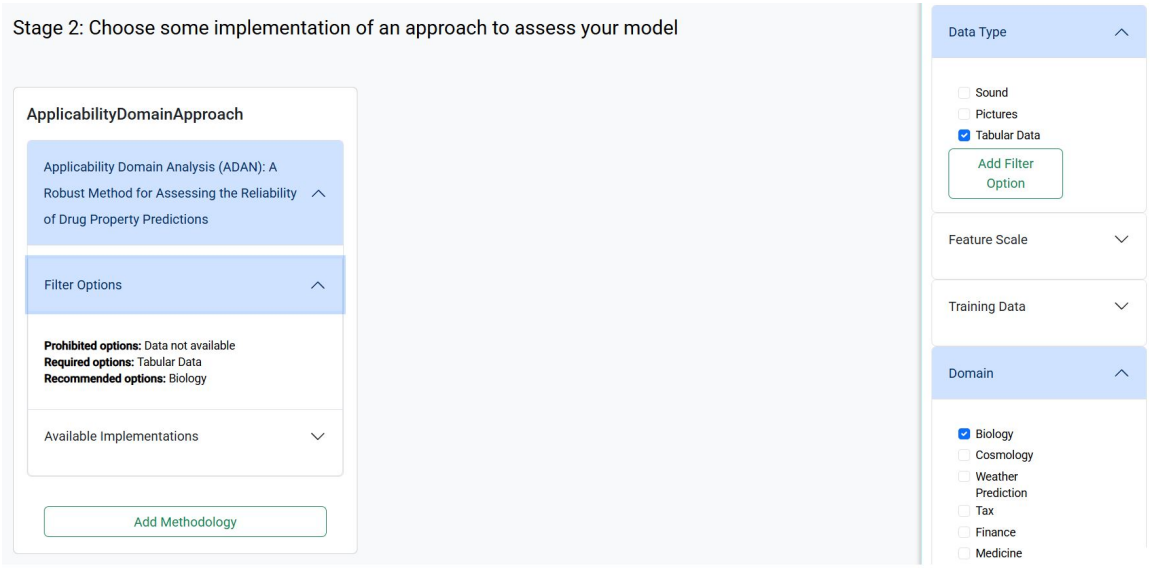


Figure 5.5: Guided Section - Result after Filtering

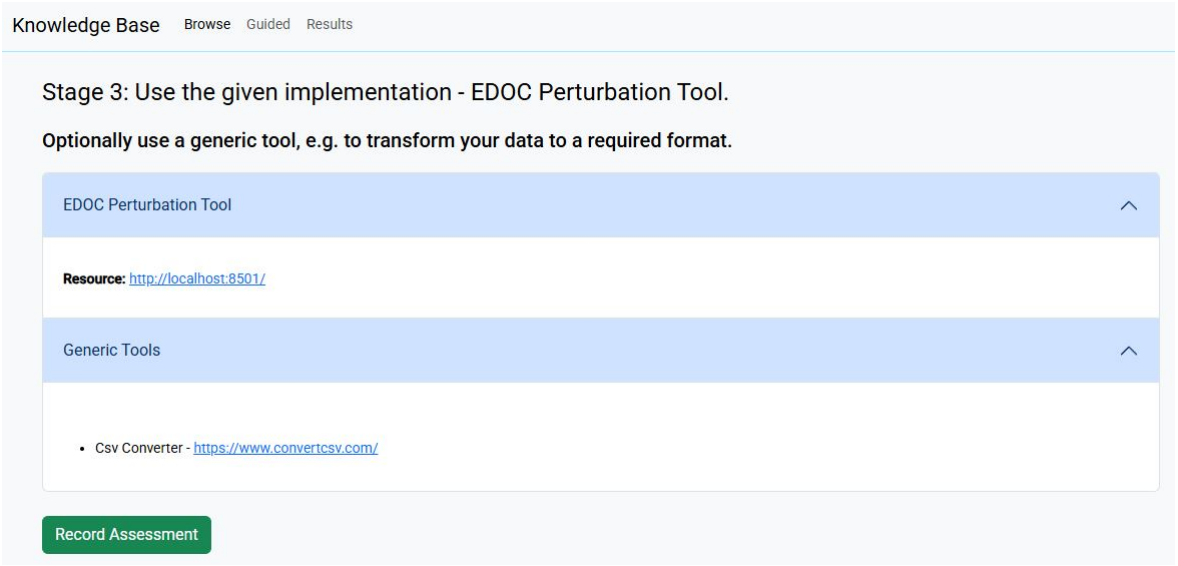


Figure 5.6: Guided Section - Stage 3

first expandable element represents the name of the tool. It is expanded by default, so the user can immediately see the available details. This section displays the resource where the tool can be found, for example a link to the project website or a repository, or, in the case of a deeply integrated tool, a link to execute it (such as a local endpoint on `localhost`).

Generic tools are meant to support the user, for example, they be used for data parsing, format conversion, or other preprocessing steps that prepare the input for the actual assessment. The knowledge base recommends suitable generic tools automatically. This is achieved through the *hasOption* property, which links a tool to specific filter options. Whenever an implementation has filter options that match those of a generic tool, the tool is suggested to the user as a potential addition for the given implementation. From the user perspective, these tools are displayed within the accordion area labeled *"Generic Tools"*. By default, this section is collapsed; users need to expand it in order to view the available recommendations.

Guided Section - Stage 3: Record Assessment

The page includes a button labeled *"Record Assessment"*, which allows the user to store assessments produced with a given tool directly in the knowledge base. At a deeper level of integration—for instance, in the case of the *Perturbation Tool*—this button not only records the results but can also be used to execute the tool itself. Execution is triggered by redirecting the user to the resource specified for the tool, which may take the form of a web link (e.g., a local endpoint on `localhost`) or another mechanism for starting the tool, such as an executable file. In case of a lower integration level this button has a different functionality. When the user clicks this button, for example when trying to record an assessment for the ADAN package, the modal view highlighted in Figure 5.7 will be opened. This dialog provides two input fields and a confirmation

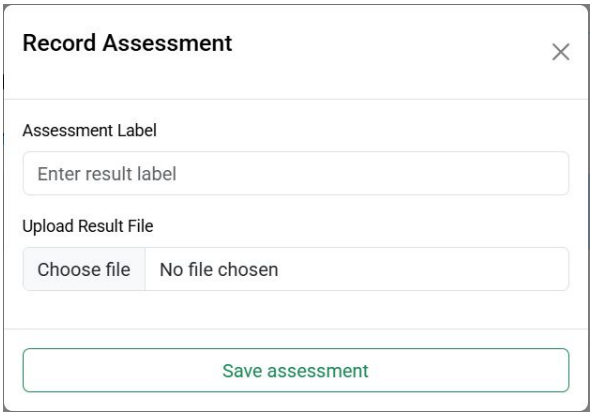


Figure 5.7: Guided Section - Record Assessment Modal

button. The first field, labeled *"Assessment Label"*, allows the user to assign a descriptive name to the result that is being stored. The second field, *"Upload Result File"*, enables the user to select a corresponding file from the local system, which will be attached to the assessment. A variety of file types are supported, including structured formats such as *CSV* and *XML*, as well as images, for example screenshots or plots of graphs that visualize results. At the bottom of the dialog, the button *"Save assessment"* finalizes the process and stores the information in the knowledge base. The dialog can also be closed without saving by clicking the close icon in the upper right corner. These stored results are later displayed in the *Results Section*, where users can review them in detail, preview attached files, and trace them back to the methodology or implementation that produced them. The functionality of the **Results Section** will be described in depth in the corresponding chapter of this thesis.

5.1.4 Add Filter and Filter Options

Next, this chapter continues by highlighting the two pages for adding a Filter and adding Filter Options to that Filter. Accordingly, Figure 5.8 shows the page the knowledge base redirects users after clicking the *"Add Filter"* button in the sidebar shown in Figure 5.4. In area 1. the label, more precisely the name of the new filter must be entered, as it is stated in the small tool tip below the input. On all of these types of pages throughout the knowledge base, tool tips such as the ones from the figure are there to help the user get an understanding on the type of information necessary to add elements to the knowledge base. In this particular case the example of adding the filter *"Feature Scale"* was chosen. In area 2. the user has the option to add filter options to the new filter, right away by clicking the *"Add new Filter Option"* button. This click displays the

option to add one new filter option indicated, for example, by the heading *"Add Filter Option 1"* and can be clicked any number of times. In this case the filter options *"nominal"* and *"ordinal"* have been chosen as examples and will be added to the knowledge graph alongside the new filter. This is achieved by clicking the button *"Add Filter"* at the bottom of the highlighted page. If a new filter is added this way, the knowledge base redirects the user back from the page they came from, in this example to the overview of assessment approaches and the new filter can immediately be selected from within the sidebar. This is standard practice within the knowledge base and all buttons of type *"submit"* reroute the user to the page from which they were redirected.

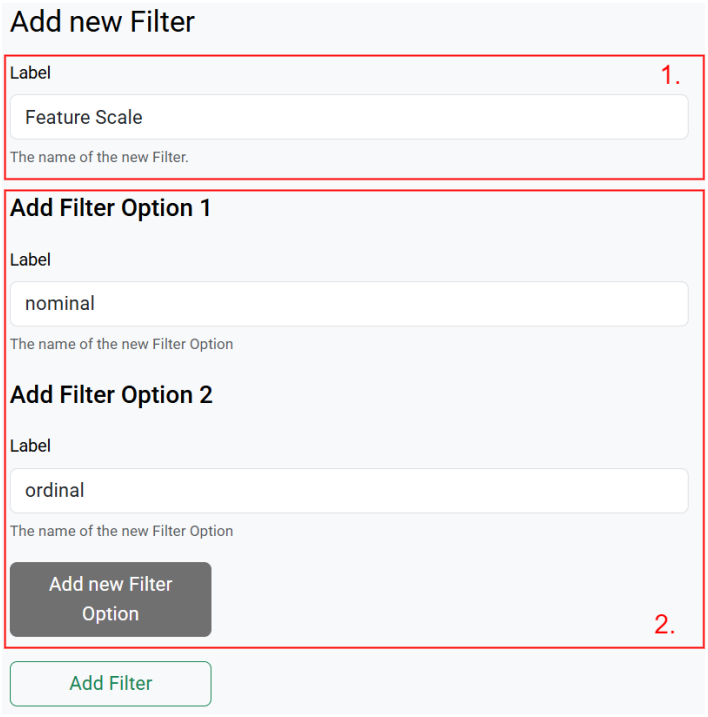


Figure 5.8: Add new Filter

Figure 5.9 highlights the page that gets shown when the *"Add Filter Option"* is clicked from within a given filter. The same principal to add filter options to a filter as previously examined applies in the highlighted page. The button *"Add new Filter Option"* can be clicked any number of times to add the according amount of filter options to a filter. In this case the filter option *"cardinal"* will be added to the filter *"Feature Scale"*. This is indicated by the heading on top of the highlighted page, where the label states to which filter new options are added. A backend request will be made by submitting this form through a click of the button *"Add Filter Option"*.

5.1.5 Add Assessment Approach

This chapter now introduces the way on how the user can extend the knowledge base even further, by adding the elements of the three abstraction levels, generic, method-specific and problem-specific, namely, assessment approaches, methodologies and implementations respectively. Figure 5.10 shows the page the user gets redirected to after clicking the button *"Add Approach"* highlighted in area 2. of Figure 5.2.
The user has the option to input three values when adding a new assessment approach to the knowledge base. In the input shown as *"Label"* the name of the newly added assessment approach has to be entered. The next input, labeled *"Comment"*, represents a field for providing a short

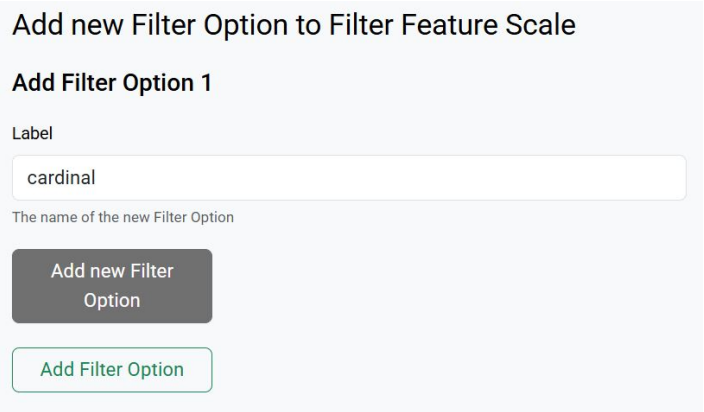


Figure 5.9: Add new Filter Option

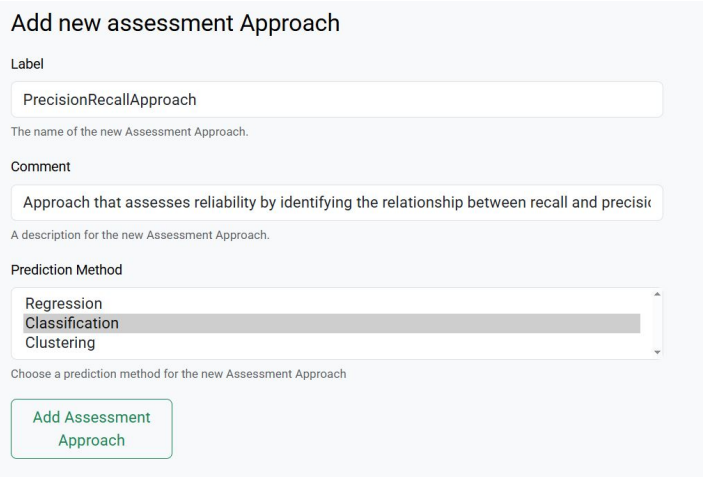


Figure 5.10: Add new Assessment Approach

description on how reliability can be assessed with the newly added approach. Thirdly, the bootstrap element *"select"* provides a way for the user to assign a prediction method to the new assessment approach, and it contains all the prediction methods already stored in the knowledge base. In this example the *"Precision Recall Approach"* for the *"Classification"* prediction method will be added to the knowledge base. By clicking the button *"Add Assessment Approach"* the backend will receive a request and store the entered information in the knowledge graph. Furthermore, the user will be redirected to the page from where the page shown was opened. Figure 5.11 shows the newly added assessment approach in the **Guided Section**.

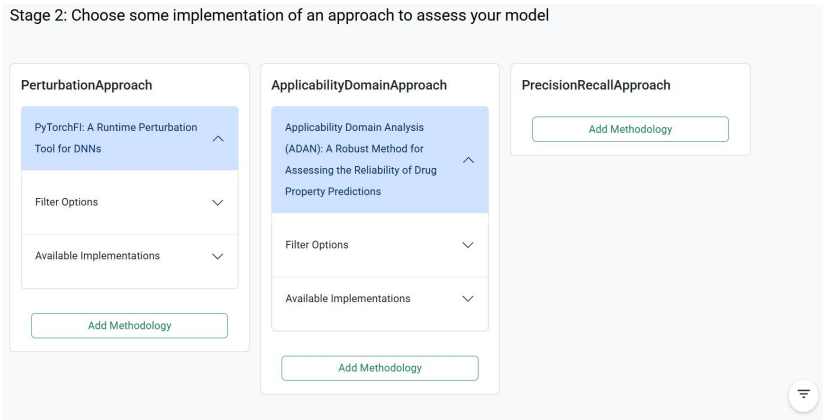


Figure 5.11: Guided Section - Result after adding a new approach

5.1.6 Add Methodology

The page in Figure 5.12 highlights the page to add a new methodology. By clicking the button *"Add Methodology"* the user gets redirected to a page with a heading at the top indicating the assessment approach the new methodology will be added to. Area 1. features the input fields to enter the relevant data needed to add a new Methodology. The input field *"Label"* stands for the title of the newly added methodology, in this example the reference process paper will be added to the knowledge base. The input below that, labeled *"Resource"* holds information about the external resource from which the methodology originates, in this case corresponding to the DOI link of the reference process paper. Next, the three select elements shown enable the user to add required, recommended and prohibited filter options to the new methodology right away. These selects feature multiple choice as indicated by the small description label below each element.

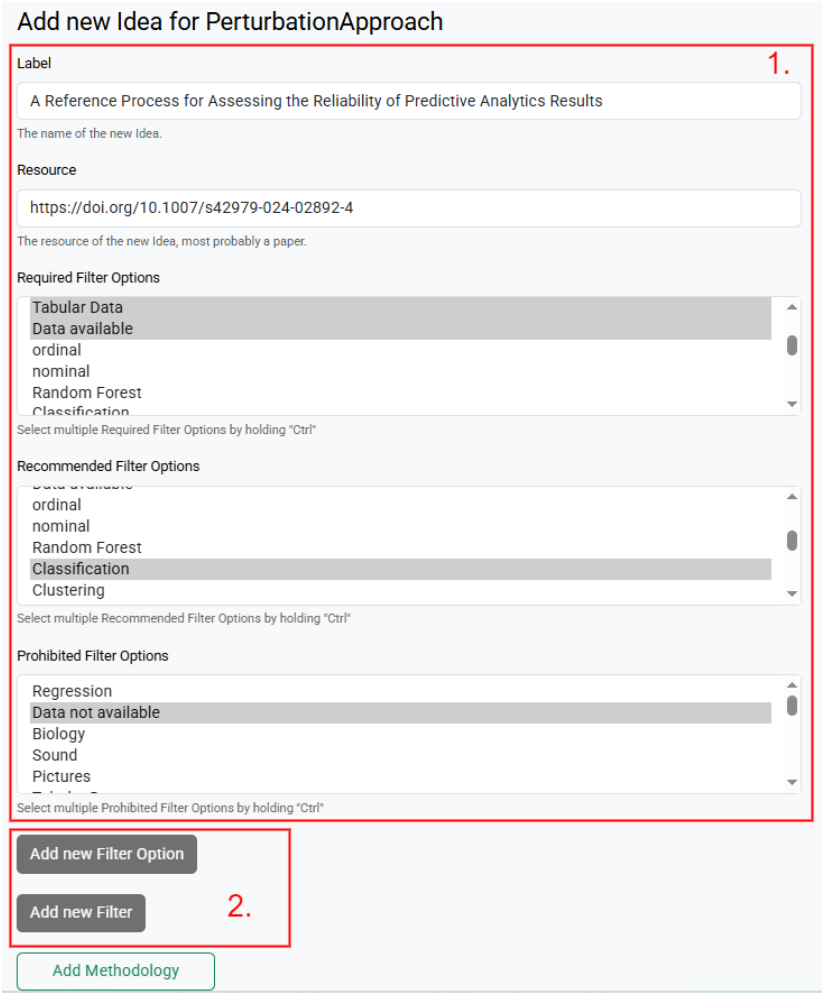


Figure 5.12: Add new Methodology

There is currently no way within the knowledge base to add additional filter options to a methodology after it was created, so the user has to make the choice of corresponding filter options while adding the new Methodology. To account for this, a way to add filter options that do not exist when adding a new methodology is provided in area 2. Figure 5.13 shows an excerpt of the page where the hidden elements for adding a new filter option and a new filter are expanded. Areas 1. and 2. look slightly similar to the pages for adding a new filter and filter option previously highlighted. However, this section of the page has more features. Lets first take a look at area 1. Here the user can input the name of a new filter option to be added to the methodology currently being added,

indicated by the heading *"Label"*. Next, the select element labeled *"Filter"* holds information about all available filters across the knowledge base, from which the user can select one filter to correspond to the new filter option. Lastly a type for the filter option can be chosen, specifically if the new filter option should be recommended, required or prohibited for the methodology. The description label of the *"Filter"* select element states, that if there is no filter available in the knowledge base that fits for the filter option, a new one can be created by clicking *"Add new Filter"*. In Area 2. it is highlighted how to do so. In the label input field, the name of the new filter needs to be entered. The second element features another select input field. In this select all **new** filter options that have been created within this page are visible, as long as a filter was not already selected for any particular filter option. In the example given in Figure 5.13, a new filter option *"unstructured"* is about to be created and it is of type prohibited for the methodology. The corresponding filter for this filter option is *"Data Type"* but the filter was not selected in this example as to show the functionality of the select element in area 2, where filter options without a filter are displayed. In the end, the filter option *"unstructured"*, with corresponding filter *"Data Type"* and type *"prohibited"* will be added to the methodology.

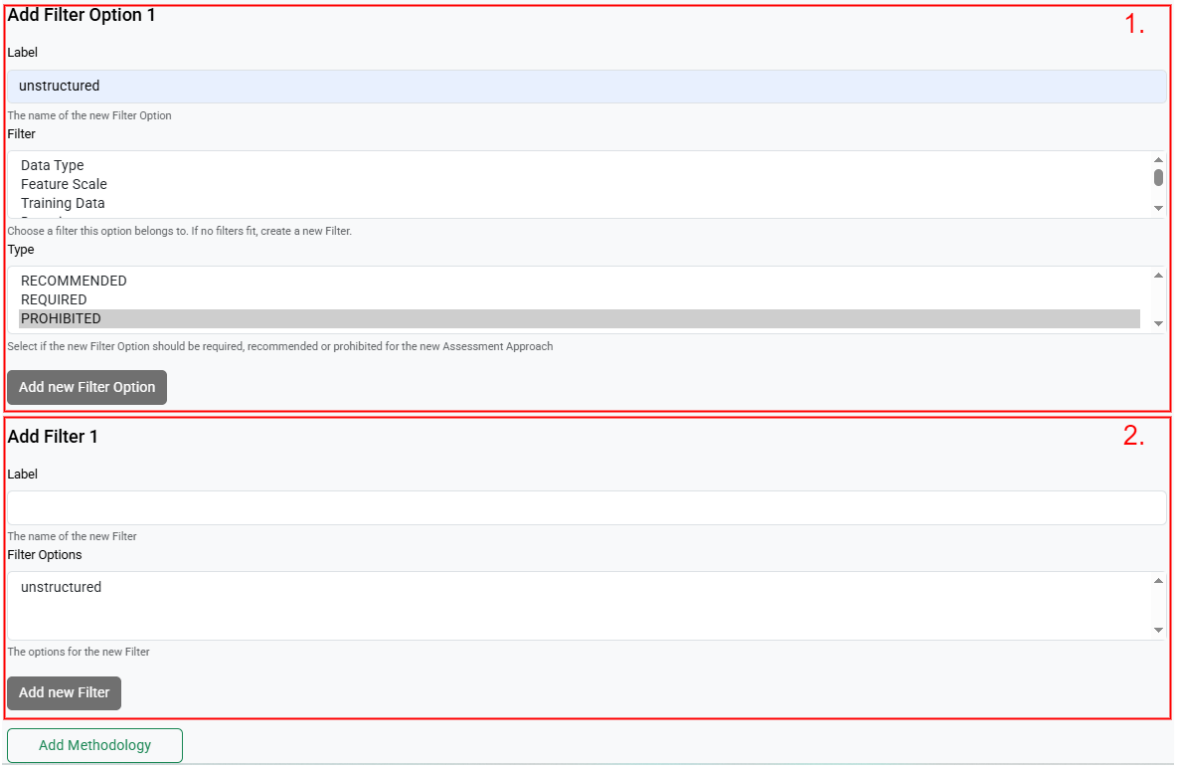


Figure 5.13: Add new Methodology - Filter Options and Filter

After clicking the button *"Add Methodology"* the entered information will be sent to the backend where everything will be persisted to the knowledge graph. Afterwards the new methodology can be viewed in the **Guided Section** within the *"PerturbationApproach"* card.

5.1.7 Add Implementation

The user can add a new implementation to an existing methodology by clicking the button *"Add Implementation"* shown in Figure 5.3. By doing so, the knowledge base redirects the user to the page highlighted in Figure 5.14. This page is structured similarly to the previously discussed page to add a new methodology. Again, the label and a comment can be entered to provide

information about the title of the implementation and a short description of the implementation respectively. In addition, the user can specify the type of the implementation, by selecting one of the two options available. An example for a script would be the previously mentioned ADAN package, where as an example for a tool would be the Perturbation Tool that is added in this example. Afterwards, similar like when adding a new methodology, a resource can be specified. In this case the resource represents a local endpoint on `"localhost"`, indicating that the tool can be executed from within the knowledge base itself, provided the user has access to such a tool. Other examples of resources could be links to a git repository or a link to a documentation and download of a package, such as https://phi.upf.edu/adan/ for the ADAN package. Furthermore, required, recommended, and prohibited filter options can be specified for the new implementation. Like before, new filter options can not be added to an already existing implementation so the user has the option to add new filter options and filters via the buttons *"Add new Filter Option"* and *"Add new Filter"* respectively.

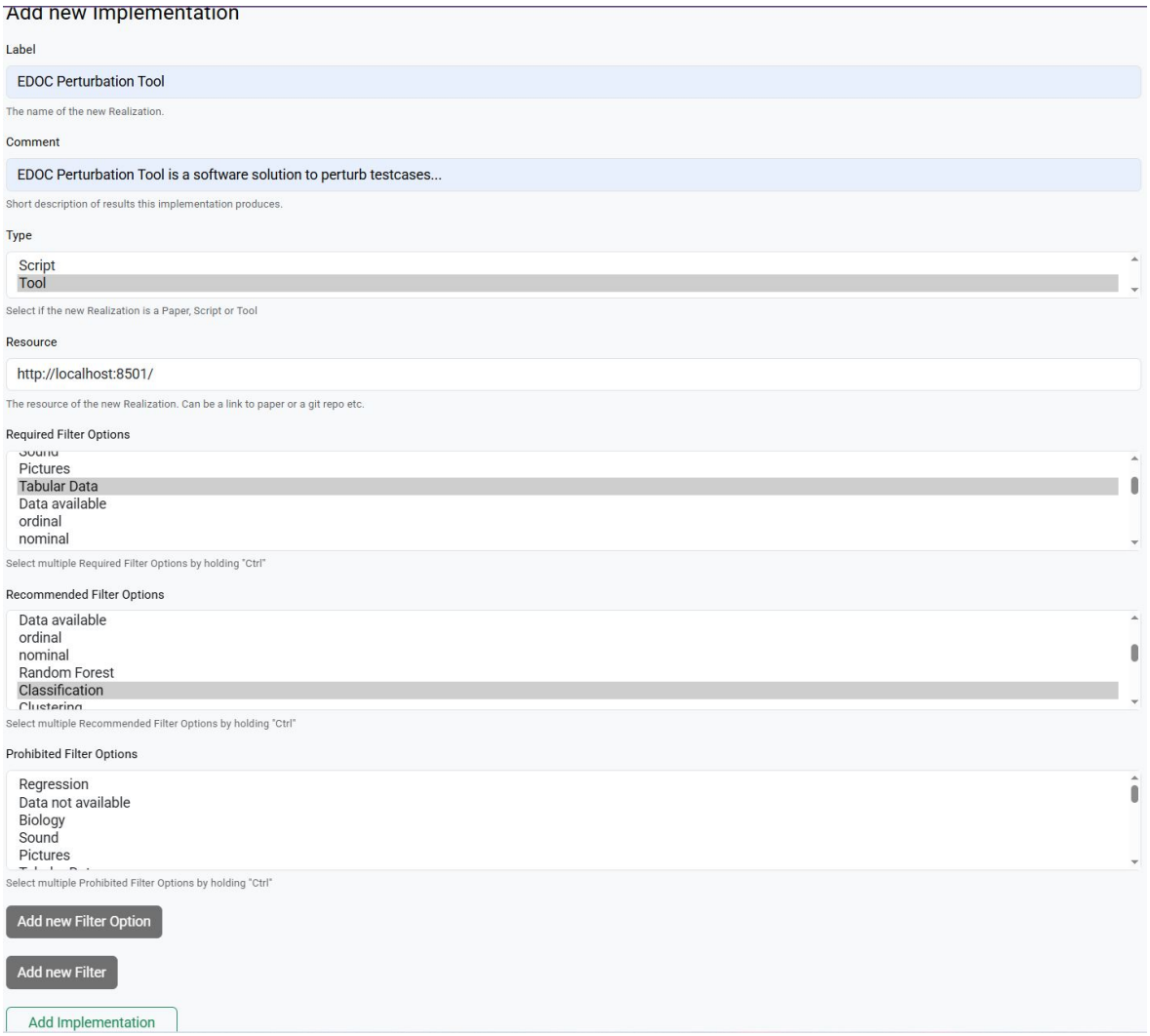


Figure 5.14: Guided Section - Result after adding a new methodology

After clicking the button *"Add Implementation"* a request gets sent to the backend where the entered information will be stored in the knowledge graph. The user then gets redirected to the **Guided Section** again where the new implementation can be viewed by expanding the *"Available Implementations"* area of the corresponding accordion element.

5.1.8 Guided Section - Conclusion

This concludes the description of the **Guided Section**. It has been shown how prediction methods serve as the entry point, how users are directed to suitable assessment approaches with their associated methodologies and implementations, how additional elements such as generic tools can be included, and how the user can extend the knowledge base by adding elements along the three abstraction levels. Furthermore it has been shown how to filter results based on the available filter options and how to extend the knowledge base with new filters and filter options. Finally, the **Guided Section** provides the means to record assessments, thereby linking the theoretical elements of the knowledge base with concrete results.

5.2 Browsing Section

The **Guided Section** was designed to help the user navigate the knowledge base step by step. The **Browsing Section**, in contrast, works differently. Here the system shows all available elements of the knowledge base in one place. These include *Assessment Approaches*, their *Methodologies*, and also *Implementations*. Instead of being guided through a sequence, users can simply browse freely, expand what interests them, and use filters to narrow down the view.
This part of the system is meant for exploration. It makes it possible to compare different elements like Assessment Approaches, Methodologies, or Implementations directly, to check details of a single approach, or to review newly added elements without having to go through the guided path first. In that sense, the **Browsing Section** complements the **Guided Section** but is less restrictive and leaves the user more freedom.

5.2.1 Browsing Section - Overview

The presentation starts with highlighting an overview of the **Browsing Section** shown in Figure 5.15. This overview is split into three areas. The first area shows a navigation side bar for browsing that highlights the elements currently in focus. The second area represents a search bar where users can enter search strings to narrow down the results shown within the third area. The third area shows a hierarchical visualization of all elements contained within the knowledge base, namely, *"Prediction Methods"*, *"Assessment Approaches"*, *"Methodologies"*, *"Implementations"*, *"Tools"*, *"Packages"*, and lastly *"Generic tools"*. At the top of the **Browsing Section** page the accustomed navigation bar, available on every page of the knowledge base, can be seen. As previously explained it highlights the section that is currently displayed. The first area located on the left-hand side of the page is a sidebar for navigation. It highlights the category of elements that are in focus in the main area of the **Browsing Section**. The highlight changes when the user scrolls down to view the remaining elements. Additionally, with a click on one of these categories the systems automatically scrolls down to where the elements of the category are located.
The second area features a search bar and is positioned at the top of the interface. Its functionality allows users to enter the names or parts thereof to locate elements within the browsing main area. Such a feature becomes more relevant as the knowledge base is extended and the number of elements increases.
Lastly, the third main area acts as the main content display for the **Browsing Section**. The actual elements of the categories highlighted in the side bar are displayed. Each main category is

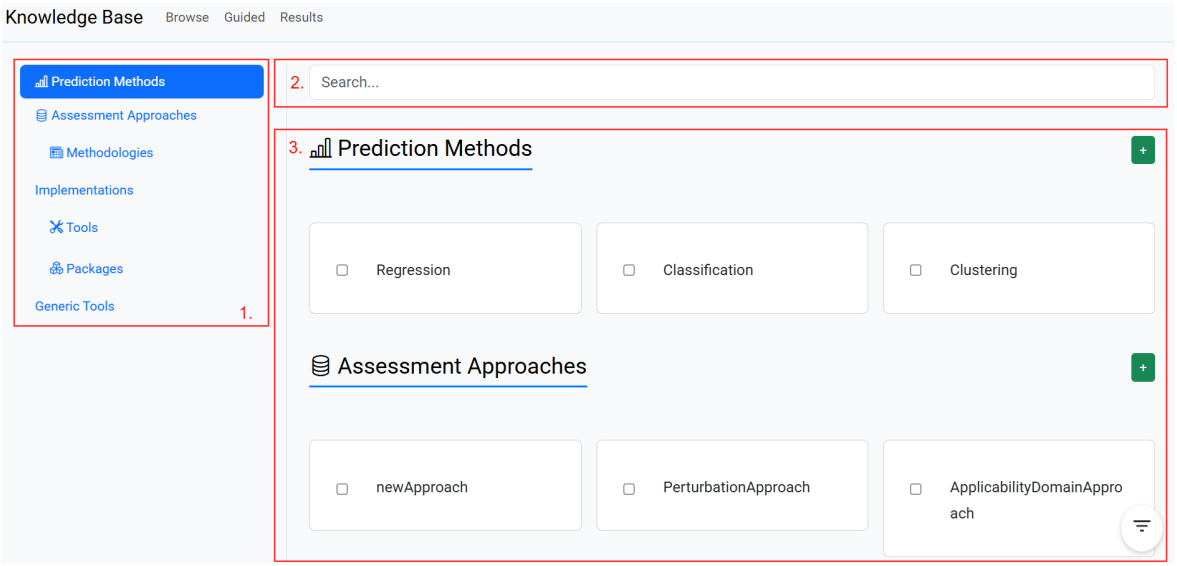


Figure 5.15: Browsing Section - Overview

displayed by an icon next to a label such as *"Prediction Methods"* and underlined by a blue line. The actual elements of a category are represented as card elements with a label. Each element in the prediction method category additionally features a checkbox next to its label indicating a selection feature. This selection will be explained in detail later, but acts as an additional way of managing the amount of elements displayed in this area. Furthermore, on the right-hand side of each category's label, a green "+" button provides the option to add new entries to the respective category.

It is important to note that the relationship between the sidebar and the main content area is not implemented as a strict hierarchy. Instead, both areas are designed to complement each other without restricting the user to a purely top-down exploration of the elements. The sidebar provides entry points by category, while the main area can present several types of elements at once, as shown in Figure 5.15. This design decision was made to avoid limiting the user to a strict hierarchical order, which could interfere with the desired free exploration.

The next two figures show the remaining elements of the knowledge base. Figure 5.16 depicts the remaining assessment approach elements that were missing from the previous figure as well as the methodology elements contained within the knowledge base. Figure 5.17 highlights the included implementations by grouping them into their respective subcategories *"Tools"* and *"Packages"*. Furthermore, it shows the generic tools available to the user. In addition to that, it can be seen that the sidebar updates its highlighting accordingly, depending on where the focus lies in the main area. By clicking on an element in the main area, the user will get redirected to the corresponding resource, similar to the **Guided Section**, when the user clicked on any given accordion element.

Another important distinction that becomes clear from the two figures above is that in contrast to the **Guided Section**, elements are not presented in a strictly hierarchical order. For example, *Methodologies* are listed below *Assessment Approaches*, but they are not displayed in a way that directly visualizes their relation to a specific approach. Similarly, *Implementations* are shown as *Tools* and *Packages*, yet they are not embedded directly under the methodologies to which they belong. This design decision was intentional: the **Browsing Section** is not meant to reproduce the ontology structure but instead to provide users with a way to freely explore the knowledge base. By avoiding rigid hierarchies, users are able to view and compare all available elements side

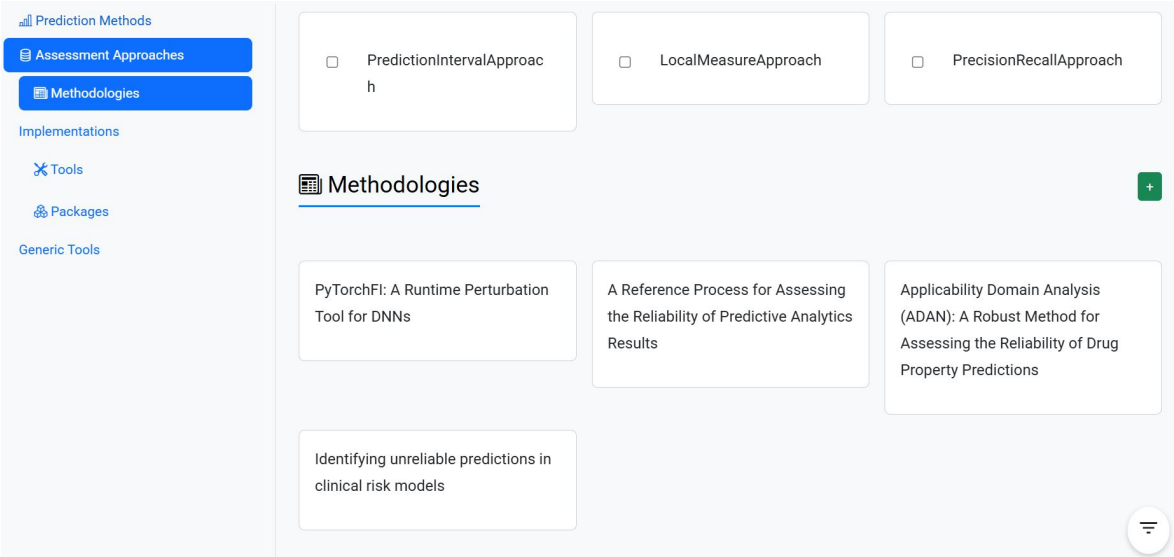


Figure 5.16: Browsing Section - Methodologies

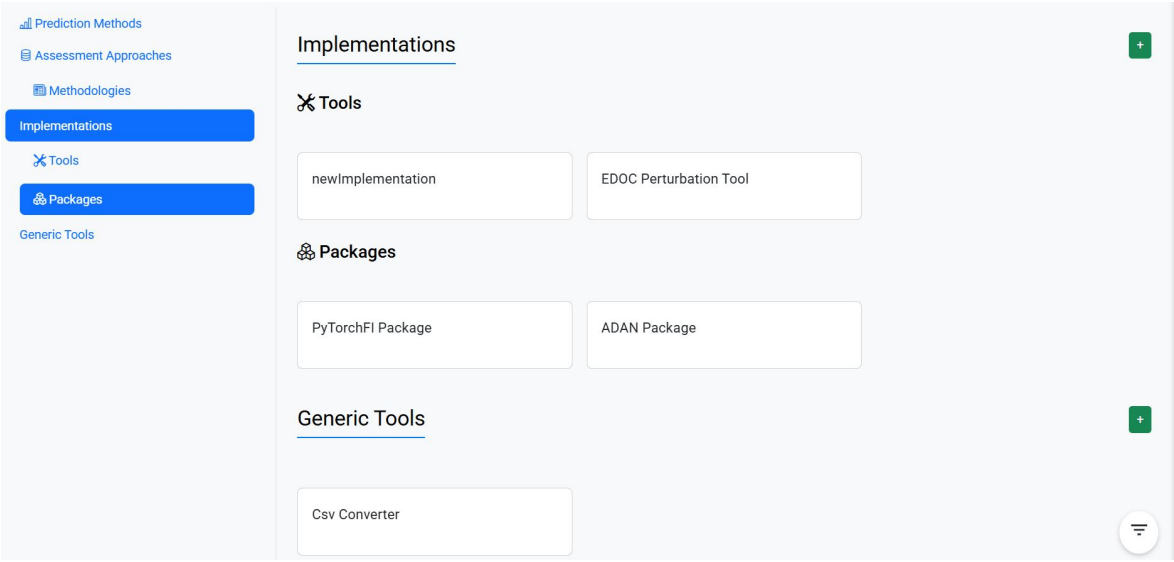


Figure 5.17: Browsing Section - Implementations and Generic Tools

by side, without being constrained to follow the relational structure of the underlying graph. This ensures that the **Browsing Section** functions as a flexible exploration tool that complements the more structured and sequential logic of the **Guided Section**.

5.2.2 Interacting with the Browsing Section

The focus of this chapter lies on the previously mentioned ways to of managing the amounts of elements shown in the **Browsing Section**. First, are the filter options that are also included in the **Guided Section**. Similarly to the **Guided Section**, in the bottom right corner of the **Browsing Section** an icon is located that hides the filter side menu before it is clicked. When expanded, the side menu from Figure 5.4 is shown. The filtering works just like described previously in the **Guided Section**, where the backend retrieves all suitable assessment approaches and their underlying methodologies and implementations. For example if the filter option *"Tabular Data"* is selected and the *"Filter"* button is clicked, only the assessment approaches with that filter option will be displayed, as highlighted in Figure 5.18

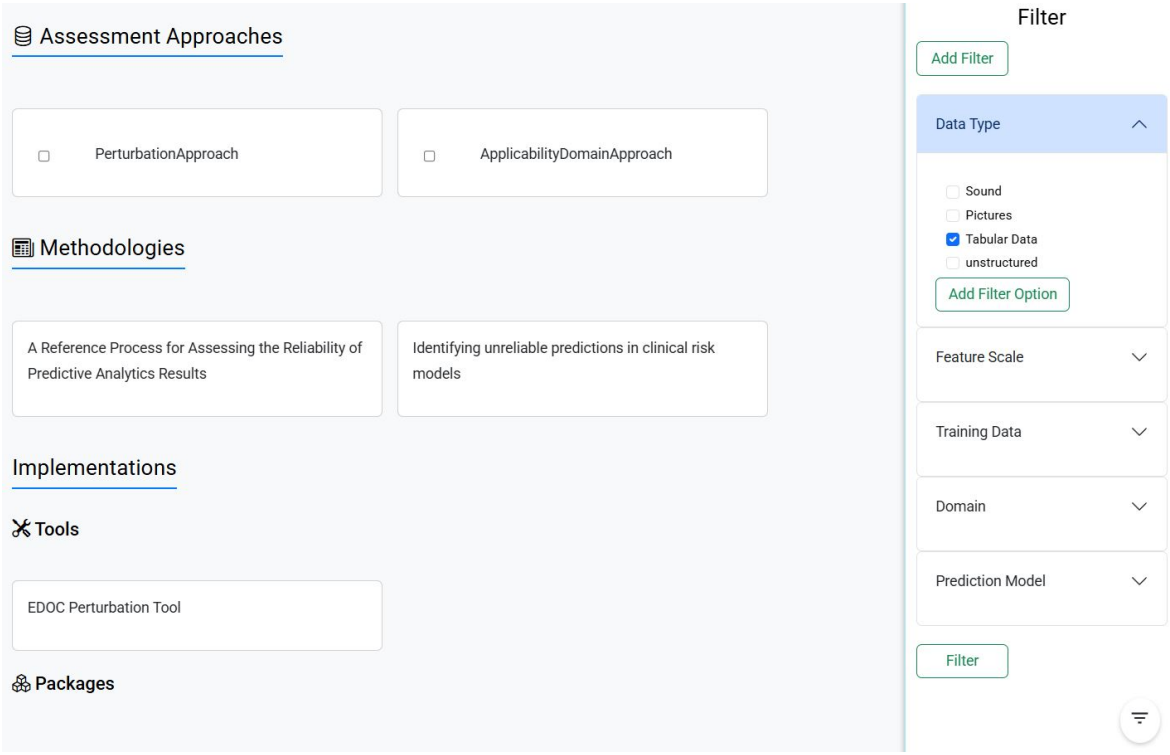


Figure 5.18: Browsing Section - Results after filtering

The second way of narrowing down data in the **Browsing Section** is by selecting one or more prediction methods via the checkbox provided next to the labels. Initially, no method is selected implying that all elements will be shown in the main area. If a prediction method is selected, all assessment approaches that work for this prediction method will be displayed as available, all other will be displayed as disabled. If multiple prediction methods are selected, all approaches that work for all the prediction methods will be displayed as available, all other will be displayed as disabled. All sub elements of all the working approaches will also be either enabled or disabled depending on the selected prediction method(s). If prediction methods get unselected the corresponding approaches and all the sub elements of the approach will be displayed as disabled until no prediction method is selected anymore, then everything will be displayed as enabled once again. Figure 5.19 shows a practical example of the above explanation. The figure shows that the prediction methods *"Regression"* and *"Clustering"* are slected, indicated by the blue check mark next to the labels. Appropriately, all assessment approaches that do not work for the other prediction method *"Classification"* are displayed as disabled elements, indicated by them having a reduced opacity and that they are not clickable, meaning the elements can no longer be interacted with in any way. All other assessment approaches are still displayed as active as they match the selected prediction methods. In this case the *"LocalMeasureApproach"* and the *"PredictionIntervalApproach"* are still enabled. Subsequently, all related elements that correspond to either the enabled or disabled assessment approaches are also enabled and disabled respectively. In this case only one methodology remains enabled, as the knowledge base does not store any information about methodologies for the local measure approach and only one methodology for the prediction interval approach. The implementations are not shown in this figure, as all of them are disabled due to the fact that the only methodology still enabled does not instantiate any available implementations in the knowledge base at this time.

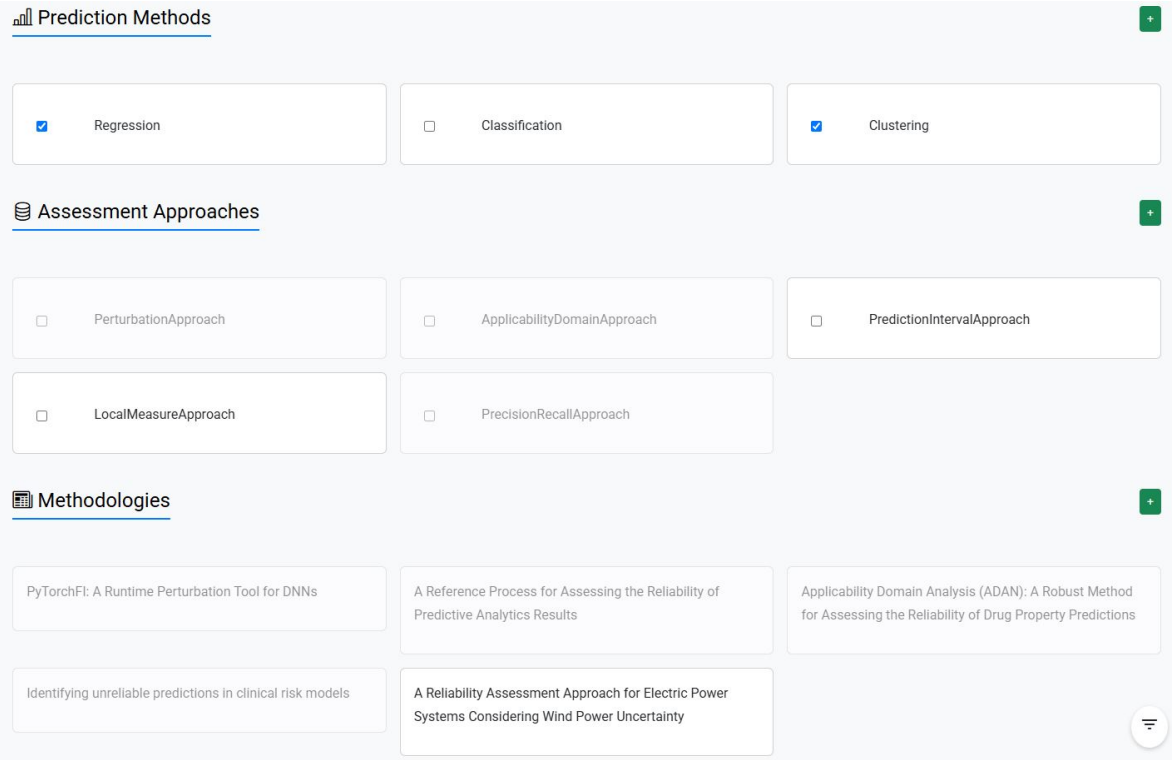


Figure 5.19: Browsing Section - Results after selecting prediction methods

5.2.3 Add new Elements from the Browsing Section

To conclude the explanations of this section, the last part will cover how to extend the knowledge base from within the **Browsing Section** by using the green "+" buttons next to the elements. The first two buttons are quickly explained as their functionality is identical to the *"Add Filter Option"* button to add new prediction methods, described in Chapter 5.1.4, in the filter sidebar and the *"Add Approach"* button to add new approaches already explained in Chapter 5.1.5. What slightly differs from the way, new methodologies and new implementations are added to the knowledge base in the **Guided Section**, is the use of an additional dialog to select the corresponding super element for these elements. More precisely, choosing an assessment approach to which to add a new methodology or choose a methodology to which to add a new implementation to.

Add new Elements from the Browsing Section - Methodologies

Figure 5.20 shows such an additional dialog when the user wants to add a new methodology from the **Browsing Section**. Since each methodology must be associated with an *Assessment Approach*, the dialog requires the user to first select the corresponding approach from a drop-down menu. Once an entry is chosen, the process can be continued by clicking the *"Continue"* button, which redirects the user to the page for entering the details of the new methodology. If the action is not intended to be completed, the dialog can be closed at any time by clicking the *"Cancel"* button or the close icon in the top right corner.

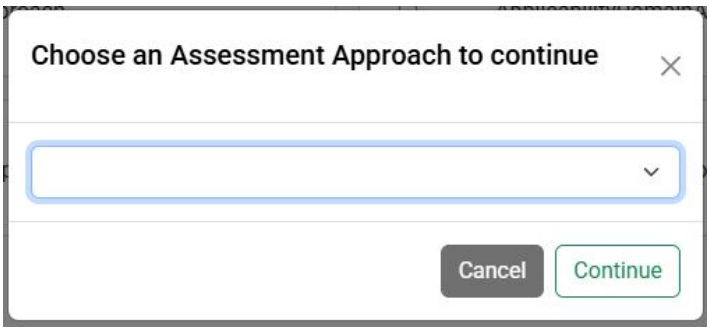


Figure 5.20: Browsing Section - Add Methodology: additional Dialog

Add new Elements from the Browsing Section - Implementations

Figure 5.21 depicts the dialog that is shown when a new *Implementation* is to be added to the knowledge base. Since each implementation must be linked to a specific *Methodology*, the dialog prompts the user to select the appropriate methodology from a drop-down menu before proceeding. After making a selection, the user can continue by pressing the *"Continue"* button, which opens the form for entering the details of the new implementation. If the process should not be completed, the dialog can be dismissed either by clicking the *"Cancel"* button or by closing the modal using the icon in the top right corner.

Figure 5.21: Browsing Section - Add Implementation: additional Dialog

Both *"Continue"* buttons route the user to the respective pages to add either a new methodology as highlighted in Figure 5.12 or a new implementation as highlighted in Figure 5.14. In both cases, submitting a new methodology or implementation within these pages, the user will get redirected to the **Browsing Section** again.

5.2.4 Browsing Section - Conclusion

In summary, the **Browsing Section** offers a complementary way of interacting with the knowledge base alongside the **Guided Section**. Whereas the **Guided Section** follows a sequential, step-by-step logic that highlights dependencies between prediction methods, assessment approaches, methodologies, and implementations, the **Browsing Section** is deliberately designed to provide more flexibility. By presenting elements side by side without enforcing hierarchical relations, users are free to explore the available content in whichever order suits their needs.
The sidebar navigation, search functionality, filtering and selection options, and open content area together form a consistent structure that supports both overview and detail-oriented exploration. In addition, modal dialogs for adding new elements ensure that dependencies are respected at the point of data entry, while still allowing the user to navigate the content in a non-restrictive

manner.
Through this design, the **Browsing Section** balances openness and structure: it grants users the freedom to inspect all stored elements at once while maintaining enough safeguards to preserve the integrity of the knowledge graph.

5.3 Result Section

The **Results Section** is the place where the outcomes of assessments are shown. Unlike the guided and **Browsing Section**s, which focus on exploring approaches, methodologies, and tools, this part of the system is about what those elements actually produce. In other words, it collects the results that were created when an implementation or tool was executed.
Each result is stored together with some basic information. This includes when it was created, which tool or package was used, and what resource file belongs to it. The files can have different formats, such as CSV tables or images, and in some cases they can be previewed directly inside the interface. By keeping all results in one section, the user can easily find them again, check details, and compare outcomes from different assessments.

5.3.1 Results Section - Results of a deeply integrated Tool

Lets start the presentation of the **Results Section** by showing how an actual assessment case from a deeply integrated tool, in this example the Perturbation Tool is represented in this section. Figure 5.22 shows the result of such an assessment case. On top of this page the navigation bar is once again shown, and it highlights that the results page is currently shown. In the main section of this page, the figure shows the heading of the page *"Results"*. As shown in the figure, every assessment result is represented as a card. The card header displays the label of the case (e.g., *Departure Delay Case*), while the metadata below specifies when the case was generated and completed, as well as which tool was used to produce it. If the user clicks on the tool button, the knowledge base will redirect them to the details page (highlighted in Figure 5.6 for the selected tool. A direct link to the resource file belonging to the case is also provided. This file can take different formats, such as *CSV* or *XML* datasets or image files (e.g., plots of results). If the uploaded file is in tabular format, the system additionally offers a preview of its contents directly within the card. In the example shown, the CSV preview displays the first few rows of the dataset, allowing users to inspect the recorded output without leaving the interface. The description highlighted below the actual results is taken directly from the description of an implementation, which the user can enter when adding an implementation to the knowledge base; this was shown in Figure 5.14.
To illustrate how the results of a deeply integrated tool are stored in the knowledge base, the following section uses the *Perturbation Tool* as an example. It shows how assessment cases created with this tool are recorded and later displayed in the **Results Section**. As described earlier, Figure 5.6 contains the *"Record Assessment"* button, which provides the option to store assessments in the knowledge base. For the *Perturbation Tool*, which represents a tool at the deep integration level, this button goes a step further. In this case, it is used to execute the tool by opening the `localhost` link specified in the resource property of the tool. A detailed explanation of how to perform perturbation with the Perturbation Tool is not part of this thesis. For that, reference is made to the work of (Badzura, 2023). Nevertheless, Figure 5.23 shows the final step of the deployment stage, where the perturbed test case is created and a download for a *CSV* file is

Figure 5.22: Results Section - Example assessment case retrieved from the Perturbation Tool

provided to the user. This file needs to be downloaded and stored in the upload folder provided in the backend in order to retrieve it and display a preview of the values inside the **Results Section**. Afterwards the knowledge base queries the assessment case directly from the knowledge graph of the Perturbation Tool, links the file to the selected assessment and finally displays the assessment case in the knowledge base. In addition to the assessment results shown in this section, the knowledge base is able to directly access the knowledge graph of the Perturbation Tool in order to retrieve the corresponding perturbation parameters and make them available within the interface of the knowledge base. Although this capability is supported by the underlying architecture, it is not part of the current implementation and is therefore left for future extension.

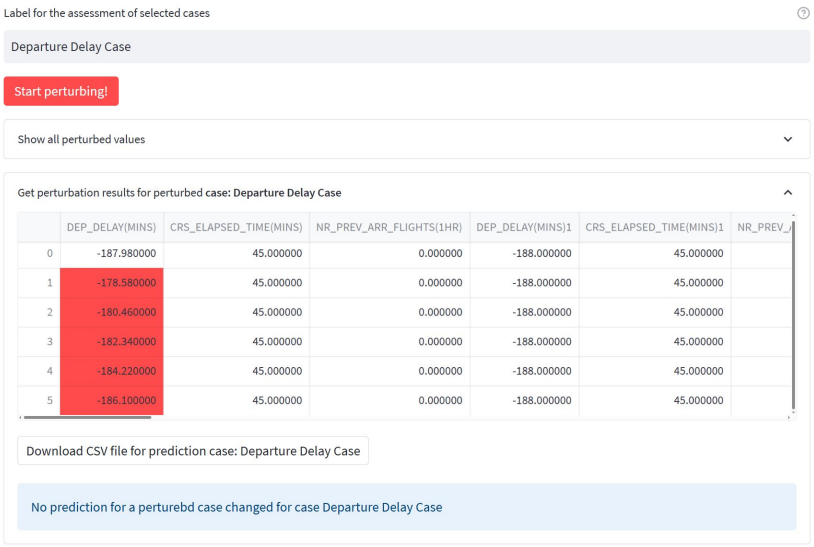


Figure 5.23: Results Section - Example assessment case created with the Perturbation Tool

5.3.2 Results Section - Results of all other Tools

For all other implementations that are not deeply integrated, like the ADAN package, the *"Record Assessment"* button opens the modal shown in Figure 5.7, where the user can enter a name for the newly recorded assessment as well as optionally provide a file of the generated results. The next Figure 5.24 shows two new results added to the knowledge base, generated by the ADAN package. The first case, called *Genome Prediction Case #1 – Adan*, has a CSV file attached as its resource. The system shows a small preview of this file directly in the interface. The preview lists the input values in three columns and the reliability category in a separate column named *adan_category*. This makes it easy to check the outcome without downloading the file first. The second case, *TestCaseAdanNew*, works differently. Here the attached resource is an image file. It contains a plot that visualizes the results of the package. Instead of showing tabular data, the system displays the figure directly in the card.
Both results contain the usual metadata such as when they were created and finished, and which package was used. Each case also comes with a short description of the ADAN Package, which helps to understand the context. These two examples make clear that the **Results Section** can handle different file types, such as CSV tables and image plots, and present them in a way that is easy for the user to view.

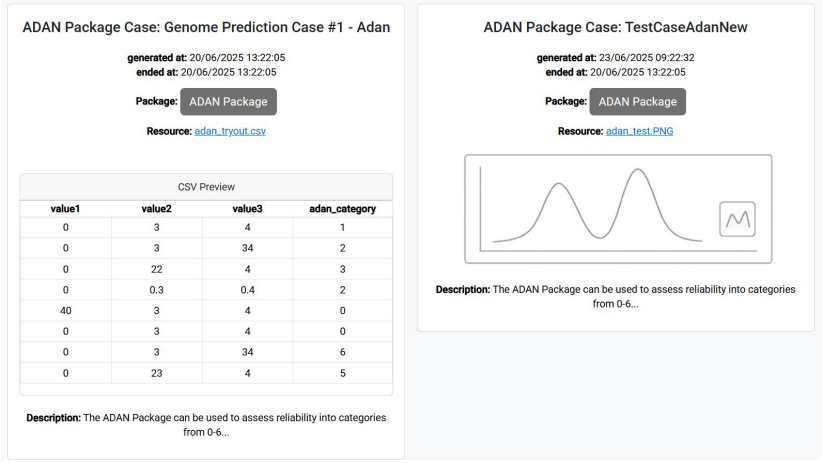


Figure 5.24: Results Section - Example assessment cases recorded for the ADAN package

5.3.3 Results Section - Conclusion

In conclusion, the **Results Section** brings together the outcomes of the recorded assessment cases. It stores the basic metadata, such as timestamps and the tool or package that was used, and it also links each case to its resource file. These files are made directly accessible to the user. Depending on the format, results can either be previewed in the interface, for example CSV tables or images, or downloaded and inspected outside the system. This combination makes the **Results Section** the place where abstract elements of the knowledge base meet their practical outputs. Approaches and methodologies, once implemented, lead to concrete results that can be stored, checked, and compared across different cases.
Taken together, the **Guided Section**, **Browsing Section**, and **Results Section** give a complete picture of how users can work with the system. The **Guided Section** provides a step-by-step workflow, the **Browsing Section** makes it possible to freely explore all elements, and the **Results Section** shows the outcomes of the recorded assessments. The screenshots and examples in

this chapter have shown how these parts of the interface work in practice and how they support different types of interaction. At the same time, the layout stays consistent across all sections. From the user's point of view, the knowledge base is not only a place to store information but also a system that makes the information usable and meaningful.

6

Implementation Perspective

While the previous chapter highlighted the knowledge base from the user perspective this chapter focuses on the underlying implementation of the knowledge base. The implementation perspective is structured around the three main sections of the knowledge base, namely the **Guided Section**, the **Browsing Section** and the **Results Section**. For each of those sections interesting implementation aspects from the backend, developed with *Java Spring Boot*, the persistence layer, handled by *Apache Jena Fuseki*, and the frontend, implemented in *Angular* with *TypeScript* and styled with *Bootstrap* and how they interact with each other will be discussed. Additionally, the structure and design of the underlying Knowledge Graph, as well as the systems architecture and implementation structure will be examined in this chapter.

6.1 Knowledge Graph

To store the data needed for the knowledge base and to convert these data into knowledge, some form of persistence for all kinds of data is necessary. The choice to use a knowledge graph as the persistence method for the knowledge base will be explained in detail in this section of the thesis. Furthermore, the explanation of the structure of the knowledge graph and its elements will be discussed in greater detail.

To help provide a system, in which users can find suitable assessment approaches to assess their prediction models results these approaches with all their sub-elements and elements that have a relationship with them, a flexible method of storing the data is essential. As previously discussed in the related work Section 3 of this thesis, the reference process proposed by Staudinger et al. (2024), especially its underlying knowledge graph are highly relevant in the context of such a system. The need to store elements described by the reference process in the knowledge base, including but not limited to business understanding entities, namely, assessment approaches is thereby an obvious one. Since many of the important elements needed for developing the knowledge base already exist in the reference process's knowledge graph, it makes sense to design a similar knowledge graph for this project as well. However, it was not completely ignored to use another form of persistence, such as a relational database system. Needless to say, the advantages of knowledge graphs being easily extensible when adding new elements to the graph or make changes to the structure of the data captured by the knowledge base outweighed other advantages provided by a relational database. Furthermore, a knowledge graph enables more accurate recommendations than a relational database, e.g. a collection of assessment approaches, by looking at the relations

between the various filter options stored as triples in the graph; a task with which a relational database may struggle. Ultimately, the decision to use a knowledge graph was made by considering all of the reasons outlined above, not to mention that the basis of this thesis is the reference process and its underlying knowledge graph.

6.1.1 Design of the KG

The knowledge graph is stored in the Apache Jena Fuseki store, that runs as standalone SPARQL Server. The elements in the knowledge graph are stored as RDF triples that can be retrieved from Fuseki via SPARQL queries. In the following all triples of the KG will be explained and short examples will be provided for a full understanding of the design.
As a starting point the entities directly taken from the reference process will be introduced. The namespace *rprov* was taken based on the reference process and is used for provenance information not contained in the PROV-O. Tables 6.1 and 6.2 show those entities in RDF format:

Table 6.1: BusinessUnderstandingEntity from the reference process

Subject	Predicate	Object
rprov:BusinessUnderstandingEntity	rdf:type	owl:Class
	rdfs:subClassOf	prov:Entity

Table 6.2: DeploymentEntity from the reference process

Subject	Predicate	Object
rprov:DeploymentEntity	rdf:type	owl:Class
	rdfs:subClassOf	prov:Entity

Each entity is an instance of an *owl:Class* and a subclass of an Entity defined by the PROV-O. In the following, the elements relevant to the **Browsing** and **Guided Section** of the knowledge base are introduced using a top-down approach. Accordingly, the with elements that are subclasses of *rprov:BusinessUnderstandingEntity* will be shown first. Subsequently, the elements instantiated by these classes are described, followed iteratively by their respective instances, and so forth. This explanation will show the elements on the three different abstraction levels generic, method-specific and problem-specific. But first, two instances of *owl:Class* vital to the workings of how the knowledge base recommends assessment approaches based on all different kinds of properties of practical data mining project need to be introduced; they are shown in Tables 6.3 and 6.4.

KG Elements: Browsing and Guided Section

Table 6.3: Filter

Subject	Predicate	Object
rprov:Filter	rdf:type	owl:Class

The *rprov:Filter* element represents a general category of filter, such as Prediction Method, and the *rprov:FilterOption* element represents an entry from that category, such as Classification.

Table 6.4: Filter Options

Subject	Predicate	Object
rprov:FilterOption	rdf:type	owl:Class

The relationship between these elements becomes clear from the next Table 6.5, which shows an *owl:ObjectProperty* connecting the elements by its *rdfs:range* and *rdfs:domain* attributes.

Table 6.5: Object property to connect Filter and Filter Options

Subject	Predicate	Object
rprov:hasOption	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:Filter
	rdfs:range	rprov:FilterOption

Now, as proposed earlier the following tables will show the elements on the different abstraction levels, starting with the generic element, *rprov:AssessmentApproach*.

Table 6.6: Generic element: rprov:AssessmentApproach

Subject	Predicate	Object
rprov:AssessmentApproach	rdf:type	owl:Class
	rdfs:subClassOf	rprov:BusinessUnderstandingEntity

As shown in Table 6.6 *rprov:AssessmentApproach* is a subclass of *rprov:BusinessUnderstandingEntity* and a instance of an *owl:Class* and represents a generic element of an assessment approach, that does, at the generic level, not include any information on how to use a specific kind of method for the approach. This kind of information is stored in the next element shown in Table 6.7:

Table 6.7: method-specific element: rprov:Methodology

Subject	Predicate	Object
rprov:Methodology	rdf:type	owl:Class
	rdfs:subClassOf	rprov:AssessmentApproach

The *rprov:Methodology* element is an instance of *owl:Class* and a subclass of the generic element *rprov:AssessmentApproach*. In it, information about theoretical ways to apply a given assessment approach for reliability assessment is stored and this information mostly comes from literature embedded in the knowledge base in the form of the Methodology element. At the method-specific abstraction level, where this element lies, all of the loosely integrated methods to apply an assessment approach are stored. The elements that hold information about the other two levels of integration can be seen in Tables 6.8, 6.9, 6.10 and 6.11 and are assigned to the problem-specific abstraction level.

As described in Table 6.8 *rprov:Implementation* is an *owl:Class* and it instantiates the *rprov:Methodology* element. Additionally, it acts as a superclass for the two types of implementations supported within the knowledge base, namely, *rprov:Package* and *rprov:Tool*, both of which are subclasses of

Table 6.8: problem-specific element: rprov:Implementation

Subject	Predicate	Object
rprov:Implementation	rdf:type	owl:Class
	rdfs:subClassOf	rprov:Methodology

Table 6.9: subclass element: rprov:Package

Subject	Predicate	Object
rprov:Package	rdf:type	owl:Class
	rdfs:subClassOf	rprov:Implementation

Table 6.10: subclass element: rprov:Tool

Subject	Predicate	Object
rprov:Tool	rdf:type	owl:Class
	rdfs:subClassOf	rprov:Implementation

Table 6.11: subclass element: rprov:GenericTool

Subject	Predicate	Object
rprov:GenericTool	rdf:type	owl:Class
	rdfs:subClassOf	rprov:Implementation

an implementation. These elements feature a description of the implementation whose information is stored, but more importantly attributes to record the resource of the given package or tool, which ties directly into the integration level of the implementation. A deeply integrated tool for example, might feature a resource where it can be executed directly from the knowledge base, whereas a package located on a lower integration level, might just provide a link to a resource where the package can be downloaded, or an explanation on how to use it. The third subclass, *rprov:GenericTool*, is designed to represent tools that provide general support, for example, converters that transform data from one format to another, such as from CSV to JSON. In the following Table 6.12 the *owl:ObjectProperty* used to model the relationship between the elements on all three abstraction levels will be introduced.

Table 6.12: ObjectProperty to connect elements of all abstraction levels

Subject	Predicate	Object
rprov:realizedBy	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:Methodology, rprov:Implementation
	rdfs:range	rprov:AssessmentApproach

The *rprov:realizedBy* property forms the connection between assessment approaches and their corresponding methodologies and implementations. It enables the storage of information about one or more elements that realize a given assessment approach. This aspect is particularly important, because there can be assessment approaches, that cannot be adequately represented

by a single publication and its corresponding implementation of the proposed method. To address this, the knowledge base is designed to support the association of multiple methodologies and implementations with any given assessment approach. This property ensures a more comprehensive and flexible representation of the underlying concepts. Another relevant *owl:ObjectProperty* is *rprov:predictionMethod*, as presented in Table 6.13. While prediction methods are modeled as filters, with their individual elements represented as filter options, the introduction of a dedicated property for prediction methods proved necessary. This decision was motivated by the requirement to define a clear entry point for the **Guided Section** of the knowledge base, which was designated to be the prediction method. Consequently, to establish an explicit connection between assessment approaches and prediction methods, the following object property is defined:

Table 6.13: ObjectProperty to connect prediction methods with assessment approaches

Subject	Predicate	Object
rprov:predictionMethod	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:FilterOption
	rdfs:range	rprov:AssessmentApproach

The last elements in relation to the *prorv:BusinessUnderstandingEntity* are the properties that enable the recommendation of suitable assessment approaches and their corresponding methodologies and implementations. The following properties, shown in Tables 6.14, 6.15 and 6.16 tie filter options to all elements of the abstraction levels generic, method-specific and problem-specific:

Table 6.14: ObjectProperty rprov:requires

Subject	Predicate	Object
rprov:requires	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:AssessmentApproach, Methodology, Implementation
	rdfs:range	rprov:FilterOption

Table 6.15: ObjectProperty rprov:recommends

Subject	Predicate	Object
rprov:recommends	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:AssessmentApproach, Methodology, Implementation
	rdfs:range	rprov:FilterOption

Table 6.16: ObjectProperty rprov:prohibits

Subject	Predicate	Object
rprov:prohibits	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:AssessmentApproach, Methodology, Implementation
	rdfs:range	rprov:FilterOption

The property *rprov:required* is used to capture the conditions or characteristics that must apply to a given prediction model—more precisely, to the results produced by such a model—in order for a

specific approach, methodology, or implementation to be applicable for the assessment of these results.
The property *recommended* is used to represent characteristics that are not mandatory but are considered beneficial for a given prediction model—or, more precisely, for the results generated by such a model—in order to increase the suitability or effectiveness of applying a specific approach, methodology, or implementation for the assessment of these results.
The property *prohibited* is used to denote characteristics that must not apply to a given prediction model, or to its generated results, as their presence would render the use of the corresponding approach, methodology, or implementation for assessing these results invalid or inappropriate.

KG Elements: Results Section

In order to store information on assessment results, generated by implementations or by manually entering assessment results, the elements relating to *rprov:DeploymentEntity* are explained in this section. Most of these elements are taken directly from the reference processes knowledge graph and are essential to convey the information needed in the **Results Section** of the knowledge base. Accordingly, the elements that are subclasses of the Deployment Entity are described, followed iteratively by their instances such as in the previous section. The first important elements are described in Tables 6.17 and 6.18:

Table 6.17: Subclasses of *rprov:DeploymentEntity*: **rprov:Assessment**

Subject	Predicate	Object
rprov:Assessment	rdf:type	owl:Class
	rdfs:subClassOf	rprov:DeploymentEntity

Table 6.18: Subclasses of *rprov:DeploymentEntity*: **rprov:AssessmentCase**

Subject	Predicate	Object
rprov:AssessmentCase	rdf:type	owl:Class
	rdfs:subClassOf	rprov:DeploymentEntity

As illustrated, both of these elements are instances of *owl:Class* and are defined as subclasses of *rprov:DeploymentEntity*. They form the baseline for representing information on the results of reliability assessments. Subsequently, specific case elements are introduced, each corresponding to a prediction method—such as *Classification*—which serves as an example for demonstrating the modeling of these elements. A key requirement for the knowledge base is the automatic generation of a new triple representing such a case whenever a new prediction method is added to the knowledge graph. In other words, if prediction method *Classification* is added to the knowledge graph, a *ClassificationCase* is also added to the knowledge graph automatically.

Table 6.19 shows an example of a specific case element, in this case *rprov:ClassificationCase*, which is a subclass of *rprov:AssessmentCase* and contains an Object Property *rprov:wasAssignedToDeploymentEntity*. This property, links an Assessment Case to an Assessment, e.g. a Perturbation Assessment would be linked to a Classification Case.

Table 6.19: Specific Case Elements

Subject	Predicate	Object
rprov:ClassificationCase	rdf:type	owl:Class
	rdfs:subClassOf	rprov:AssessmentCase
	rprov:wasAssignedToDeploymentEntity	rprov:Assessment

Table 6.20: Object Property to link Assessment Cases and Assessments

Subject	Predicate	Object
rprov:wasAssignedToDeploymentEntity	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:AssessmentCase
	rprov:range	rprov:DeploymentEntity

The last element featured in this section, is another property, this time linking the Deployment Entity back to the Business Understanding Entity. Specifically, it is a property that forms the connection between an Assessment and an Implementation . Table 6.21 depicts the triple that represents this element:

Table 6.21: Object Property to link Assessments to Implementations

Subject	Predicate	Object
rprov:producedBy	rdf:type	owl:ObjectProperty
	rdfs:domain	rprov:Assessment
	rprov:range	rprov:Implementation

This concludes the description of how the elements of the knowledge graph are modeled. Lastly, the following code listing shows a small excerpt of an example where one assessment approach with its corresponding subclasses from the knowledge graph is shown to conclude this section:

Listing 6.1: Knowledge Graph example

```
<URI-ClassificationFilterOption>
        rdf:type    rprov:FilterOption;
        rdfs:label  "Classification"@en .

<URI-PredictionMethodFilter>
        rdf:type         rprov:Filter;
        rdfs:label       "Prediction Method"@en;
        rprov:hasOption  <URI-ClassificationFilterOption> .

rprov:ApplicabilityDomainApproach
        rdf:type                 owl:Class;
        rdfs:subClassOf          rprov:AssessmentApproach;
        rprov:predictionMethod   <URI-ClassificationFilterOption>;
        rprov:realizedBy         <URI-ADANMethodology>,
            <URI-ADANPackage>.
```

```
15
16 <URI-ADANMethodology>
17         rdf:type          rprov:Idea;
18         rdfs:label        "Applicability Domain Analysis (ADAN)..."@en;
19         rdfs:resource     "https://doi.org/10.1021/ci500172z"@en;
20         rdfs:subClassOf   rprov:ApplicabilityDomainApproach.
21
22 <URI-ADANPackage>
23         rdf:type          rprov:Package;
24         rdfs:comment      "The ADAN Package can be used to assess
               reliability into categories from 0-6..."@en;
25         rdfs:instanceOf   <URI-ADANMethodology>;
26         rdfs:label        "ADAN Package"@en;
27         rdfs:resource     "https://phi.upf.edu/adan/"@en.
28
29 <URI-ADANPackageAssessment>
30         rdf:type                rprov:ADANPackageAssessment;
31         rdfs:label              "Genome Prediction Case #1 - Adan"@en;
32         rdfs:resource           "adan_testCase.csv"@en;
33         rprov:producedBy        <URI-ADANPackage>;
34         rprov:resourceContent   "..."@en;
35         prov:generatedAtTime    "2025-06-20T13-22-05"^^xsd:dateTime .
36
37 <URI-AssessmentCase>
38         rdf:type           rprov:ClassificationCase;
39         rdfs:label         "Genome Prediction Case #1 - Adan"@en;
40         rprov:wasAssignedToDeploymentEntity <URI-ADANPackageAssessment>;
41         prov:endedAtTime   "2025-06-20T13-22-05"^^xsd:dateTime .
```

6.2 System Architecture and Implementation Structure

This section of the implementation perspective deals with the system architecture and general implementation structure of the backend and frontend. First the architecture of the different components will be summarized. Afterwards the implementation structure of the front and backend will be shown.

6.2.1 System Architecture

Figure 6.1 summarizes the system architecture of the knowledge base.
The back-end business logic can be divided into four distinct packages, namely, *Models*, *Services*, *Configurations*, and *Controllers*.

- **Models**
  The models package defines data structures for entities represented in the knowledge base. These Java classes act as mappings of the RDF resources and properties stored in the knowledge graph and received from the Fuseki Graph Store. This package allows for consistent handling of data throughout the backend by providing a structured representation of elements.

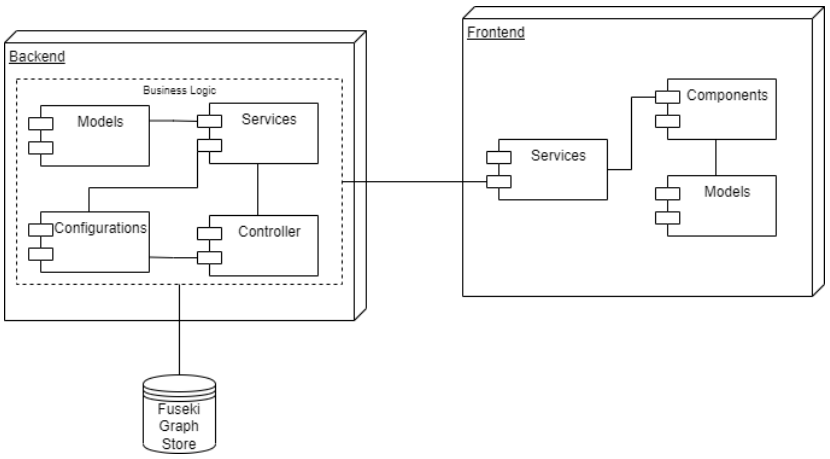


Figure 6.1: System architecture of the knowledge base

- **Services**
  The services package contains the actual logic of the application. Here, the results of SPARQL queries are handled by transforming the results into Java objects and applying additional logic required before passing data to the controllers. Furthermore, the setup to interact with the Fuseki Graph Store is handled in the classes of this package, indicating that this package provides the interaction with the persistence layer.

- **Controllers**
  The controllers package exposes the REST interface. Each controller receives HTTP requests from the frontend, calls the appropriate service, and returns the response in JSON format. In that way, the controllers connect the frontend and backend.

- **Configurations**
  Finally, the package for configurations is a bit different. It does not hold general application setup, but predefined SPARQL queries. These queries are combined with the required input data to form valid SPARQL statements. The results of these queries are then used by the services to retrieve the correct information from Fuseki.

On the frontend side, the implementation is divided into three main parts: *Services*, *Models*, and *Components*.

- The **Services** handle the communication with the backend. They send requests to the REST endpoints provided by the Spring Boot application and return the received data to the components. In this way, the services act as the connection layer between user interactions in the interface and the stored knowledge base data.

- The **Models** represent the data structures used within the frontend. They mirror the entities from the backend and make sure that the exchanged information is handled consistently in TypeScript.

- The **Components** form the user interface itself. They define what is shown to the user, such as the **Browsing Section**, guided workflows, or the results pages. Components use the models to display data and call the services when new information needs to be retrieved or stored.

This separation ensures that data handling, logic, and presentation remain clearly divided, which simplifies development and improves maintainability.

6.2.2 General Implementation Structure

Figure 6.2 shows the structure of the implementation of the four different backend packages with all corresponding classes.

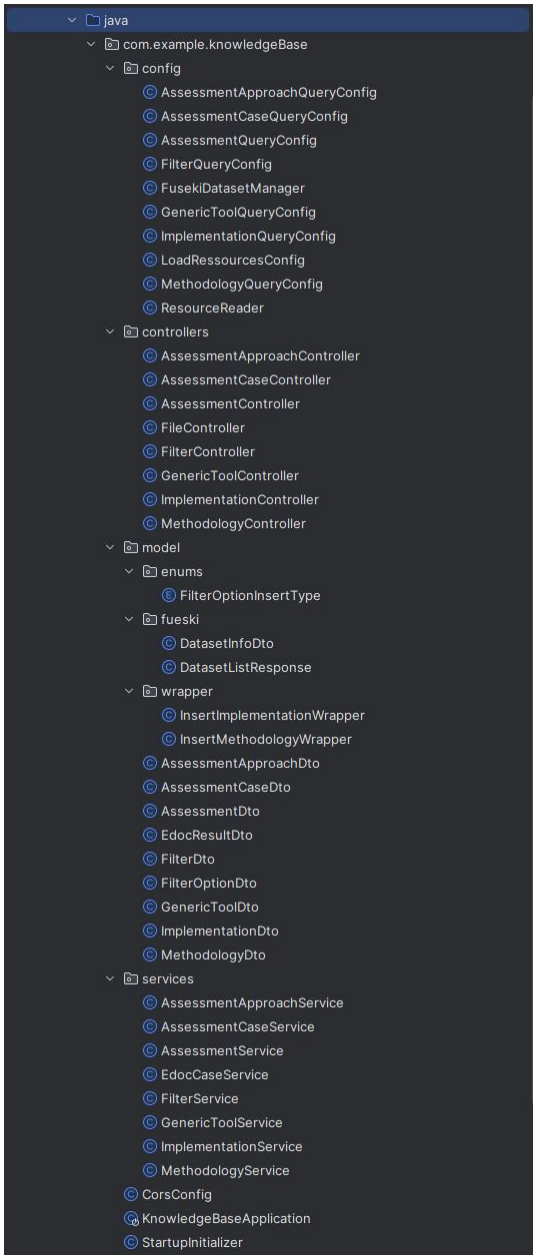


Figure 6.2: Implementation Structure Backend

As can be seen in the figure, from top to bottom the packages each include the corresponding classes needed to provide the backend functionality. In the config package each distinct *Query-Config* class holds information about the queries executed against the knowledge graph to retrieve the necessary information from it. Each class is designed as a static class with static methods, each of those responsible for providing one SPARQL query to be executed. Furthermore, the *FusekiDatasetManager* class is needed to check for existing datasets and creating a new dataset if one does not already existing that corresponds to the knowledge base. Additionally, it uploads

initial data to a newly created dataset. Lastly, the *LoadResourcesConfig* and *ResourceReader* classes are responsible to load resources from the respective directory, common within Spring Boot applications.

In the controllers package each class exposes the available REST endpoints for the corresponding elements like assessment approaches or implementations.

The classes within the model package are mostly made up of data transfer objects (DTO), which contain the data structures corresponding with elements from the knowledge graph. Additionally, Enumeration and wrapper classes support insertion or retrieval operations. In the fueski sub package, models that represent datasets and are used to check for dataset availability as described in the config package section, are contained.

Lastly, the services package implements the application logic. All classes represents the corresponding element for which logic has to be implemented, such as processing query results and transforming the data into objects ready to use within the backend. Additionally, each service corresponds to a controller to separate logic and request handling.

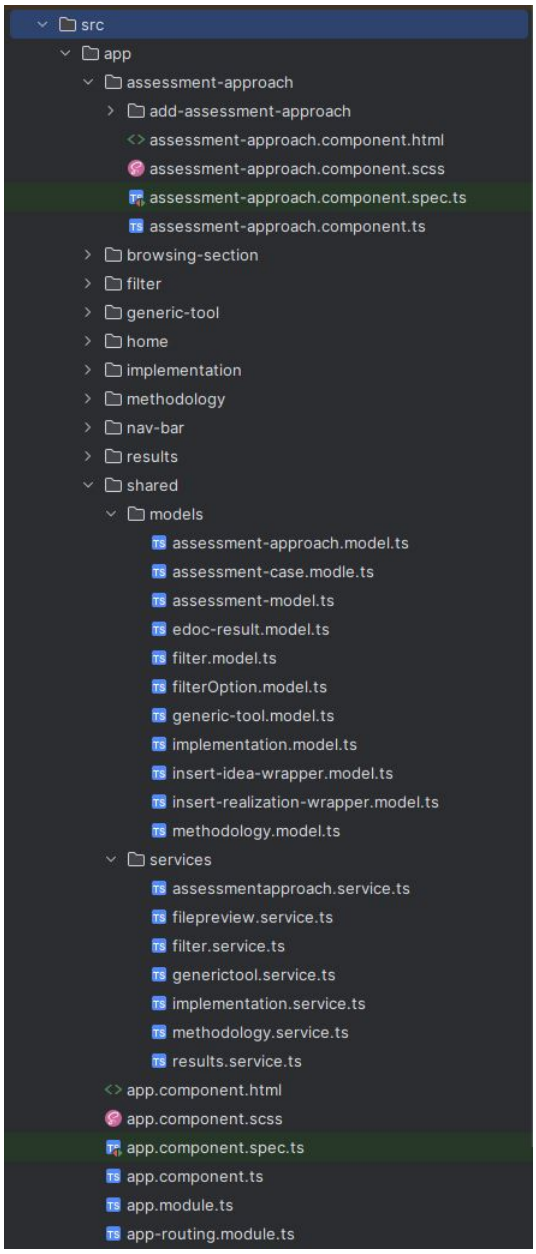


Figure 6.3: Implementation Structure Frontend

The frontend of the knowledge base is built with Angular and organized into several feature folders. Each folder represents one part of the system, for example *assessment-approach*, *browsing-section*, *filter*, *generic-tool*, *implementation*, *methodology*, or *results*. Inside these folders, the Angular components are defined, usually with an HTML file for the template, a SCSS file for the styles, and a TypeScript file for the actual logic. This makes it clear where to look when a change in the interface is needed. In addition to these feature-specific parts, there is a *shared* folder. It collects code that is reused in several places. Two main areas can be found here:

- **models**, which are TypeScript classes describing the data that comes from the backend, such as filters, implementations, or methodologies. They help keep the data consistent and typed.
- **services**, which are responsible for communication with the backend. Each service matches a certain domain, for example *AssessmentApproachService*, *FilterService*, or *ResultsService*. They handle the HTTP requests and make the data available to the components.

At the top level, the application is tied together by the *AppModule*. It declares the main component, sets up routing between pages, and loads the feature modules. The global layout is controlled through the *AppComponent*, which includes its HTML, SCSS, and TypeScript files.
Overall, the structure keeps components focused on presentation, services on data exchange, and models on data consistency. This separation makes the front easy to maintain and extend when new elements need to be added to the presentation of the knowledge base.

6.3 Guided Section and Browsing Section

This part of the implementation perspective will introduce the most interesting implementation aspects concerning the guided and the **Browsing Section** of the knowledge base and will be divided into three main areas. The first, highlights implementation aspects regarding the presentation of information within the guided and **Browsing Section**, more specifically how assessment approaches and their corresponding sub elements are queried from the knowledge graph, transformed into their respective Java objects and prepared for the subsequent controllers to convey the necessary information to the front end.
The second main area concerns the extensibility of the knowledge base. Here, a detailed examination of the implementation aspects on how to add the elements regarding methodologies and implementations will be provided. More specifically, how the queries to add this information to the knowledge graph are built, especially factoring in the addition of newly created and already existing filter options in this process.
Lastly, a deep dive into the filtering and selection procedures will be conducted within this section. It will explain how filter options are applied to elements within both the guided and the **Browsing Section** by providing implementation details from the backend. Furthermore, aspects of the frontend implementation will be highlighted, to show the workings of the selection algorithm presented in the **Browsing Section**.

6.3.1 Presentation of Information

Consider Listing 6.2, which shows an excerpt of the knowledge graph containing those elements that are relevant for presentation in both the **Guided** and **Browsing Section**s of the system. The raw information stored in the knowledge graph is expressed as RDF triples, which on their own are not directly suitable for user interaction. To make the data accessible and visually clear within the knowledge base, these triples must first be processed and transformed into a format that can be displayed in the web interface. This transformation step ensures that abstract graph structures are converted into comprehensible entities, such as cards, lists, or expandable panels, as presented in the guided and browsing views. This section will explain in detail how this is achieved.

Listing 6.2: Example elements for presentation

```
rprov:PerturbationApproach
    rdf:type                owl:Class;
    rdfs:subClassOf         rprov:AssessmentApproach;
    rprov:predictionMethod
        <urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e>;
    rprov:realizedBy
        <urn:uuid:a7de5b48-b7e2-4ff5-978e-c6418a93b78e> .

<urn:uuid:f6b36ec4-5bd8-43a7-ad17-b15cdf39bc04>
    rdf:type          rprov:Methodology;
    rdfs:label        "A Reference Process for Assessing the Reliability
        of Predictive Analytics Results"@en;
    rdfs:resource     "https://doi.org/10.1007/s42979-024-02892-4";
    rdfs:subClassOf   rprov:PerturbationApproach;
    rprov:recommends  <urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e> .

<urn:uuid:a7de5b48-b7e2-4ff5-978e-c6418a93b78e>
    rdf:type          rprov:Tool;
    rdfs:comment      "EDOC Perturbation Tool is a software solution to
        perturb testcases..."@en;
    rdfs:instanceOf   <urn:uuid:f6b36ec4-5bd8-43a7-ad17-b15cdf39bc04>;
    rdfs:label        "EDOC Perturbation Tool"@en;
    rdfs:resource     "http://localhost:8501/";
    rprov:recommends  <urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e>;
    rprov:requires    <urn:uuid:92f9b0e7-2c6e-465a-ada6-2acb62e64f9c> .

<urn:uuid:2eac4a04-7cfd-4283-91d5-ca975e418352>
    rdf:type         rprov:Filter;
    rdfs:label       "Data Type"@en;
    rprov:hasOption  <urn:uuid:92f9b0e7-2c6e-465a-ada6-2acb62e64f9c>  .

<urn:uuid:92f9b0e7-2c6e-465a-ada6-2acb62e64f9c>
    rdf:type    rprov:FilterOption;
    rdfs:label  "Tabular Data"@en .

<urn:uuid:0efc2f8b-05be-4055-a784-e5854aecd389>
    rdf:type         rprov:Filter;
    rdfs:label       "Prediction Method"@en;
```

```
    rprov:hasOption  <urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e> .

<urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e>
    rdf:type    rprov:FilterOption;
    rdfs:label  "Classification"@en .
```

In order to retrieve the necessary information from the knowledge graph, first, the frontend has to make a request to the backend, more specifically to the GET endpoint `assessmentApproaches` exposed by the *AssessmentApproachController* in the backend. As additional optional information, the assessment approach service in the frontend can provide a prediction method to retrieve only those assessment approaches corresponding to the provided prediction method. Additionally, the service can also provide a request body, containing filter options to narrow down the results even more (a detailed explanation of this will be provided at a later point in this chapter). For the sake of simplicity, this explanation will focus only on a request without the optional information provided and only on the implementation aspects regarding the backend. When this request is received by the backend, the controller firstly calls the `getAssessmentApproaches` method located in the *AssessmentApproachService*. This method calls the *AssessmentApproachQueryConfig* to request the SPARQL query to retrieve the required information from the knowledge graph. Afterwards, it builds a connection to the Fuesik Graph Store, executes the query and returns the transformed result back to the controller. The method can be seen in Listing 6.3.

Listing 6.3: AssessmentApproachService - getAssessmentApproaches(...)

```
public List<AssessmentApproachDto> getAssessmentApproaches(List<String>
    optionIds, String predictionMethodUuid) {
    Query query = QueryFactory.create(
            AssessmentApproachQueryConfig.q_selectAssessmentApproach(
                optionIds, predictionMethodUuid));
    ByteArrayOutputStream out = new ByteArrayOutputStream();

    try ( RDFConnectionFuseki conn =
        (RDFConnectionFuseki)builder.build() ) {
        conn.queryResultSet(query, rs ->
            ResultSetFormatter.outputAsCSV(out, rs));
    }
    return cascadeFilterOptionsUpwards(
        new AssessmentApproachDto().convertCsvToDtos(out.toString()));
}
```

This method is listed here, because it features some important aspects relevant all across the service classes. Within the `try` block, a connection to the Fuseki Graph Store is established, following the same pattern that is consistently applied throughout the backend whenever the knowledge graph is queried. The query result is written to a temporary output stream in CSV format. Afterwards, the data is processed by the `convertCsvToDtos()` method, which transforms the raw query results into DTOs. This transformation step is crucial, as it provides the structured data representation required for later transmission to the frontend. There are three methods worth discussing here, `q_selectAssessmentAppraoch(...)`, `convertCsvToDtos(...)` and `cascadeFilterOptionsUpwards(...)`. Lets start with the one that provides the actual

SPARQL query, `q_selectAssessmentAppraoch(...)`. Listing 6.4 shows the query built in this method.

Listing 6.4: AssessmentApproachQueryConfig - q_getAssessmentApproach(...)

```
SELECT * WHERE {
    ?label rdfs:subClassOf rprov:AssessmentApproach ;
    rprov:predictionMethod ?predictionMethod ;
    OPTIONAL {
        ?methodologyUuids rdfs:subClassOf ?label;
        rdfs:label ?methodologyLabels;
        rdfs:resource ?methodologyResources ;
        OPTIONAL { ?methodologyUuids rprov:recommends
            ?recommendedMethodologyOptions. }
        OPTIONAL { ?methodologyUuids rprov:requires
            ?requiredMethodologyOptions. }
        OPTIONAL { ?methodologyUuids rprov:prohibits
            ?prohibitedMethodologyOptions. }
        OPTIONAL {
            ?implementationUuids rdfs:instanceOf ?methodologyUuids;
            rdfs:label ?implementationLabels;
            rdfs:comment ?implementationComments;
            rdfs:resource ?implementationResources;
            rdf:type ?implementationTypes;
            OPTIONAL { ?implementationUuids rprov:recommends
                ?recommendedImplementationOptions. }
            OPTIONAL { ?implementationUuids rprov:requires
                ?requiredImplementationOptions. }
            OPTIONAL { ?implementationUuids rprov:prohibits
                ?prohibitedImplementationOptions. }
        }
    }
}
```

The first part of the query identifies all elements that are subclasses of *rprov:AssessmentApproach*. The next block, inside an *OPTIONAL* clause, extends the query to cover methodologies that are related to a given assessment approach. A methodology is a subclass of the approach and described by a label and a resource. In addition, each methodology may have filter option properties, *recommends*, *requires*, and *prohibits*, which capture the conditions or recommendations relevant for applying the method. Since not all methodologies define such properties, they are queried as optional values. Inside this methodology block lies another nested *OPTIONAL* clause that deals with implementations. An implementation is connected to its methodology via the relation *rdfs:instanceOf* and may represent a tool or a package as specified by its *rdf:type*. Each implementation includes descriptive information such as a label, a comment, and a resource link. Similar to methodologies, implementations can also be associated with the properties *recommends*, *requires*, and *prohibits*, which are again queried in an optional section.

The use of *OPTIONAL* in this query ensures that even if certain elements are missing, for example, an assessment approach without a defined methodology, or a methodology without implementations, the higher-level elements still appear in the results. This prevents incomplete

data from being excluded entirely and allows the backend to assemble consistent result sets for display. The outcome of the query is a combined view of assessment approaches, their prediction methods, any associated methodologies, and, where available, their implementations. At the same time, all filter option properties across approaches, methodologies, and implementations are included. This structure makes it possible to gather all data required for presenting assessment approaches in the guided and **Browsing Section**s with a single query.

The important aspect to highlight about this query is the occurrence of "duplicate" results in the query, more specifically multiple returned rows per assessment approach. This occurs because the query pattern introduces several one-to-many relationships that lead to multiple rows for the same conceptual elements. More specifically, assessment approaches can be associated with several methodologies, those in turn can have multiple implementations and each element can have multiple filter options. The usage of `OPTIONAL` does not eliminate this behavior as it acts similar to a left join in SQL. These implications together explain why the CSV result potentially contains the same assessment approach label across multiple rows, why methodologies appear repeatedly with different option bindings, and why implementations are listed more than once when they are associated with multiple options or types.

After the query is built, the result yielded is stored temporarily in an output stream which will be handeled by the second method `convertCsvToDtos(...)`. This method completes multiple tasks starting with mapping the headers of each column from the CSV result to properties of the `AssessmentApproachDTO`. The labels prefaced with *?* in the SPARQL query stand for the names of the properties in the DTO and thus allow for a matching of the retrieved data. Afterwards, three steps need to be taken in order to yield the correct result needed. First, a way to remove potentially duplicate entries, while preserving the correct parent-child links between assessment approaches, methodologies and implementations. Second, reconstructing explicit methodology objects and assign them to the correct assessment approaches. Lastly, reattaching the implementations to the corresponding methodologies. Listing 6.5 shows how duplicate assessment approaches are grouped together by label and how duplicate information regarding methodologies and implementations are avoided.

Listing 6.5: AssessmentAppraochDto - convertCsvToDtos(...) - Step 1

```
List<AssessmentApproachDto> assessmentApproachDtosUnmerged =
    iterator.readAll();
List<AssessmentApproachDto> assessmentApproachDtos = new ArrayList<>();
for(AssessmentApproachDto assessmentApproachDto:
    assessmentApproachDtosUnmerged) {
    if (assessmentApproachDtos.stream().noneMatch(a -> a.getLabel()
    .equals(assessmentApproachDto.getLabel()))) {
        assessmentApproachDtos.add(assessmentApproachDto);
    } else {
        assessmentApproachDtos.stream()
                .filter(a -> a.getLabel()
                    .equals(assessmentApproachDto.getLabel()))
                .forEach(a -> {
                    Set<String> methodologyUuids = new
                        LinkedHashSet<>(a.getMethodologyUuids());
                    methodologyUuids.addAll(
                        assessmentApproachDto.getMethodologyUuids());
```

```
15                      a.setMethodologyUuids(new
                            ArrayList<>(methodologyUuids));
16
17                      ...
18                  });
19      }
20  }
```

The grouping by label can be seen in the first `if` clause of the listing. If an assessment approch lable is seen for the first time a new row is added to the list. If this is not the case the code unions the multi-valued fields. Within the `AssessmentApproachDto` there are multiple fields corresponding to the retrieved elements which are stored as lists of `String`. These lists are converted into a `LinkedHashSet` to preserve the order and the prevent duplicate entries. Afterwards it writes the information back to the list contained in the assessment approach. Step 1 therefore yields the outcome of one object per assessment approach label, containing lists without duplicated methodologies and implementations. However, those lists are still parallel lists containing the *Uuids* of the corresponding elements and not yet structured. Steps 2 and 3 solve this problem by converting the parallel lists into structured objects. Listing 6.6 shows the excerpt of the method for this step. The methodologies are assigned to the assessment approach objects by iterating over the aligned properties (UUID, label, resource) and adding each instance to a new list of methodologies. This list is added to the methodology list of the assessment approach after merging.

Listing 6.6: AssessmentAppraochDto - convertCsvToDtos(...) - Step 2

```
1  for(AssessmentApproachDto assessmentApproachDto: assessmentApproachDtos)
       {
2      List<MethodologyDto> methodologyDtos = new ArrayList<>();
3      for (int i = 0; i <
           assessmentApproachDto.getMethodologyUuids().size(); i++) {
4          methodologyDtos.add(new MethodologyDto(
5              assessmentApproachDto.getMethodologyUuids().get(i),
6              assessmentApproachDto.getMethodologyLabels().get(i),
7              assessmentApproachDto.getMethodologyResources().get(i)));
8      }
9      assessmentApproachDto.setMethodologies(methodologyDtos);
10 }
```

The final step completes this method and tries to reattach the implementation objects to the methodologies. Listing 6.7 shows the excerpt of step 3 below.

Listing 6.7: AssessmentAppraochDto - convertCsvToDtos(...) - Step 3

```
1  assessmentApproachDtosUnmerged = iterator.readAll();
2  for (AssessmentApproachDto assessmentApproachDto :
       assessmentApproachDtos) {
3      List<MethodologyDto> methodologyDtos =
           assessmentApproachDto.getMethodologies();
4      for (MethodologyDto methodologyDto : methodologyDtos) {
5          Set<String> uniqueImplementationUuids = new HashSet<>();
6          List<ImplementationDto> implementationDtos = new ArrayList<>();
```

```
7           for (AssessmentApproachDto assessmentApproachDtoUnmerged :
                assessmentApproachDtosUnmerged) {
8               if (assessmentApproachDtoUnmerged.getMethodologyUuids()
9                   .contains(methodologyDto.getUuid())) {
10                  for (int i = 0; i < assessmentApproachDtoUnmerged
11                      .getImplementationUuids().size(); i++) {
12                      String implementationUuid =
                            assessmentApproachDtoUnmerged
13                          .getImplementationUuids().get(i);
14                      if
                            (uniqueImplementationUuids.add(implementationUuid))
                            {
15                          implementationDtos.add(new ImplementationDto(
16                                  implementationUuid,
17                                  assessmentApproachDtoUnmerged
18                                      .getImplementationLabels().get(i),
19                                  assessmentApproachDtoUnmerged
20                                      .getImplementationComments().get(i),
21                                  assessmentApproachDtoUnmerged
22                                      .getImplementationResources().get(i),
23                                  assessmentApproachDtoUnmerged
24                                      .getImplementationTypes().get(i)
25                          ));
26                      }
27                  }
28              }
29          }
30          methodologyDto.setRealizations(implementationDtos);
31      }
32  }
```

To preserve the correct parent–child mapping (methodology - implementation), the algorithm makes a second pass over the unmerged rows. For each methodology attached to an approach, it scans the raw rows and selects those that reference this methodology's UUID. From these rows, it collects implementation attributes (UUID, label, comment, resource, type) by aligned index, instantiates *ImplementationDto* objects, and de-duplicates them using a HashSet of UUIDs. The resulting list is then assigned to the methodology via `setImplementations(...)`. Outcome of Step 3: each methodology contains exactly its own implementations, de-duplicated and correctly linked—thus restoring the hierarchy that was flattened by the join. To summarize, the `convertCsvToDtos(...)` method reutrns a list where each assessment approach appears once, each approach holds a list of methodologies without duplicate results, and each methodology holds a list of implementations without duplicates and in correct ordering. With this result generated, the final method `cascadeFilterOptionsUpwards(...)`, highlighted in Listing 6.8 is now called by the algorithm.

Listing 6.8: AssessmentAppraochService - cascadeFilterOptionsUpwards(...)

```
1   private List<AssessmentApproachDto>
        cascadeFilterOptionsUpwards(List<AssessmentApproachDto>
```

```
    assessmentApproachDtos) {
2    for (AssessmentApproachDto assessmentApproachDto :
         assessmentApproachDtos) {
3        List<MethodologyDto> methodologyDtos =
             assessmentApproachDto.getMethodologies();
4        for(MethodologyDto methodologyDto : methodologyDtos) {
5            List<ImplementationDto> implementationDtos =
                 methodologyDto.getImplementations();
6            for (ImplementationDto implementationDto :
                 implementationDtos) {
7                ImplementationDto currentImplementation =
8                    implementationService.
9                        getRealizationById(implementationDto.getUuid());
10               assessmentApproachDto.addRecommendedFilterOptions(
11                   currentImplementation.getRecommendedFilterOptions());
12               assessmentApproachDto.addRequiredFilterOptions(
13                   currentImplementation.getRequiredFilterOptions());
14               assessmentApproachDto.addProhibitedFilterOptions(
15                   currentImplementation.getProhibitedFilterOptions());
16           }
17       }
18       assessmentApproachDto.setRecommendedFilterOptions(
19           assessmentApproachDto.getRecommendedFilterOptions()
20               .stream().distinct().toList());
21       assessmentApproachDto.setRequiredFilterOptions(
22           assessmentApproachDto.getRequiredFilterOptions()
23               .stream().distinct().toList());
24       assessmentApproachDto.setProhibitedFilterOptions(
25           assessmentApproachDto.getProhibitedFilterOptions()
26               .stream().distinct().toList());
27   }
28   return assessmentApproachDtos;
29 }
```

This method iterates over all assessment approaches and subsequently over all sub elements. For each sub element, the lists of recommended, required and prohibited filter options is added to the list of the respective filter options of the assessment approach. After all iterations are completed for the sub elements, the algorithm checks for duplicates in the filter option lists at the end of each iteration of assessment approaches. The result of this method is a combined list of each type of filter option for all elements related to an assessment approach.
This method marks the last piece of data processing before the result of the frontend request to retrieve the data for all assessment approaches is sent back to the frontend as a JSON object. In the frontend, this data will be serialized to the model classes of the front end and displayed as shown in the users point of view section of this thesis.

6.3.2 Extending the Knowledge Base

As previously discussed in Chapter 5 the knowledge base provides the possibility to extend the knowledge graph by adding elements such as filter, assessment approaches, or methodologies

and implementations. This section will focus on the aspects of adding an implementation to a methodology. As highlighted before, when adding an implementation, or a methodology for that fact, the user has the option to additionally add existing or newly created filter options and filters to this implementation. The following listings and explanations will discuss how this is achieved in the system. The first step to adding new elements is for the user to input the information needed into the corresponding forms in the knowledge base. Then the frontend needs to send an HTTP Post request to the backend, where the request body needs to contain a JSON object representing the new data to be added. Listing 6.9 shows such a request body when adding the Perturbation Tool to the knowledge base.

Listing 6.9: Request Body for adding a new Implementation

```
{
    "label": "Perturbation Tool",
    "comment": "Perturbation Tool is a software solution to perturb
       testcases...",
    "resource": "https://doi.org/10.1007/s42979-024-02892-4",
    "methodologyUuid": "urn:uuid:f6b36ec4-5bd8-43a7-ad17-b15cdf39bc04",
    "type": "Tool",
    "recommendedOptions": [
        "urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e"
    ],
    "requiredOptions": [
        "urn:uuid:92f9b0e7-2c6e-465a-ada6-2acb62e64f9c"
    ],
    "prohibitedOptions": [
        "urn:uuid:6cec1810-2f8a-4512-8e1a-2e37c2c62b16"
    ],
    "newFilterOptions": [
        {
            "label": "unstructured",
            "filter": "urn:uuid:f9cbbc87-9ee9-41a5-8e00-7a40083b57fe",
            "type": "PROHIBITED"
        },
        {
            "label": "Finance",
            "filter": null,
            "type": "RECOMMENDED"
        }
    ],
    "newFilters": [
        {
            "label": "Domain",
            "options": [
                "Finance"
            ]
        }
    ]
}
```

In this JSON representation the values entered by the user are stored and mapped by the backend to the corresponding java object `InsertImplementationWrapper`. This object is highlighted in Listing 6.10. Each field within this java class matches with each property in the JSON object, allowing for seamless mapping from JSON representation to a Java object.

Listing 6.10: Wrapper for inserting a new implementation

```
1  public class InsertImplementationWrapper {
2      String label;
3      String comment;
4      String resource;
5      String methodologyUuid;
6      String type;
7      List<String> recommendedOptions;
8      List<String> requiredOptions;
9      List<String> prohibitedOptions;
10     List<FilterOptionDto> newFilterOptions;
11     List<FilterDto> newFilters;
12
13     /* constructors, getter and setter methods */
14 }
```

When the request to add a new implementation is received by the backend the corresponding controller calls the `ImplementationService` which establishes a connection to the Fuseki Graph Store. Afterwards the method for building the necessary SPARQL query is called. The tasks of this method can be separated into three areas. First the part of the `INSERT DATA` statement concerning the addition of potential new filters with new filter options is built. Afterwards, the query to filter options to either already existing filters is constructed. Lastly, the final insert statement is built considering all the other available data. Listing 6.11 shows an excerpt of the first part of the `q_insertImplementation(...)` method.

Listing 6.11: q_insertImplementation(...) - insert new Filters

```
1  public static String q_insertImplementation(InsertImplementationWrapper
       newImplementation) {
2      StringBuilder query = new StringBuilder();
3      ImplementationDto implementation = new ImplementationDto(...);
4      List<FilterOptionDto> newFilterOptions =
           newImplementation.getNewFilterOptions();
5      List<FilterDto> newFilters = newImplementation.getNewFilters();
6
7      for (FilterDto newFilter : newFilters) {
8          List<String> providedUuids = newFilter.getOptions().stream()
9                  .map(option -> String.format("%s", UUID.randomUUID()))
10                 .toList();
11         for (int i = 0; i < newFilter.getOptions().size(); i++) {
12             String optionLabel = newFilter.getOptions().get(i);
13             String optionUuid = providedUuids.get(i);
14
15             newFilterOptions.stream()
```

```
16                      .filter(filterOption ->
                            filterOption.getLabel().equals(optionLabel))
17                      .forEach(filterOption ->
                            filterOption.setUuid(String.format("urn:uuid:%s",
                            optionUuid)));
18          }
19          query.append(FilterQueryConfig
20                  .q_insertFilter(newFilter, providedUuids.toArray(new
                        String[0]))
21                  .replace(prefixString, "")
22          );
23      }
24      ...
25  }
```

At the start of this method a new `StringBuilder` object is created which holds the string for the query. Afterwards, the data for the implementation is retrieved from the wrapper object and the new filter options and filters are stored in separate lists. To start the first task the algorithm iterates over all new filters provided by the request body and generates new UUIDs for each new filter option corresponding to the new filter. In a second nested iteration, the algorithm iterates a number of times equal to the amount of new UUIDs generated and initializes the label and UUID for the new filter option. In order to assign the correct data to the new filter options, the options contained in the list of new filters are matched with the options contained in the list of new filter options. Lastly, a SPARQL `INSERT DATA` statement is appended to the query by calling the `q_insertFilter` method from the `FilterQueryConfig` class. The called method takes a new filter and an arbitrary amount of UUIDs as parameters and is responsible for building the statement to insert a new filter and all new filter options associated with this filter.

Listing 6.12: q_insertImplementation(...) - insert new Filter Options

```
1  ...
2  for(FilterOptionDto newFilterOption : newFilterOptions) {
3      if(newFilterOption.getUuid() == null) {
4          String uuid = String.format("urn:uuid:%s", UUID.randomUUID());
5          newFilterOptions.stream()
6                  .filter(filterOption -> filterOption.getLabel().equals(
7                      newFilterOption.getLabel()))
8                  .forEach(filterOption -> filterOption.setUuid(uuid));
9          query.append(FilterQueryConfig
10                 .q_updateFilterWithExistingOption(newFilterOption, uuid)
11                 .replace(prefixString, ""));
12     }
13     switch (newFilterOption.getType()) {
14         case RECOMMENDED -> implementation.getRecommendedOptions()
15             .add(newFilterOption.getUuid());
16         case REQUIRED -> implementation.getRequiredOptions()
17             .add(newFilterOption.getUuid());
18         case PROHIBITED -> implementation.getProhibitedOptions()
19             .add(newFilterOption.getUuid());
20     }
```

```
}
```

The second task of the `q_insertImplementation(...)` method looks for any new filter options which do not already have a provided UUID and builds a statement to add them to the corresponding and already existing filter. This is achieved by calling the `q_updateFilterWithExistingOption` method, responsible for building the statement to insert a filter option into the knowledge graph and extending the *rprov:hasOption* attribute of the filter the new filter option is associated with. Additionally, it adds the new filter options to the implementations recommended, required or prohibited filter options based on the implementation type. Listing 6.12 shows this in practice. The last part is to build the final query with the remaining data available. After this step the resulting query, considering the request body shown previously, is highlighted in Listing 6.13.

Listing 6.13: q_insertImplementation(...) - final INSERT DATA statement

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rprov: <http://www.dke.uni-linz.ac.at/rprov#>

INSERT DATA {
  <urn:uuid:a463161e-8596-4071-b80b-27ec8cf0fc39> a rprov:FilterOption ;
  rdfs:label 'Finance'@en .
};
INSERT DATA {
  <urn:uuid:23ec4ef8-3391-465f-b3de-c05c7e78bb48> a rprov:Filter ;
  rprov:hasOption <urn:uuid:a463161e-8596-4071-b80b-27ec8cf0fc39> ;
  rdfs:label 'Domain'@en .
};
INSERT DATA {
  <urn:uuid:98614629-36fc-4b65-a16d-074dbe222610> a rprov:FilterOption ;
  rdfs:label 'unstructured'@en .};
INSERT DATA {
  <urn:uuid:f9cbbc87-9ee9-41a5-8e00-7a40083b57fe> rprov:hasOption
      <urn:uuid:98614629-36fc-4b65-a16d-074dbe222610> .
};
INSERT DATA {
  <urn:uuid:bc63c6c8-a4a9-4a15-b549-a94773483e5c> a rprov:Tool ;
  rdfs:label 'EDOCPerturbationTool'@en ;
  rdfs:comment 'EDOC Perturbation Tool is a software solution to perturb
      testcases...'@en ;
  rdfs:resource 'https://doi.org/10.1007/s42979-024-02892-4'@en ;
  rdfs:instanceOf <urn:uuid:f6b36ec4-5bd8-43a7-ad17-b15cdf39bc04> ;
  rprov:recommends <urn:uuid:61381eba-0d9f-441b-9a95-bc61ddd10a5e> ;
  rprov:recommends <urn:uuid:a463161e-8596-4071-b80b-27ec8cf0fc39> ;
  rprov:requires <urn:uuid:92f9b0e7-2c6e-465a-ada6-2acb62e64f9c> ;
  rprov:prohibits <urn:uuid:6cec1810-2f8a-4512-8e1a-2e37c2c62b16> ;
  rprov:prohibits <urn:uuid:98614629-36fc-4b65-a16d-074dbe222610> ;
};
```

6.3.3 Filtering Available Information

As previously mentioned in chapter 6.3.1 the SPARQL query mentioned in Listing 6.4 was not shown in its completed form. The aspect of filtering the results based on a selection of filter options was omitted previously. The following excerpt from this query shows how the process of filtering was implemented with the power of SPARQL.

Listing 6.14: Extended select query - Filtering results

```
FILTER (
  ((
    ?recommendedRealizationOptions IN (%s) ||
    ?requiredRealizationOptions IN (%s)
  )
  &&
  (
    !BOUND(?prohibitedOptions) ||
    ?prohibitedRealizationOptions NOT IN (%s)
  ))
  ||
  ((
    ?recommendedIdeaOptions IN (%s) ||
    ?requiredIdeaOptions IN (%s)
  )
  &&
  (
    !BOUND(?prohibitedOptions) ||
    ?prohibitedRIdeaOptions NOT IN (%s)
  ))
)
```

In the extended query, the FILTER part is used to check whether the user's chosen filter options match the conditions stored in the knowledge base. It restricts the results so that only those methodologies and implementations are returned that fit the selected options, while excluding cases that are explicitly defined as prohibited. The structure of the filter consists of two logical blocks that are connected by a disjunction (OR). The first block applies to implementations, whereas the second block applies to methodologies. A result is included if either of the two blocks evaluates to true.

At the implementation level, the filter requires that an implementation must match at least one of the user-selected options via either a recommended or a required condition. This is expressed by checking whether the variable representing recommended implementation options or the variable representing required implementation options is contained in the set of user-selected option identifiers. At the same time, the implementation must not violate any prohibited condition. This is ensured by requiring either that no prohibited option is bound for the implementation, or that the prohibited option is not among the selected option identifiers. Together, these conditions guarantee that only implementations which align with the user's chosen options and do not contradict prohibited ones are included in the results.

The second block applies the same logic at the methodology level. A methodology is considered valid if at least one of its recommended or required options corresponds to one of the user's

selected options. Similar to implementations, a methodology is excluded if it declares a prohibited option that overlaps with the user's selection. If no prohibited option is bound for the methodology, the condition is automatically satisfied. This ensures that methodologies are only accepted if they fit positively with the chosen options and do not conflict with explicitly excluded conditions. Overall, the filter ensures that the query output only contains elements that are applicable under the given conditions. This means that a methodology or implementation will only appear if it is compatible with at least one of the selected options and, at the same time, does not violate any of the prohibitions. By doing so, the query respects both the positive and negative applicability rules defined in the knowledge base.

6.3.4 Selection of Available Information

The last part of interesting implementation aspects for this chapter concerns the selection of prediction methods in the **Browsing Section**. As explained in the users perspective, this is the second way of narrowing down the data displayed in the **Browsing Section**. The following section will provide code listings and explanations on how this selection is achieved. An important note about the implementation of the selection algorithm is, that it is implemented purely on the client side to seamlessly enable and disable the elements shown in the **Browsing Section**. First, lets highlight a small snippet from the html page in Listing 6.15.

Listing 6.15: Prediction Method element with Checkbox

```
<div *ngFor="let predictionMethod of filteredPredictionMethods; index as
    i">
    <div class="card" (click)="toggleCheckbox(predictionMethod, $event)">
      <div class="card-body">
        <div class="container">
          <div class="row">
            <div class="col-2">
              <input class="mt-2" type="checkbox" title="Classification"
                     (change)="onCheckboxChange(predictionMethod,
                        $event)"
                     [checked]="predictionMethod.selected"
                     (click)="$event.stopPropagation()">
            </div>
            <div class="col-10">
              <h3 class="card-title">{{predictionMethod.label}}</h3>
            </div>
          </div>
        </div>
      </div>
    </div>
</div>
```

The HTML snippet represents an Angular template that iterates over a list of prediction methods and renders them as selectable cards. The outer element uses the Angular directive `<div *ngFor="let predictionMethod of filteredPredictionMethods; index as i">` to loop through all elements in the list filteredPredictionMethods. For each prediction method, a separate card is created. The `index as i` part additionally provides the loop index, which can be used if needed. Inside each

loop iteration, the prediction method is displayed in the form of a Bootstrap card. The card itself is clickable, as defined by the event binding `(click)="toggleCheckbox(predictionMethod, $event)"` which calls a component method whenever the card is clicked. This design allows the entire card to act as a selection area, not only the checkbox. The content of the card is structured using Bootstrap's grid system. The card body contains a container with one row and two columns. The first column (col-2) contains the checkbox input. The checkbox is linked to the corresponding prediction method and reflects its current state through the binding `[checked]="predictionMethod.selected"` so that the checkbox remains checked if the element's selected property is true. To prevent double execution of events, the checkbox click is stopped from propagating to the surrounding card by `(click)="$event.stopPropagation()"` while changes in the checkbox state are handled by `(change)="onCheckboxChange(predictionMethod, $event)"` The second column ('col-10') displays the label of the prediction method as the card title, shown inside an <h3> element. The important aspect of this snippet is the onCheckboxChange(...) method, which will be explained in detail in the following.

Listing 6.16: Selection Algorithm - onCheckboxChange(...) function

```
onCheckboxChange(element: any, $event: Event) {
    this.resetSelectFlags()
    let isChecked = ($event.target as HTMLInputElement).checked

    function isFilterOption(element: any) {
      return 'filter' in element;
    }

    if(isFilterOption(element)) {
      let nonCorrespondingAssessmentApproaches: AssessmentApproach[] =
          [];
      if(isChecked) {
        nonCorrespondingAssessmentApproaches =
            this.filteredAssessmentApproaches.filter(a =>
          a.predictionMethod != element.uuid
        )

        if(this.disabledApproaches.length == 0 &&
            this.selectedMethods.length == 0) {
          this.disabledApproaches = nonCorrespondingAssessmentApproaches;
        } else {
          let common = this.disabledApproaches.filter(a =>
              nonCorrespondingAssessmentApproaches.includes(a))
          this.disabledApproaches = common;
          nonCorrespondingAssessmentApproaches = this.disabledApproaches;
        }
        this.selectedMethods.push(element);
        this.setSelectFlags(nonCorrespondingAssessmentApproaches,
            isChecked);
      } else {
        let temp: AssessmentApproach[] = []
        this.selectedMethods = this.selectedMethods.filter(m => m.uuid
```

```
            != element.uuid)
28        if(this.selectedMethods.length == 0) {
29          this.disabledApproaches = [];
30        }
31        this.selectedMethods.forEach(m => {
32          temp.push(...this.filteredAssessmentApproaches.filter(a =>
                a.predictionMethod != m.uuid))
33        })
34        if(this.selectedMethods.length > 1) {
35          const getUniqueDuplicates = <T>(arr: T[]): T[] => {
36            const counts = new Map<T, number>();
37            arr.forEach(item => counts.set(item, (counts.get(item) || 0)
                + 1));
38            return [...counts.keys()].filter(item => counts.get(item)! >
                1);
39          };
40          nonCorrespondingAssessmentApproaches =
              getUniqueDuplicates(temp);
41        } else {
42          nonCorrespondingAssessmentApproaches = temp;
43        }
44        this.setSelectFlags(nonCorrespondingAssessmentApproaches,
            !isChecked)
45      }
46    }
47  }
```

This function is executed whenever a checkbox of a prediction method is interacted with by the user and determines how the state of the elements should be updated based on this interaction. At the very beginning, a helper function `resetSelectFlags()` is called that ensures that all affected elements are reset to an enabled state. This ensures consistency when applying the filter logic on toggle of the checkbox, by preventing elements being disabled from previous interactions. Next, the boolean variable `isChecked` holds information about the incoming event object and if the checkbox was ticked or unticked. Additionally, a small helper function checks if the element is an actual filter option, as prediction methods are stored as filter option in the knowledge base.

The next stage begins by checking whether the element is indeed a filter option and if the checkbox has been checked. If this is the case the algorithm determines all assessment approaches that are not compatible with the checked prediction method. This is done by comparing the prediction method of any particular assessment approach with the selected prediction methods UUID. These "non-corresponding" assessment approaches form the basis of elements to be disabled. If this is the first prediction method selected by the user, all non-corresponding approaches will be added to the list of disabled approaches and directly disabled by calling the `setSelectFlags(...)` function. If, however, other approaches are already disabled and other prediction methods are already selected, the function computes the intersection between the previously disabled approaches and the new set of non-corresponding approaches. This ensures that only those approaches remain disabled that are incompatible across all selected prediction methods.

The final stage handles the case when a prediction methods becomes unselected and starts by

removing the selected prediction method from the list of selected methods. If, after this removal, no more prediction methods are selected the list of disabled assessment approaches is cleared, effectively enabling all previously disabled assessment approaches and their sub elements. If only one prediction method remains selected the non-corresponding assessment approaches will be marked as disabled, as can be seen in line 42 of Listing 6.16. If this is however not the case, the logic becomes more strict and this case is handled by the utility function `getUniqueDuplicates`. The idea of this helper function is relatively straight forward. When more than one prediction method remains selected after unselecting one prediction method, the algorithm needs to determine those assessment approaches which are incompatible with all remaining selected prediction methods. The function collects all assessment approaches that are non-corresponding to each selected prediction method and stores them temporarily. As this list may contain duplicates, because one approach can be incompatible with several of the selected prediction methods. To resolve this issue, the functions counts how often an approach appears in the temporary list and keeps only those in a list that appear more than once. This is done by using a `Map` where the assessment approach is used as the key, and the number of occurrences is stored as the corresponding value. This tells the system, that these approaches are the ones that need to be disabled, as they fail to match several selected prediction methods at the same time. Once the non-corresponding assessment approaches are determined this set is passed to the `setSelectFlags(...)` function to update the elements accordingly.

This function is responsible for enabling or disabling elements through the hierarchy of the knowledge base. It takes the non-corresponding assessment approaches and the boolean variable `isChecked` as parameters and iterates over the elements in this list. It marks the current assessment approach element as selected of deselected and continues to do the same for the corresponding methodologies and its implementations. The function is highlighted in Listing 6.17 below.

Listing 6.17: Selection Algorithm - setSelectFlags(...) function

```
1  setSelectFlags(nonCorrespondingAssessmentApproaches:
       AssessmentApproach[], isChecked: boolean) {
2      nonCorrespondingAssessmentApproaches.forEach(a => {
3        a.selected = isChecked;
4        let correspondingMethodologies = [...a.methodologies];
5        correspondingMethodologies.forEach(m => {
6          m.selected = isChecked;
7          let correspondingTools = [...m.implementations.filter((r) =>
               this.getType(r) == 'Tool')]
8          correspondingTools.forEach(t => {
9            t.selected = isChecked
10         })
11         let correspondingPackages = [...m.implementations.filter((r) =>
               this.getType(r) == 'Package')]
12         correspondingPackages.forEach(p => {
13           p.selected = isChecked
14         })
15       })
16     })
17 }
```

6.4 Results Section

The last section of the implementation perspective highlights some key implementation aspects of the **Results Section**. Lets start by going over the elements stored in the knowledge graph needed to present assessment results within the knowledge base. Listing 6.18 shows an excerpt of the knowledge graph containing these elements.

Listing 6.18: Example elements for Assessment Results

```
rprov:ADANPackageAssessment
        rdf:type          owl:Class;
        rdfs:subClassOf   rprov:Assessment;
        rprov:producedBy  rprov:Realization .

rprov:ClassificationCase
        rdf:type         owl:Class;
        rdfs:subClassOf  rprov:AssessmentCase;
        prov:wasAssignedToDeploymentEntity
                rprov:Assessment .

<urn:uuid:f735b4be-227f-4ccb-8f96-21e8da18b124>
        rdf:type               rprov:ADANPackageAssessment;
        rdfs:label             "Genome Prediction Case #1 - Adan"@en;
        rdfs:resource          "adan_tryout.csv"@en;
        rprov:producedBy
           <urn:uuid:7d57b432-5042-493f-8c3c-f5b7911794de>;
        rprov:resourceContent  ""@en;
        prov:generatedAtTime   "2025-06-20T13-22-05"^^xsd:dateTime .

<urn:uuid:85376e74-6318-4690-abdb-ef0804aecee6>
        rdf:type          rprov:ClassificationCase;
        rdfs:label        "Genome Prediction Case #1 - Adan"@en;
        rprov:wasAssignedToDeploymentEntity
                <urn:uuid:f735b4be-227f-4ccb-8f96-21e8da18b124>;
        prov:endedAtTime  "2025-06-20T13-22-05"^^xsd:dateTime .
```

The elements shown in the RDF representation above store one specific assessment result for the ADAN package. Important to note is that the two elements on top of the listing represent elemetns which are created as a product of adding elements of another type to the knowledge graph. The `rprov:ClassificationCase` element for example, a subclass of `rprov:AssessmentCase` was generated when the prediction method `Classification` was added to the knowledge base. Similar to that, the `ADANPackageAssessment` that is a subclass of `rprov:Assessment` was created when the implementation for the ADAN package was inserted to the knowledge base. An example of this can be found in Listing 6.19 below, which shows a previously omitted excerpt of the SPARQL query used for inserting a new implementation into the knowledge graph, discussed in the chapter before.

Listing 6.19: Excerpt of insert new Implementation query to add Assessment

```
query.append(String.format("""
    INSERT DATA {
```

```
3       rprov:%sAssessment a owl:Class;
4       rdfs:comment '%s'@en ;
5       rdfs:subClassOf rprov:Assessment;
6       rprov:producedBy rprov:Realization;
7     }
8     """, implementation.getLabel().replaceAll(" ", ""),
          implementation.getComment()));
```

The placeholder %s in the query gets replaced by the implementations label and the description, modeled as rprov:comment provided by the user. The reason these elements are created implicitly is because every assessment case is tied to a prediction method and every assessment is tied to a specific implementation, also highlighted by the rprov:producedBy property of an assessment.

7

Conclusion

The goal of this thesis was to design and implement a system that supports the structured reliability assessment of predictive analytic results by combining established assessment approaches, methodologies, and implementations within an extensible knowledge base. Based on the reference process by Staudinger et al. (2024), this work features a knowledge graph at its persistence layer, connected by a Spring Boot backend and Angular frontend. A design such as this, ensures flexibility in storing data as well as providing end users with an intuitive interface.

The results of this work demonstrate that the integration of assessment approaches into a knowledge graph structure allows for consistent representation of the available data in the knowledge graph as well as for efficient retrieval and filtering. By modeling dependencies such as prediction models for assessment approaches and various filter options, such as required, recommended, and prohibited the system supports both a guided navigation as well as free browsing of the elements contained within the knowledge base. Both the guidance and the free navigation, facilitate informed decision making in the choice of reliability assessment techniques that fit specific characteristics of the predictive analytic results to be assessed. These characteristics can include but are not limited to a specific domain, a data type or the availability of training data. The developed user interfaces, the **Guided Section**, the **Browsing Section**, and the **Results Section**, show that abstract entities can be presented in an intuitive way to the user. Together, they show that a graph-based implementation can connect theoretical constructs with practical usage when assessing the reliability of predictive analytic results.

Future work to this thesis may include the development of more advanced recommendation mechanisms to enable a more fine grained filtering of the available assessment techniques. Furthermore, integration of more methodologies and corresponding tools would greatly increase the applicability in domains not currently supported within the knowledge base. Lastly, usability studies with domain experts could be conducted, as to receive feedback on how good the system is in supporting decision-making for choosing a suitable reliability assessment approach.

In summary, this master's thesis demonstrates the design and implementation of a knowledge base that provides structured, user-oriented, and extensible access to approaches for assessing the reliability of predictive analytic results. It shows that a knowledge graph can serve as an extensible foundation for organizing reliability assessment techniques. It not only supports transparent navigation and filtering but also provides a basis for future extensions, such as more advanced recommendation mechanisms or closer integration with more external methodologies and corresponding implementations.

Bibliography

Badzura, P. O. (2023). Design and implementation of a web-based user interface for the guided assessment of reliability of classification results using the perturbation approach. https://resolver.obvsg.at/urn:nbn:at:at-ubl:1-65909

Carrió, P., Pinto, M., Ecker, G., Sanz, F., & Pastor, M. (2014). Applicability Domain Analysis (ADAN): A Robust Method for Assessing the Reliability of Drug Property Predictions [Publisher: American Chemical Society]. *Journal of Chemical Information and Modeling*, *54*(5), 1500–1511. https://doi.org/10.1021/ci500172z

El Hachimi, C., Belaqziz, S., Khabba, S., & Chehbouni, A. (2022). Data Science Toolkit: An all-in-one python library to help researchers and practitioners in implementing data science-related algorithms with less effort. *Software Impacts*, *12*, 100240. https://doi.org/10.1016/j.simpa.2022.100240

Getting Started | Building REST services with Spring. (n.d.). Retrieved September 22, 2025, from https://spring.io/guides/tutorials/rest

Grimm, S. (2009). Knowledge representation and ontologies. In *Scientific data mining and knowledge discovery: Principles and foundations* (pp. 111–137). Springer.

Han, J., Pei, J., & Tong, H. (2022). *Data mining: Concepts and techniques*. Morgan kaufmann.

Myers, P. D., Ng, K., Severson, K., Kartoun, U., Dai, W., Huang, W., Anderson, F. A., & Stultz, C. M. (2020). Identifying unreliable predictions in clinical risk models [Publisher: Nature Publishing Group]. *npj Digital Medicine*, *3*(1), 8. https://doi.org/10.1038/s41746-019-0209-7

Pacha, N. H., Khebazi, F. Z., & Mazouz, N. (2021). Data Mining and Its Contribution to Decision-Making in Business Organizations [Num Pages: 14]. In *Big Data Analytics*. Apple Academic Press.

Peng, C., Xia, F., Naseriparsa, M., & Osborne, F. (2023). Knowledge Graphs: Opportunities and Challenges. *Artificial Intelligence Review*, *56*(11), 13071–13102. https://doi.org/10.1007/s10462-023-10465-9

RDF 1.1 Primer. (n.d.). Retrieved August 11, 2025, from https://www.w3.org/TR/rdf11-primer/

Rushing, J., Ramachandran, R., Nair, U., Graves, S., Welch, R., & Lin, H. (2005). ADaM: A data mining toolkit for scientists and engineers. *Computers & Geosciences*, *31*(5), 607–618. https://doi.org/10.1016/j.cageo.2004.11.009

Schröer, C., Kruse, F., & Gómez, J. M. (2021). A Systematic Literature Review on Applying CRISP-DM Process Model. *Procedia Computer Science*, *181*, 526–534. https://doi.org/10.1016/j.procs.2021.01.199

SPARQL 1.1 Overview. (n.d.). Retrieved August 11, 2025, from https://www.w3.org/TR/sparql11-overview/

Staudinger, S., Schuetz, C. G., & Schrefl, M. (2024). A Reference Process for Assessing the Reliability of Predictive Analytics Results. *SN Computer Science*, *5*(5), 563. https://doi.org/10.1007/s42979-024-02892-4

Wirth, R., & Hipp, J. (2000). CRISP-DM: Towards a standard process model for data mining | Request PDF. *ResearchGate*. Retrieved September 10, 2025, from https://www.researchgate.net/publication/239585378_CRISP-DM_Towards_a_standard_process_model_for_data_mining

Yao, P., Kosmajac, M., Waheed, A., Guzhva, K., Hervieux, N., & Barbosa, D. (2023, May). NLP Workbench: Efficient and Extensible Integration of State-of-the-art Text Mining Tools. In D. Croce & L. Soldaini (Eds.), *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations* (pp. 18–26). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.eacl-demo.3