UNIVERSITÄT
LEIPZIG

Faculty of Mathematics & Computer Science
Practical Computer Science
Business Information Systems

Master's Thesis

A thesis in partial fulfillment of the requirements for the degree *Master of Science* in Computer Science.

Visualizing & Manipulating RDF Graphs within the Kanban Paradigm

A Prototypical Implementation of an RDF Kanban Board

Steven Kalinke

[redacted]

First Supervisor Dr. rer. nat. Michael Martin, *Leipzig University*
Second Supervisor Dr. rer. nat. Sebastian Tramp, *eccenca GmbH*

December 20, 2019

Abstract

In the field of Semantic Web, much effort is invested in developing possible solutions for exploring and managing graph data in a visual context. eccenca's Corporate Memory, an enterprise application suite, enables users to work with semantic models and allows intuitive data exploration. The current work uses eccenca's software infrastructure to develop a novel approach to visualize RDF resources by mapping graph data in a Kanban board. Based on an RDF configuration graph, users can select the resources represented as the cards of the Kanban board, as well as the property used to represent the columns of the board. In addition to this visualization, the developed prototype allows to modify a resource by the prior selected column property when moving a card between columns on the board. Dropping a card to a novel column triggers the resource to update its property to the value of this column. Visualizing and manipulating knowledge data by relocating cards on a Kanban board represents a innovative approach in the field of semantic data exploration.

Keywords Kanban Board, React, JavaScript, Semantic Web, SPARQL, RDF(S), OWL, SHACL

Contents

List of Figures

List of Tables

List of Code

List of Prefixes & Abbreviations

`dbug:`	`http://ontologi.es/doap-bugs#`
`dct:`	`http://purl.org/dc/terms/`
`foaf:`	`http://xmlns.com/foaf/spec/#`
`owl:`	`http://www.w3.org/2002/07/owl#`
`rdf:`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
`rdfs:`	`http://www.w3.org/2000/01/rdf-schema#`
`rmb:`	`https://vocab.eccenca.com/rmb/`
`sh:`	`http://www.w3.org/ns/shacl#`
`skos:`	`http://www.w3.org/2004/02/skos/core#`
`uneskos:`	`http://purl.org/umu/uneskos#`
`vs:`	`http://www.w3.org/2003/06/sw-vocab-status/ns#`
FOAF	Friend Of A Friend
IRI	Internationalized Resource Identifier
JSON	JavaScript Object Notation
JSX	Javascript XML
OWL	Web Ontology Language
RDFS	Resource Description Framework Schema
RDF	Resource Description Framework
RMB	Resource Management Board
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locators
UX	User Experience

1 Introduction

1.1 Motivation & Objective

Modifying resources by a specific property within the graph-based Resource Description Framework (RDF) can be an exhausting and expensive task. This is particularly the case when managing graph resources in plain text formats (e.g., in Turtle or SPARQL), as it requires trained people to maintain the data, and, despite their expertise, it would still be prone to human error. To overcome this challenge, much effort has been previously invested in developing possible solutions for managing graph data in a visual context. One solution to manage semantic and metadata is the software solution *Corporate Memory* developed by eccenca GmbH in Leipzig, Germany. *Corporate Memory* consists of three core components, namely (1) *DataIntegration*, (2) *DataPlatform*, and (3) *DataManager*. The latter component provides a comprehensive visual representation of a knowledge base and allows intuitive authoring of semantic content.

The aim of the current thesis was to create a prototypical component for the *DataManager*, that provides an approach to address the issue of modifying resources by a specific property in an intuitive way. The prototype (throughout the work also referred to as *Resource Management Board* (RMB)), addresses two main goals, that are documented in the current thesis: (1) visualizing a certain section of a knowledge graph (i.e., a subgraph) within a Kanban board, and (2) the possibility of modifying a specific property by dragging a resource into another column.

The following example captures the essence of both goals and illustrates the transformation process. Code 1.1 illustrates a minimal RDF graph. In simple terms, the graph expresses that the material *marble* has a *color* that is set to *white*.

```
1  <http://dbpedia.org/page/Marble>
2    <http://dbpedia.org/property/color>
3      <http://dbpedia.org/page/White> .
```

Code 1.1: A minimal graph in RDF.

Typically, (directed) graphs are visualized by a set of ellipses that are interconnected by arrows. Ellipses represent the graph's nodes, while arrows describe the graph's edges. Figure 1.1 visualizes Code 1.1 as a directed graph:

Figure 1.1: A minimal graph visualizing Code 1.1.

In the previous example, the graph's property is *color*, and its current value is a resource that refers to *white*. In order to change the value of the property from white to red, the graph could be manually modified, for example, by a SPARQL query. However, as mentioned above, this approach is prone to error in many ways.

In basic terms, this work merges the concept of RDF with the concept of a Kanban board. Specifically, a particular portion of an RDF graph will be selected and mapped into a Kanban board. Furthermore, from that portion, a particular property will be selected to represent the columns of the board. Finally, the board will allocate its content (i.e., resources) over these columns.

Figure 1.2 depicts a mockup version of a Resource Management Board. In addition to the data from the previous example, this illustration contains one more resource (i.e., the material copper) and embeds both elements in a broader domain (i.e., a material database, which is also used as the board's title). Since *color* was selected as the preferred column property, the contents of the board have been structured accordingly. This means that all resources are grouped by their inherent color property. Therefore, the resource marble was initially placed in the column labeled white. In contrast to Figure 1.1, the visual projection of graph data, as seen in Figure 1.2, is a novel approach for semantic data exploration.

In addition to visualize graph data, the second goal of this thesis is the modification of a property's value by dragging a resource into another column. To illustrate this goal, the resources in Figure 1.2 (i.e., marble and copper) are depicted as draggable cards that can be moved arbitrarily over the columns of the board. As depicted by the arrow, the card holding the resource for marble is getting dragged from the first to the second column. Eventually, when dropping the card to its target column, an update on the underlying graph gets triggered, which will assign the corresponding column value (i.e., red) to that specific property (i.e., color) on that particular resource (i.e., marble). Using the drag and drop capabilities of a Kanban board to manage specific properties of a resource is a novel and user-friendly approach to manage knowledge data. Without requiring profound expertise in the field of RDF, it enables users to easily and intuitively manage complex graph structures.

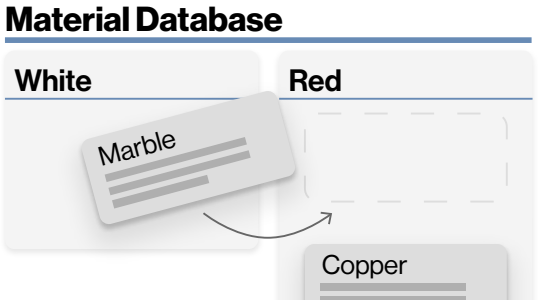

Figure 1.2: Mockup of the Resource Management Board.

1.2 Structure of This Work

This thesis is structured as follows. First, chapter 2 provides a theoretical background for the two main concepts involved in this work: (1) Semantic Web with the focus on the RDF and (2) Kanban. Then, chapter 3 introduces four use cases and derives corresponding functional and non-functional requirements. Thereupon, chapter 4 presents an overview of existing Kanban board solutions and evaluates them with regard to the desired requirements. Hereafter, chapter 5 provides specifications for the prototype, including the target data model and query strategy. In chapter 6, the corresponding implementation is outlined in detail. Lastly, chapter 7 provides an overall evaluation of this implementation, based on the outlined requirements. Moreover, the limitations of the current status of the prototype will be discussed, together with possible future directions.

2 Background

People create all sorts of digital content in a variety of formats and on a massive scale. Regardless of how information is shaped, we—as humans—are capable of extracting the inherent semantics by inferring implicit knowledge about a given information. For example, a piece of information represented in a textual form is: *The marble is white*. We probably understand the meaning of that statement, as we know that marble refers to a material and white to a color. Even though this additional information was not provided explicitly to us, we can infer that knowledge implicitly, to understand the meaning of the statement. In addition to this one possible meaning, however, this sentence may also refer to a toy: a marble. Ultimately, the context in which this statement is embedded would resolve this ambiguity for a human reader. Nevertheless, this example demonstrates the way we link information. In fact, our brain consists of neurons (analogous to graph nodes) interconnected by synapses (analogous to graph edges), creating a neural network, which is a complex graph structure (Stanley et al., 2013, p. 1). Linking, interpreting, and evaluating information may seem trivial to us since our brain has evolved to perform cognitively demanding tasks on a daily basis. For a computer, in contrast, understanding semantics is a complex task, as it is not capable of inferring knowledge by default. Making a computer understand (especially ambiguous) semantics thus represents a challenging endeavor.

2.1 Semantic Web

The research field *Semantic Web* (or *Linked Data*) aims to address the previous matter. The basic idea is to enrich information with resource identifiers that link to a distinct entry in a catalog (i.e., a vocabulary/ontology[1]). This link allows computers (and humans) to understand what a resource is referring to. There is no need to provide more context, as linked data grants semantic uniqueness by design. Linked data uses the graph-based Resource Description Framework (RDF) as the underlying data model, which broadly describes the relationship between resources (E. Miller & Schloss, 1997).

An RDF graph consists of three components, which can be expressed as a semantic triple: *subject-predicate-object*. To illustrate the previous statement, the semantic triple for white marble would translate to *marble-hasColor-white*. In RDF, the subject designates a specific resource (e.g., *marble*), while the predicate denotes a specific property of that subject (e.g., *hasColor)*. Furthermore, a predicate describes the relation between a subject and an object. However, depending on the domain and the language being used in this field, there are different ways to express a triple. While some authors (e.g., Fensel, 2005, p. 115 and Khosrow-Pour, 2006, p. 581) refer to *object-property-value* triples, the EAV model expresses RDF as an *entity-attribute-value* triple. Powers (2003, p. 17) concludes that

> "[…] simple facts can almost always be defined given three specific pieces of information: the subject of the fact, the property of the subject that is currently being defined, and its associated value. This correlates to what we understand to be a complete thought, regardless of differing syntaxes based on language."

[1] Disambiguation of vocabulary and ontology: *"There is no clear division between what is referred to as 'vocabularies' and 'ontologies'. The trend is to use the word 'ontology' for more complex, and possibly quite formal collection of terms, whereas 'vocabulary' is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web."* (W3C, n.d.)

Subjects and predicates always use a resource identifier (i.e., a URI/IRI)[2] to describe their exact entity and property. Objects, on the other hand, may either refer to a URI as well or to a (string) literal.[3] To disambiguate the marble example, URIs can be used to express the meaning distinctly, which was demonstrated in the previous chapter (Introduction) in Code 1.1, where the URI referred to marble as a material and not the toy.

As stated, RDF intrinsically represents a graph structure. Moreover, it is a *"[…] good example of a labeled directed multigraph […]"* (Shaposhnik et al., 2015, p. 21). To illustrate this concept, Figure 2.1 depicts a graph expressing the geological classification of the material marble.

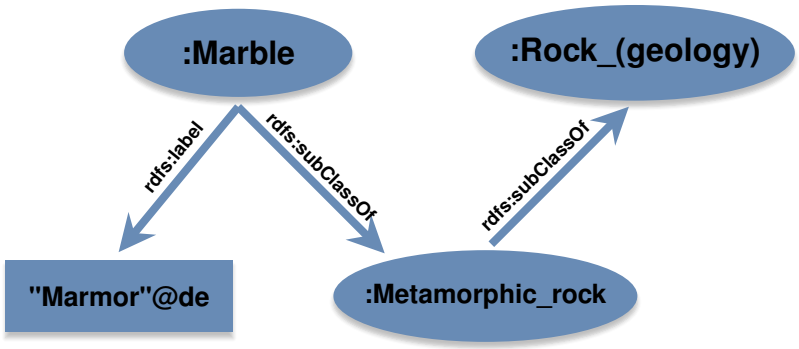

Figure 2.1: A labeled directed RDF multigraph, expressing the broader classification of the material marble, and stating marble's (German) label as an annotated object literal.

In a directed (RDF) graph, arrows are reflecting the graph's directional nature, and objects, as illustrated, may refer to other resources, creating new semantic triples. These features do not only allow RDF to describe logical relations between things but also to create complex frameworks working on top of it. Relevant for this work are frameworks defining a syntax, structure, or shape (e.g., RDFS,[4] OWL,[5] SHACL,[6] or different vocabularies/ontologies defining specific terms (e.g., FOAF,[7] DCT,[8] etc.).

A notable application in the field of linked data is DBPedia (Bizer et al., 2009), which extracts structured information from the Wikipedia project and allows users to query this information. Similar to the preceding examples of graph data, Code 2.1 is using resources referring to DBPedia. The following code is representing Figure 2.1 in Turtle notation.[9]

```
1  <http://dbpedia.org/resource/Marble>
2    rdfs:subClassOf
3      <http://dbpedia.org/resource/Metamorphic_rock> ;
4    rdfs:label "Marmor"@de .
5  <http://dbpedia.org/resource/Metamorphic_rock>
6    rdfs:subClassOf
7      <http://dbpedia.org/resource/Rock_(geology)> .
```

Code 2.1: A minimal graph in Turtle notation representing Figure 1.1.

[2] Throughout this work, I will interchangeably use both abbreviations URI (Uniform Resource Identifier) and IRI (Internationalized Resource Identifier). The latter extends the characters in URIs from a subset of the ASCII character set to almost all characters of the Universal Character Set (Unicode/ISO 10646).
[3] In RDF objects (and thus subjects) may also be a blank node; however, this has no further relevance for this work.
[4] Resource Description Framework Schema by Brickley & Guha, 1999
[5] Web Ontology Language by McGuinness & Harmelen, 2004
[6] Shapes Constraint Language by Knublauch & Kontokostas, 2015
[7] Friend Of A Friend by L. Miller & Brickley, 2014
[8] Dublin Core Metadata Terms by the DCMI Usage Board, 2002
[9] Code listings in this work are omitting the prefix notation shared by N3, Turtle, and SPARQL. In the digital version of this work, clicking a prefixes will link to the corresponding IRI on page VI at the beginning of this work.

2.2 Kanban

The Kanban system was invented by Taiichi Ohno, an industrial engineer at Toyota in 1947. It establishes a just-in-time method of inventory control. The word Kanban, in its literal translation (jap. 看板), describes a cardboard, which represents one of the core components of the Kanban system.

In general, a *Kanban board* visualizes work in progress (WIP) items that are referred to as *cards* on the board. In most scenarios, cards are flowing from left to right over the *columns* of a board, indicating their current stage of progress. Boards may also be divided into horizontal *swimlanes* (or just *lanes*), which add another container to categorize groups of cards (e.g., different teams performing the work).

2.2.1 Board Anatomy

Figure 2.2 illustrates the evolving stages of a Kanban board by gradually adding more features to the first board (A). While all boards share the same three labels for their columns (i.e., *ToDo*, *Doing*, *Done*), they differ in their card and lane structure. In particular, board (A) consists of five cards, board (B1) adds three more cards that belong to another domain or *class*, and lastly, board (B2) extends (B1) by separating the cards into their classes using swimlanes (i.e., *Box X* and *Box Y*).

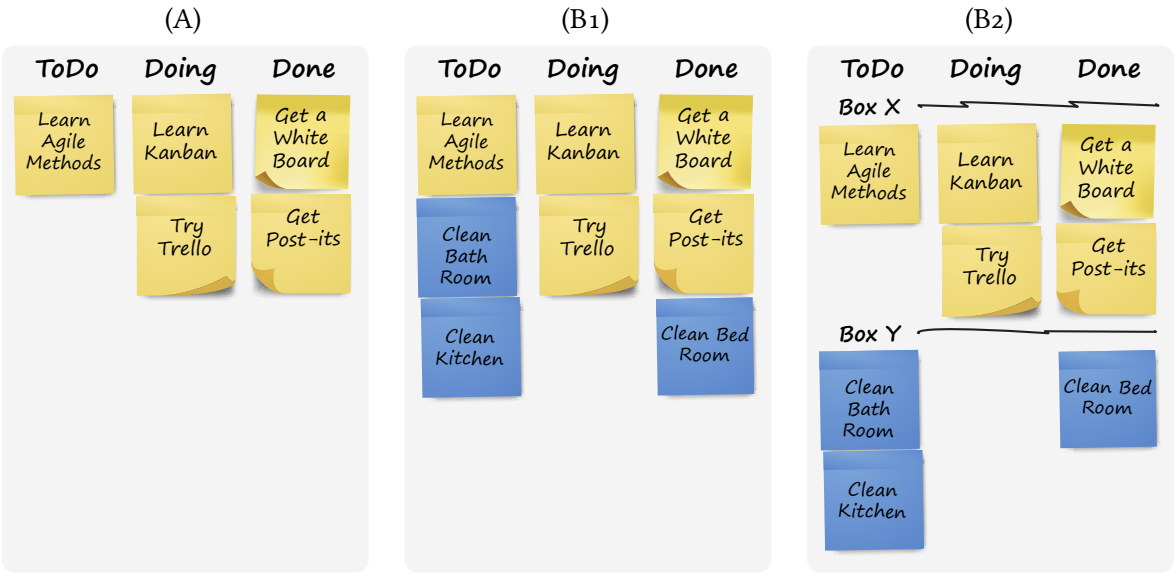

Figure 2.2: Three examples of a Kanban board, with different board components (i.e., cards, columns, and lanes). Cards are depicted by post-it notes. Board (B1) contains cards with mixed classes, board (B2) is separating card classes using swimlanes (i.e., *Box X* and *Box Y*).

There are two perspectives on how a Kanban board can be conceptualized, that is (1) by their *board component resources* and (2) *card component resources*. Both perspectives will be repeatedly referenced throughout this work.

Board component resources refer to the *class* of each structural component (i.e., cards, columns, and lanes). In other words, it refers to the origin of each board component. For example, in board (A) in Figure 2.2, all the post-it notes may be stored in a storage box labeled *Box X*. This storage box would represent the *class* of the cards. Principally, one board can carry cards from multiple classes. For example, a second storage box labeled *Box Y* could contain various post-it notes referring to a different class. Board (B1), as depicted in Figure 2.2, provides an example of a board keeping track

of two classes: a person's cleaning progress and their Kanban achievements. Both classes fit the semantics of the board columns (| *ToDo* | *Doing* | *Done* |). However, when using multiple card classes, it may be helpful to clearly separate cards into their classes by using swimlanes, as depicted by board (B2).

In contrast to cards, columns and lanes typically have one single class each. The column labels in Figure 2.2, for example, could belong to a storage box labeled *Basic ToDo Progress Labels*. Let us assume that one would like to add another box, labeled *Weekplaner* containing column labels ranging from Monday to Friday, containing column labels ranging from Monday to Friday. Merging both classes *Basic ToDo Progress Labels* and *Weekplanner* would require a more complex (three-dimensional or nested) board solution, lacking in user-friendliness and applicability.

Card component resources refer to the resources that are depicted on a card (i.e., their content). Compared to the tight boundaries of board component resources, card component resources are virtually endless regarding the number of displayable elements. The text on the post-it notes in Figure 2.2, for example, can be considered as the card's title resource. Principally, cards may further depict a descriptive text, a due date, a creation date, or the name of an assignee.

2.2.2 General Board Usage

Building upon the idea of the traditional paper-pencil boards (e.g., Figure 2.2), a variety of digital Kanban solutions have been developed.[10] Even though these solutions vary in their scope and price models, they all share a basic feature set: Most apparent, all solutions allow users to create cards and columns from scratch. Furthermore, cards contain at least a title and may also have a description field to provide more context to the user. Lastly, by definition, cards can be moved from one column to another to indicate their current stage of progress.

Due to the vast array of application fields (e.g., personal task management, marketing teams, human resources, etc.), there are countless ways to design a Kanban board. Notably, in the field of agile software development, Kanban had a particularly strong impact over the last two decades (Stoica et al., 2016, p. 11).

Over the course of a project, a Kanban board has the potential to grow in complexity (i.e., more cards, columns, and lanes). There are established principles to maintain a lean state to tackle this case. Two of the most prominent are: (1) define a maximum card limit for a board to prevent your team from exceeding their capacity, and (2) *walk the board* from right to left. That means although cards usually flow from left to right, *"[…] you iterate over the tickets from right to left: Closest to completion to most recently started."* (Anderson, 2016).

Nevertheless, in this work, the Kanban board is used to visualize a section of an existing RDF graph and to mutate a specific value by relocating cards on the board. In other words, and in contrast to most Kanban applications, the focus is not on creating RDF resources within the board, but on visualizing and managing existing ones. Note that throughout this thesis, I will use the terms *resource* and *card* interchangeably, since—in this shared context—a card always refers to an RDF resource. For example, the *DBpedia* entry for marble is a resource and thus a card in the board, as illustrated in Figure 1.2.

[10] Wikipedia lists some examples: `https://en.wikipedia.org/wiki/Kanban_board#Notable_tools`.

3 Requirements

This chapter provides an overview of use cases, user stories, and requirements for the current project. In general terms, a *use case* describes a specific usage scenario for a software product and carries a variety of software *requirements*, which, in turn, describe a specific functionality demanded by a stakeholder (e.g., a user). A *user story* is similar to a requirement. It also describes the request by a certain persona; however, in contrast to a requirement, it is composed of natural language. Jacobson et al. (2011, p. 5) describes the relation between the three terms as follows:

> "To understand a use case we tell stories [...] Use cases provide a way to identify and capture all the different but related stories in a simple but comprehensive way. This enables the system's requirements to be easily captured, shared and understood."
>
> Jacobson et al., 2011, p. 5

The field of requirements engineering differentiates between many different types (or categories) of requirements. The current work focuses on two main types, namely functional and non-functional requirements. As defined by the ISO/IEC/IEEE's *Systems and Software Engineering Vocabulary*, a functional requirement is: (1) *"a statement that identifies what a product or process must accomplish to produce required behavior and/or results"* and (2) *"a requirement that specifies a function that a system or system component must be able to perform"* (see ISO/IEC/IEEE, 2010, p. 301). A non-functional requirement, on the other hand, is *"a software requirement that describes not what the software will do but how the software will do it."* (see ISO/IEC/IEEE, 2010, p. 231). To give an example: An crucial functional requirement of an elevator is to transport things from one floor to another. A non-functional requirement, in contrast, might be at what speed the elevator should perform this task. In the domain of software development, typical examples for non-functional requirements include the performance of an application, its response times, reliability, aspects around documentation, and in-house coding style guides.[11]

Generally, user stories describe requirements. In contrast to requirements, however, they are composed of natural language and use a predefined template (see Ambler, 2014). There is a variety of templates that can be used to create a user story. The pattern used by eccenca and throughout this work is constructed as follows: "In order to *<benefit/outcome>*, as a *<persona/role>*, I want to *<description>*." Additionally, user stories can be marked with story points, which aim to predict the level of complexity (Ambler, 2014). However, according to various online sources, vain efforts have been previously made regarding the use of story points (see, e.g., Jailall, 2018, Kerievsky, 2012, Krimmer, 2017). As major challenges, most articles report the frustrating attempt to estimate the time and complexity of story points. As a consequence, story points needed to be repeatedly re-evaluated during the development phase.

To avoid these challenges, this chapter focuses on the use of *Requirement Levels* (see Bradner, 1997), which—in contrast to story points—aims for the actual importance of a specific requirement rather than measuring time/complexity. Table 3.1 provides an overview of Bradner's five requirement levels.

[11] An extensive list of categorized examples can be found here: `https://dalbanger.wordpress.com/2014/01/08/a-basic-non-functional-requirements-checklist/`.

Key Word *Synonyms*	Meaning
MUST *REQUIRED, SHALL*	*"[…] absolute requirement of the specification."*
SHOULD *RECOMMENDED*	*"[…] the particular behavior is acceptable or even useful […]"*
MAY *OPTIONAL*	*"[…] an item is truly optional. […]"*
SHOULD NOT *NOT RECOMMENDED*	*"[…] ignore a particular item […]"*
MUST NOT *SHALL NOT*	*"[…] absolute prohibition of the specification."*

Table 3.1: Requirement Levels (Bradner, 1997, p. 1)

Finally, a use case contains a variety of requirements and user stories, equally. Use cases explore different fields of application and describe a broader goal. According to Burris (n.d.), a use case is *"… a narrative description of a goal-oriented interaction between the system under development and an external agent."* Another purpose of a use case is to demonstrate the benefits of the software product.

The following section 3.1 describes four use cases demonstrating application fields of the prototype. In the two succeeding sections, functional (section 3.2) and non-functional requirements (section 3.3) will be derived from the presented use cases. Eventually, the last section in this chapter, section 3.4, provides a summary of the described requirements.

3.1 Use Cases

All use cases utilize either constructed or existing RDF data to demonstrate their intentions, and each use case contains the following four subdivisions: (1) an outline describing the purpose of the current use case, (2) the board component resources (i.e., the resources for cards, column, and, if applicable, lanes) providing information about the used classes and domains for each structural component, (3) the card component resources describing what elements will be depicted on the cards, and (4) a mockup of the RMB depicting the current use case.

At this stage, it is worth mentioning that there are three independent features shared by all use cases: (1) All cards should display their resource identifier (i.e., their URI), to provide an easy look-up reference to the user, (2) when clicking a card, more information about that specific resource should be revealed, and (3) a timestamp property (i.e., the *last modification date*) should be stored within a resource whenever a card gets dropped to a new column. Furthermore, all cards containing such a timestamp property should also display it by default.

3.1.1 Ontology Management

Each of the following two use cases apply a different ontology (i.e., FOAF and UNESCO) to demonstrate two ways a user can approach this scenario. To manage the ontologies by a particular status, both scenarios use the property `vs:term_status`, which values indicate *"the status of a vocabulary term, one of 'stable', 'unstable', 'testing' or 'archaic'"* (L. Miller & Brickley, 2014).

Use Case 1: Update a FOAF Term Status

In this use case, the *Friend Of A Friend* (FOAF) ontology acts as the underlying graph.[12] Throughout this scenario, the current specification will be used for reference: `http://xmlns.com/foaf/spec/`.

Outline

FOAF terms can describe individuals in various ways. For example, terms like `foaf:name`, `foaf:depiction`, and `foaf:knows`, are typically used to describe individuals by their name, photo, and relations to other individuals. Each term contains several properties, and one property shared by all terms is `vs:term_status`. For example, the term `foaf:mbox` (describing a personal mailbox) has a status value of *stable*,[13] while the status value of `foaf:depiction` is *testing*.[14]

Regarding the current use case, the first goal is to visualize all FOAF terms distributed over the board's columns, depending on their inherent status value. The second goal is to change a term's status value by dragging a card (i.e., a FOAF term) to another column. The user can easily change the status of the *depiction* resource from *testing* to *stable* by dragging the corresponding card. This will trigger a graph update, that makes changes persistent.

Board Component Resources

Cards. The FOAF terms are representing the cards of the board. Since the vocabulary definitions are written in RDF/OWL (L. Miller & Brickley, 2014), one could retrieve all FOAF terms ($n = 75$) when using the following card classes: `owl:Class`, `owl:ObjectProperty`, and `owl:DatatypeProperty`.

Columns. To manage the status of a FOAF term, the property `vs:term_status` will be used.

Lanes. To provide a clearer structure, `rdf:type` will be used to distribute all FOAF terms over the lanes of the board. Introducing lanes in this scenario affects the board's structure, similar to the transition from board (B) to (B*) in Figure 2.2 on page 5.

Card Component Resources

The FOAF terms should represent the titles of the cards. For example, the label of the term `foaf:mbox` is *personal mailbox*, and should be used as the card's title. Moreover, all FOAF terms carry an `rdfs:comment` property along with a descriptive value. These values should be displayed below the card's title to provide more context to the user. For example, the descriptive text for term `foaf:mbox` is: *"A personal mailbox, ie. an Internet mailbox associated with exactly one owner, the first owner of this mailbox."* (see the FOAF specification).

Mockup

Figure 3.1 provides a mockup of the RMB with the contents requested by the board component and card component resources above. For demonstration purposes, the mockup showcases only four FOAF terms (i.e., *Document*, *personal mailbox*, *knows*, *depiction*), including their corresponding description (see current specification for reference). Due to this limited sample size, the prototype will only reflect the content it is aware of. In other words, there are no columns values depicted for the statuses

[12] FOAF describes *"[…] persons, their activities and their relations to other people and objects […]"* (Gargouri, 2010, p. 9). The ontology was created in mid-2000 by L. Miller & Brickley (2014). FOAF Homepage: `http://www.foaf-project.org/`.

[13] Compare `http://xmlns.com/foaf/spec/#term_mbox`.

[14] Compare with `http://xmlns.com/foaf/spec/#term_depiction`.

unstable and *archaic*, since no resources are containing these values within this small sample. To give users the possibility to create new columns from scratch, it would require a solution allowing to enter an arbitrary string literal. The text boxes in Figure 3.1 depict an exemplary solution of such a feature (i.e., the *new column value* field).

Nevertheless, as stated in the outline, all resources have been allocated according to their inherent status value, resulting in a prototype consisting of two columns. Furthermore, the content is distributed over three lanes by their broader domain (i.e., `rdf:type`, as requested by the board component resources). Moreover, the user is be able to intuitively change a resource's status value by placing a card into another column, as exemplarily demonstrated in Figure 3.1 for the term *depiction*.

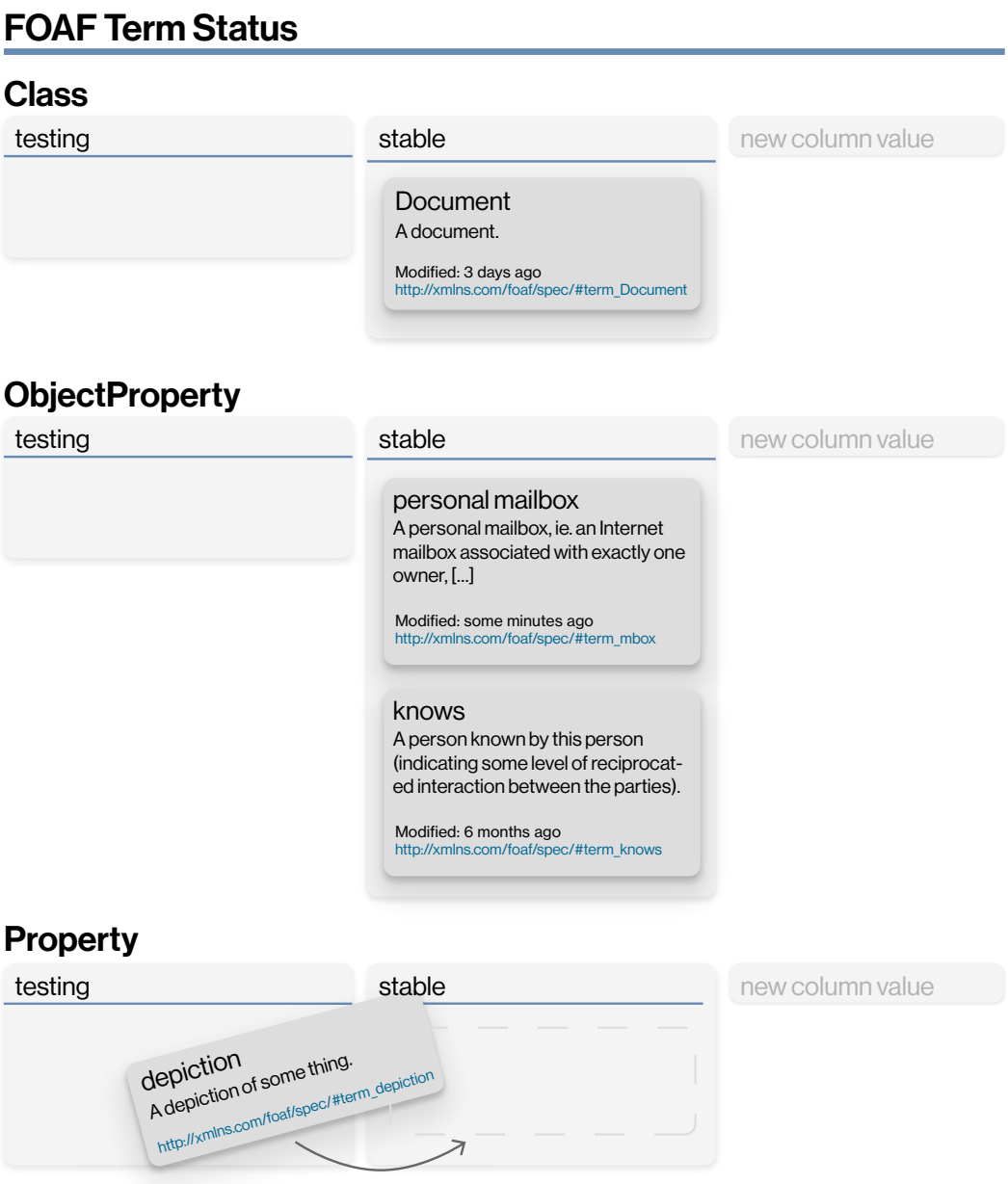

Figure 3.1: RMB Mockup of Use Case 1, showing a board that consists of two columns and three lanes grouping four RDF resources (i.e., FOAF terms). The resource *depiction* is getting dragged into another column in order to update its current term status to the value *stable*.

Note that a timestamp does not only provide information about the last point of time a user moved a card. It also provides a visual cue to tell apart cards, that have been previously 'touched', from their

untouched counterparts, as the latter ones lack a timestamp. For example, in Figure 3.1, the resource *depiction* has never been moved; thus, it does not contain a timestamp property. Eventually, when dropping the card, a timestamp gets generated, stored in the corresponding resource, and depicted on the card.

Use Case 2: Create an UNESCO Term Status

This use case is part of the scenario of managing an ontology, and it captures the special behavior of retrieving resources that lack the requested column property. This condition may either be a deliberate choice by users—as they aim to create the desired property values from scratch—or an accident due to a false configuration. In either way, the prototype should react appropriately to handle this case.

In this use case, the UNESKOS ontology (Pastor-Sanchez, 2015) acts as the underlying graph. UNESKOS is the SKOS[15] version of the UNESCO thesaurus.[16] The content of this vocabulary is built hierarchically, and its first level consists of seven major subject domains[17] containing over 4,000 terms (or concepts). For example, the concept *Deforestation* can be retrieved when traversing the graph by:

UNESCO Thesaurus ⟶ *Politics, law and economics* ⟶ *Agriculture* ⟶ *Deforestation*

Outline

Unlike the previous use case, UNESKOS terms do not contain a status property. However, it may be the user's desire to define status values for resources from scratch. Since all UNESKOS resources lack the demanded `vs:term_status` property, all resources should be placed in a fallback column labeled *no property*. From there, a user can create new column values and start to assign the resources to the desired column position. Moreover, due to the vast number of UNESKOS terms, an implicit card limit should be set to prevent a stalling behavior of the user's browser.

Board Component Resources

Cards. Each term defined within the UNESCO vocabulary refers to a `skos:Concept`. Thus, each card represents a `skos:Concept`.

Columns. As stated above, the UNESKOS graph does not contain a status property; however, this scenario utilizes the status vocabulary `vs:term_status`, similar to the first use case

Lanes. Every UNESKOS term has a broader umbrella term (i.e., a hyponym) within the hierarchical structure of the thesaurus (i.e., `uneskos:memberOf`). For example, the semantic triple for the concept *Deforestation* is:

Deforestation ⟶ `uneskos:memberOf` ⟶ *Agriculture*

Thus, *Agriculture* as a single swimlane groups a variety of related UNESKOS terms.

[15] Simple Knowledge Organization System (SKOS) is a W3C recommendation *"[…] designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary"* (Garoufallou et al., 2015, p. 455).

[16] The *"UNESCO Thesaurus is a controlled and structured list of terms used in subject analysis and retrieval of documents and publications in the fields of education, culture, natural sciences, […]"* (UNESCO, 1977).

[17] These are education, science, culture, social and human sciences, information and communication, politics, law and economics, and countries and country groupings. See `http://skos.um.es/unescothes/CS000/html`.

Card Component Resources

The card titles correspond to the UNESKOS term labels. However, unlike within the FOAF vocabulary, UNESKOS terms do not contain any descriptive property. This means that all cards initially contain only their title and their corresponding URI since both items are mandatory for every card. Nevertheless, moving a card into a new column generates a timestamp property, which is also depicted on the corresponding card.

Mockup

Figure 3.2 provides a mockup of the RMB with the content requested by the board component and card component resources above. For demonstration purposes, the mockup shows only four UNESKOS concepts (i.e., *Deforestation*, *Fisheries*, *Argentina*, *Bolivia*). Although a column property is defined (i.e., `vs:term_status`), it is not existent in the UNESKOS terms. Therefore, as requested in this use case, all terms are grouped in a fallback column labeled *no property* located at the board's first column position. Two lanes are depicted by the resources' broader `uneskos:memberOf` references.

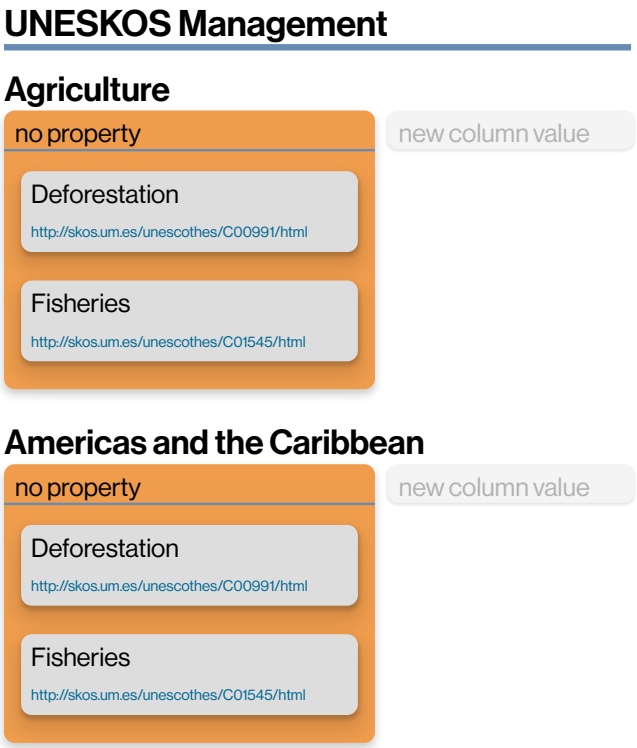

Figure 3.2: RMB Mockup of Use Case 2, showing a board that consists of two lanes, and, due to the lack of the requested column property, a fallback column. The cards only contain their mandatory title and resource identifier. Since the cards have not been moved yet, there is no timestamp depicted.

In this scenario, users can create new column values from scratch and assign resources accordingly. Eventually, when all resources have been allocated, the fallback column would vanish, as it would not hold any resource.

3.1.2 General Purpose Scenarios

Unlike the previous use cases, the following two use cases operate on a test graph to demonstrate their purpose. Moreover, they allow to depict an arbitrary amount of information on their cards.

Use Case 3: Dataset Management

This use case is applicable if a user wants to manage the state of existing datasets. The underlying graph is eccenca's *CMEM Dataset catalog*, which provides a system to manage and govern datasets and resources in eccenca's *Corporate Memory*.

Outline

In contrast to the previous use cases, these scenarios demand to display arbitrary resources on a card. For example, the *CMEM Dataset catalog* contains two resources which should be depicted on a card, if existing. That is (1) the property *version* that refers to a literal value indicating a dataset's version number or label, and (2) the property *update frequency*, which refers to a resource indicating a time interval. Ultimately, the scenario's goal is to visualize datasets and manage their predefined statuses within the RMB.

Board Component Resources

Cards. Cards represent datasets that are of type dataset. For this example, the card's class is `https://vocab.eccenca.com/dsm/Dataset`.

Columns. The board's column property is `https://vocab.eccenca.com/dsm/hasStatus` (i.e., the assigned status). The values of this property are representing the columns of the board (e.g., *needs approval*, *published*, etc.).

Lanes. Datasets are part of a broader domain; for example, a dataset containing personal data may belong to the field of human resources. A property that expresses its affiliation in the context of dataset management is `http://www.w3.org/ns/dcat#theme` from the *Data Catalog Vocabulary* (DCAT) (Erickson et al., 2014), which will be used for this purpose.

Card Component Resources

A card's title corresponds to a dataset's `rdfs:label`. Moreover, as stated above, the cards should depict a list of property-value pairs. In this use case, *version* and *update frequency* are demanded.

Mockup

Figure 3.3 provides a mockup of the RMB with the content requested by the board component and card component resources above. For demonstration purposes, the mockup showcases only three datasets (i.e., *Personal Data*, *Sales Data*, and *RND Spendings* separated by two columns (i.e., the status of the dataset), and two lanes (i.e., the dataset's domain). In deviation from the previous use cases, the dataset management scenario stresses the application of *additional properties*, which are also depicted on the cards, if they are defined on a resource (i.e., *version* and *update frequency*).

Regarding the goals of this work, this scenario allows to display RDF datasets, and to update their status by dragging cards into different columns. For example, a user may intuitively withdraw the dataset *RND Spendings* by dragging the corresponding card from the column *published* to the column *needs approval*, as depicted below.

Figure 3.3: RMB Mockup of Use Case 3, including mandatory card elements (i.e., title and resource identifier) and optional elements (i.e., description, additional properties, modified timestamp).

Use Case 4: Issue Tracking

The last use case illustrates the common Kanban usage scenario of managing issues within the field of software development. For this purpose, the DOAP ontology will be used. DOAP (Description of a Project) aims to describe software projects and offers a variety of properties to describe different aspects of this domain (Wilder-James, 2004).

Outline

For the subject of issue tracking, DOAP provides a dedicated subset vocabulary (i.e., DOAP bugs or *dbug*), which will be used in the current use case.[18] The dbug vocabulary defines various properties and value ranges to describe different aspects around the topic of issue management. For example, an issue can be described by its `dbug:status` (e.g., new, in progress, fixed, etc.), its `dbug:severity` (e.g., trivial, major, critical, etc.), its initial `dbug:reporter`, and by other properties (see specification for references). Nevertheless, this use case aims to visualize resources (i.e., issues) by their inherent issue status value (i.e., the board's columns), and grouped by their inherent severity value (i.e., lanes).

[18] Its specification can be found here: `http://ontologi.es/doap-bugs`.

Board Component Resources

Cards. The cards' classes refer to RDF resources having their `rdfs:type` set to `dbug:issue`.

Columns. Column values are derived from a resource's `dbug:status` value.

Lanes. Lanes gather resources that share the same severity value (i.e., `dbug:severity`).

Card Component Resources

Besides depicting the title and description of the issue, the DOAP vocabulary also contains an `dbug:assignee` property, which shows the person assigned for an issue.

Mockup

Figure 3.4 provides a mockup of the RMB with the content requested by the board component and card component resources above. For demonstration purposes, the mockup showcases only five issues depicted by the cards of the board. The issues are separated by three columns (i.e., the `dbug:status` value), and two lanes (i.e., the `dbug:severity` value). Regarding the current use case, the prototype visualizes any set of RDF issues and can update an issue's status by dragging the corresponding card into another column.

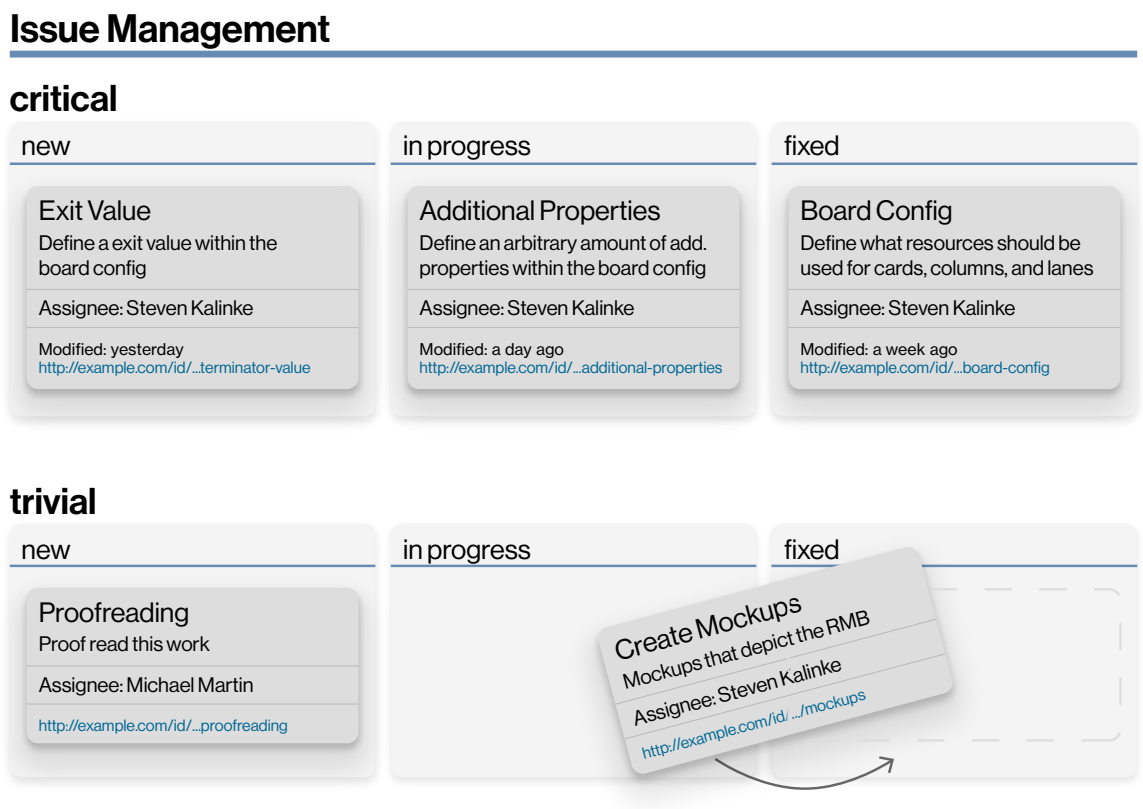

Figure 3.4: RMB Mockup of Use Case 4 depicting a mandatory title and resource identifier, and optionally a description, additional properties (i.e., assignee), and modification timestamp
.

3.2 Functional Requirements

This section derives functional requirements (FR_n) from the use cases presented at the beginning of this chapter. Each requirement will be described in terms of its specific requirement level (as described in the beginning of this chapter) and main category. In addition, each requirement will contain a brief description and a dedicated user story. Note that requirements are not derived for an individual use case but for the sum of all use cases. Table 3.2 provides an overview of the functional requirements, described in detail in the following.

№	Functional Requirement	Req. Level	Category
FR_1	Board Configuration	MUST	Feature
FR_2	Defaults Values	MUST	Feature
FR_3	Board Selection	MUST	Feature
FR_4	Drag & Drop Cards	MUST	Feature
FR_5	Swimlanes	MUST	Feature
FR_6	Disallow Card Drop On Adjacent Lanes	MUST NOT	Feature
FR_7	Disallow Column/Lane Repositioning	SHOULD NOT	Feature
FR_8	No Property Column	MUST	Feature
FR_9	Delete Property	SHOULD	Feature
FR_{10}	Everything Else Swimlane	MUST	Feature
FR_{11}	Create New Columns	MUST	Feature
FR_{12}	SPARQL Viewer	SHOULD	Feature
FR_{13}	SPARQL Editor	MAY	Feature
FR_{14}	Show Resources as Cards	MUST	Feature
FR_{15}	Board & Component Titles	MUST	UX
FR_{16}	Card Resource Identifier	MUST	UX
FR_{17}	Card Description	MUST	UX
FR_{18}	Card Additional Properties	MUST	UX
FR_{19}	Card Modified Timestamp	MUST	UX
FR_{20}	Card Click Dialog	MUST	UX
FR_{21}	Refresh Board	SHOULD	UX
FR_{22}	Loading Spinner	MUST	UX
FR_{23}	Resource Count Infobox	MUST	UX
FR_{24}	Show Warnings	MUST	UX
FR_{25}	Highlight No Property Column	SHOULD	UX
FR_{26}	Real-Time Timestamp Update	MUST	UX
FR_{27}	Real-Time Colored Cards	SHOULD	UX
FR_{28}	Trim Long Text	MUST	UX
FR_{29}	Relative Time	SHOULD	UX
FR_{30}	Tooltips	MUST	UX

Table 3.2: Overview of Functional Requirements (FR).

FR_1 — Board Configuration

Requirement Level: MUST Category: Feature

Initially, a user—or a resource manager—can define the components being displayed on a board (i.e., board and card component resources). This board configuration (or board config) is controlled from the 'outside' of the actual board implementation. In the context of this work, eccenca's *DataManager* is used to create or edit a board configuration (as introduced in chapter 1).

Note that the list of properties below is part of a prototypical application state. This means, that these properties represent the set of features which should be initially supported by the board configuration of the RMB.

User Story

In order to *create a new board OR edit an existing one,*
as a *resource manager,*
I want to *define/select the resources for every component of the board.*

Property — Description (* denotes required fields)

- Name* — the name of the board
- Description — a descriptive text on the intention of the board
- Board Limit — integer value used to set the limit of shown cards on a board
- Graph* — the knowledge graph which is used to get the cards from

Board Component Resources

- Cards* — which card class(es) (i.e., resources) should be shown in the board
- Columns* — the mutation property and the initial card-to-column allocation
- Lanes — the property for the initial card-to-lane allocation

Card Component Resources

- Description — the property which is used to fill the card body
- Additional P. — relation pointing to properties, used to show additional fields on the cards
- Modified — the property which is used to show a timestamp

FR_2 — Default Values

Requirement Level: MUST Category: Feature

In some situations, default values can serve the user's interest. This is, for example, the case when a user forgets to define a description property, although the requested cards contain one. Moreover, a default value for the board limit will prevent loading all cards from a giant graph.

User Story

In order to *catch a faulty board configuration,*
as a *user,*
I want to *see a card's description in any way, and an implicit card limit to prevent a stalling browser.*

FR_3 — Board Selection

Requirement Level: MUST Category: Feature

The UI for the RMB should allow users to display all available boards by their name property defined in the board configuration (see FR_1). Selecting a board will trigger the render process and eventually let the board appear below the selection element.

User Story

In order to *select my desired board,*
as a *user,*
I want to *see a list of available boards within the board's UI.*

FR_4 — Drag & Drop Cards

Requirement Level: MUST Category: Feature

A Kanban board—at the most basic level—allows a user to drag cards from one column to another. This action indicates a visual progress or update, and, regarding one goal of this work, relocating cards will modify the underlying knowledge graph by a specific property.

User Story

In order to *indicate card's progress AND to update a card's column property,*
as a *user,*
I want to *drag cards into other columns.*

FR_5 — Swimlanes

Requirement Level: MUST Category: Feature

As illustrated by the previous use cases, a user can define the desired swimlane resource within the board configuration (see FR_1). This makes it necessary to display swimlanes within the board of the current work. Note that, swimlanes are not used in all cases, as some Kanban solutions do not support them (e.g., *Trello*[19]).

User Story

In order to *further group the data being displayed,*
as a *developer,*
I want to *support swimlanes within the prototype.*

FR_6 — Disallow Card Drop On Adjacent Lanes

Requirement Level: MUST NOT Category: Feature

One could argue about whether or not it should be possible to move cards from their 'parental' swimlane to an adjacent lane. In this work, however, lanes are grouping cards that share the same context (as illustrated in Figure 1.2). It would therefore not be desirable to move cards to a misleading domain context. For example, the resource *Deforestation* in Figure 3.2 belongs to the domain of *Agriculture* (i.e., its lane). It should only be allowed to move this resource between columns within its parental container, as moving it to *Americas and the Caribbean* would lead to a misleading context.

User Story

In order to *protect a card's broader context,*
as a *user,*
I want to *move cards only within the parent lane container.*

[19] https://trello.com/

FR_7 — Disallow Column/Lane Repositioning

Requirement Level: SHOULD NOT Category: Feature

The subject of column and lane order is discussed in chapter 7 of this work as various aspects require further research, such as the lack of any sequencing information within RDF resources (and correspondingly the lack of information about the exact board position of a resource). Thus, as a rudimentary sorting strategy, the prototype will sort column and lane titles alphabetically.

User Story

In order to *prevent a column and lane re-ordering,*
as a *developer,*
I want to *prohibit the drag capabilities for columns and lanes.*

FR_8 — No Property Column

Requirement Level: MUST Category: Feature

As shown in the second use case (UNESCO Term Status), a column property is defined. However, it is not defined for a single resource. Nevertheless, the board should render the data in any way to further work with the data. This means, that a fallback column labeled with *no property* should be provided, if resources lack the requested column property. This condition is, for example, illustrated in the RMB mockup of the second use case (see Figure 3.2.

User Story

In order to *assign resources, that lack the requested column property, to a valid value,*
as a *user,*
I want to *move them away from the no property column.*

FR_9 — Delete Property

Requirement Level: SHOULD Category: Feature

If a user drops a card into the *no property* column the corresponding column property should be removed.

User Story

In order to *remove the column property from a resource,*
as a *user,*
I want to *drop the corresponding card to the no property column.*

FR_{10} — Everything Else Swimlane

Requirement Level: MUST Category: Feature

Similar to the previous condition, resources may lack the requested swimlane property. Therefore, a fallback swimlane should be provided at the very bottom of the board labeled with *Everything Else.*

User Story

In order to *display resources that lack the requested lane property,*
as a *user,*
I want to *manage them away in a dedicated lane.*

FR_{11} — Create New Columns

Requirement Level: MUST Category: Feature

In some scenarios, a desired column value is not present on the board. This is the case when (a) the value is not defined in all of the resources being displayed, or (b) the implicit board limit prevents the resource to display the desired column value. Moreover, as outlined in the second use case, a user may intend to assign the desired property from scratch by creating an arbitrary amount of new column values.

User Story

In order to *assign cards to not existing values,*
as a *user,*
I want to *create new columns.*

FR_{12} — SPARQL Viewer

Requirement Level: SHOULD Category: Feature

For some advanced users, it can be helpful to review the SPARQL request, which is responsible for displaying the board. Therefore, the corresponding query should displayable to the user in an unobtrusive manner.

User Story

In order to *review what resources got rendered to the board,*
as a *resource manager,*
I want to *inspect the responsible SPARQL request.*

FR_{13} — SPARQL Editor

Requirement Level: MAY Category: Feature

Moreover, it can be helpful to modify the SPARQL query to test/debug different aspects without touching the actual board configuration (FR_1). For example, users can manipulate the board's limit by directly editing the SPARQL's `LIMIT` within the *SPARQL Viewer*.

User Story

In order to *provide a SPARQL interface,*
as a *resource manager,*
I want to *modify the query within the SPARQL Viewer.*

FR_{14} — Show Resources as Cards

Requirement Level: MUST Category: Feature

RDF resources should be represented by cards of the board. In this context, resources refer to the selected card class(es) in the board configuration (FR_1).

User Story

In order to *overview all the selected resources for my card class(es),*
as a *user,*
I want to *see cards on the board that represent RDF resources.*

FR_{15} — Board & Component Titles

Requirement Level: MUST Category: UX

Each board should depict their title at the top of the board, whereas the board's description should be placed below. To identify the board's structure and semantics, column and lane titles should be placed accordingly. Likewise, cards should depict their title in a salient manner (similar as depicted by the mockups). If the title of an element is not of type literal, it should match the corresponding value of the RDF label property.

User Story

In order to *identify the board's structure and semantics,*
as a *user,*
I want to *see all the corresponding titles.*

FR_{16} — Card Resource Identifier

Requirement Level: MUST Category: UX

Since all resources have an identifier (i.e., their IRI), it should be depicted on a card. The IRI should render as a clickable link in case it is an actual URL.

User Story

In order to *know AND possible access the resource identifier,*
as a *user,*
I want to *see the resource's URI in the card.*

FR_{17} — Card Description

Requirement Level: MUST Category: UX

Resources can consist of a description property that describes their purpose (e.g., the terms in the FOAF vocabulary from the first use case). If a description property is defined on a resource, it should be shown below the card's title (e.g., Figure 3.1).

User Story

In order to *better understand a card's purpose,*
as a *user,*
I want to *see the card's description.*

FR_{18} — Card Additional Properties

Requirement Level: MUST Category: UX

Different use cases have different requirements regarding the properties displayed on the card. As illustrated by the mockups for use case 3 and 4 (see Figure 3.3 and Figure 3.4, resp.), additional properties can be used to flexibly display an arbitrary amount to resources.

User Story

In order to *display an arbitrary amount of properties and their values on a card,*
as a *user,*
I want to *define additional properties.*

FR_{19} — Card Modified Timestamp

Requirement Level: MUST Category: UX

A modified timestamp can be used to indicate whether a card has been moved or not, since it displays the exact time point when a card has been dropped the last time.

User Story

In order to *know if a card was moved and when,*
as a *user,*
I want to *see the last modification timestamp for these cards.*

FR_{20} — Card Click Dialog

Requirement Level: MUST Category: UX

User Story

In order to *provide detailed information for a particular resource,*
as a *user,*
I want to *click on cards to see a dialog.*

FR_{21} — Refresh Board

Requirement Level: SHOULD Category: UX

When users change certain properties within the card click dialog (e.g., the title of the card), the board should reflect these changes immediately.

User Story

In order to *view my recent updates,*
as a *user,*
I want to *refresh the board within the card dialog.*

FR_{22} — Loading Spinner

Requirement Level: MUST Category: UX

When interacting with the prototype, there are many situations in which data gets updated or requested. Without any visualization indicating a running process, a user may perceive the application as being in an idle in such situations. Thus, to reflect the application's loading state, a spinner element should be displayed during that period.

User Story

In order to *recognize a loading state,*
as a *user,*
I want to *see a loading indicator.*

FR_{23} — Resource Count Infobox

Requirement Level: MUST Category: UX

Unless there is an explicit large limit defined within the board configuration, it is likely the case the board shows only a subset from the entirety of all defined card classes. In this case, an infobox should inform users about the fact that not all resources are currently displayed. Otherwise, that is if the number of cards is less than the board limit, the infobox should inform about the current card count.

User Story

In order to *know how many cards are displayed on the board and whether they are limited,*
as a *user,*
I want to *be notified about the amount of cards, and—if a subset is presented—the fact that there are more cards available.*

FR_{24} — Show Warnings

Requirement Level: MUST Category: UX

Users should receive warnings. For example, if a required component (e.g., the graph) is missing. If this is the case, an infobox should be provided to the user explaining the circumstances and providing guidance if possible.

User Story

In order to *get informed about errors,*
as a *user,*
I want to *see an infobox providing details on the subject.*

FR_{25} — Highlight No Property Column

Requirement Level: SHOULD Category: UX

In the case that a resource does not contain the requested column property, it is grouped into the *no property* column (see FR_8). To highlight the unique role of this additional column, it should be more salient compared to the other regular columns.

User Story

In order to *tell regular columns apart from the no property column,*
as a *user,*
I want to *perceive a visual cue for these columns.*

FR_{26} — Real-Time Timestamp Update

Requirement Level: MUST Category: UX

After a user drops a card to another column, the card should present a real-time indication of the modified timestamp. For example, the card should show: *Modified: just now.*

User Story

In order to *perceive a card's timestamp update status,*
as a *user,*
I want to *get a visual feedback from the modified field after dropping the card.*

FR_{27} — Real-Time Colored Cards

Requirement Level: SHOULD Category: UX

When users work with many cards in a board, it is helpful to color-code cards that are moved throughout an active session (i.e., the time when a user works with the board without refreshing or closing the page). The color-coding should vanish after redrawing the board or refreshing the page.

User Story

In order to *know what cards I just moved,*
as a *user,*
I want to *see the cards appear in a different color.*

FR_{28} — Trim Long Text

Requirement Level: MUST Category: UX

Text elements can reach a length where they break the UI. For example, depicting a very long URI would create unnecessary visual noise on a card. Therefore, all text elements should have reasonable boundaries and should be trimmed off if they exceed that limit.

User Story

In order to *avoid visual noise by long text elements,*
as a *user,*
I want to *see text being truncated.*

FR_{29} — Relative Time

Requirement Level: SHOULD Category: UX

Humans are faster in perceiving a relative time designation (e.g., 2h ago) compared to an absolute one (e.g., at 14:03), especially when dealing with different time zones. Therefore, cards should depict relative times for the modification timestamp, as illustrated by the mockups in use case 1, 3, and 4.

User Story

In order to *quickly perceive modification dates,*
as a *user,*
I want to *see a relative time depicted for the modification timestamps.*

FR_{30} — Tooltips

Requirement Level: MUST Category: UX

When hovering over truncated text elements, a tooltip should reveal the full information for that particular object. Similar, when hovering over a relative time, a detailed absolute timestamp should appear.

User Story

In order to *get the full information about an element,*
as a *user,*
I want to *see a tooltip when hover over truncated elements.*

3.3 Non-Functional Requirements

This section derives non-functional requirements (NFR_n) from the use cases presented at the beginning of this chapter. Table 3.3 provides an overview of the non-functional requirements defined throughout this section.

№	Non-Functional Requirement	Req. Level	Category
NFR_1	Dynamic Lane Height	MUST	UI
NFR_2	eccenca UI Styling	MUST	UI
NFR_3	Test Data	SHOULD	Testing
NFR_4	Directory & Component Structure	SHOULD	Conventional
NFR_5	Linter Conformity	SHOULD	Conventional
NFR_6	eccenca Infrastructure Integration	MUST	Backend

Table 3.3: Overview of Non-Functional Requirements (NFR).

NFR_1 — Dynamic Lane Height

Requirement Level: MUST Category: UI

In the second use case (see mockup on page 12), it is likely the case that a single column, within a lane, holds a vast amount of cards. Thus, reaching the lanes below becomes a scrolling intense endeavor. To avoid this issue, lanes should have a maximum height of the browser's current viewport. If a column reaches this limit, the column itself should become a scrollable container.

User Story

In order to *avoid an 'endless' board,*
as a *developer,*
I want to *limit the lane's height AND make columns scrollable if they exceed that limit.*

NFR_2 — eccenca UI Styling

Requirement Level: MUST Category: UI

The prototype should visually match with the other eccenca components. Therefore, UI elements should rely on existing style guides.

User Story

In order to *visually match existing eccenca components,*
as a *developer,*
I want to *utilize existing UI style guides.*

NFR_3 — Test Data

Requirement Level: SHOULD Category: Testing

The RMB needs to handle a variety of special conditions or edge cases. Therefore, test data should be used to guarantee a stable processing of the prototype.

User Story

In order to *test the prototype under various conditions,*
as a *developer,*
I want to *provide test data for different scenarios.*

NFR_4 — Directory & Component Structure

Requirement Level: SHOULD Category: Conventional

eccenca has elaborated a set of guidelines for front-end developers. Most importantly, developers should stick to a recommended directory and component structure for a React project. These conventions provide help for other developers, and simplify the integration of novel components into existing structures.

User Story

In order to *conform to existing structuring guidelines,*
as a *developer,*
I want to *follow the company's front-end conventions.*

NFR_5 — Linter Conformity

Requirement Level: SHOULD Category: Conventional

User Story

In order to *conform to coding standards,*
as a *developer,*
I want to *keep the code conform to given linter rules.*

NFR_6 — eccenca Infrastructure Integration

Requirement Level: MUST Category: Backend

The prototype utilizes various components of eccenca's ecosystem. For example, the triple store to request and store data, various API components (e.g., to resolve a URI to a label), or Keycloak[20] for authentication and identity management. To align the integration of the prototype, these services need to be used.

User Story

In order to *connect to eccenca's backend services,*
as a *developer,*
I want to *use existing authentication methods AND components to store and retrieve data.*

3.4 Overview & Prioritization

Throughout this chapter, a total of 30 functional and 6 non-functional requirements have been defined, which also set the general scope of this work. Figure 3.5 provides an overview of the requirements listed throughout this chapter by their category. As can be observed, the categories *Feature* and

[20] https://www.keycloak.org/

UX mark the significant part of all requirements, since requirements categorized as *Feature* are scaffolding the prototype, whereas the category *UX* describes a desired behavior that aims to enhance the prototype's utility and usability regarding user interactions.

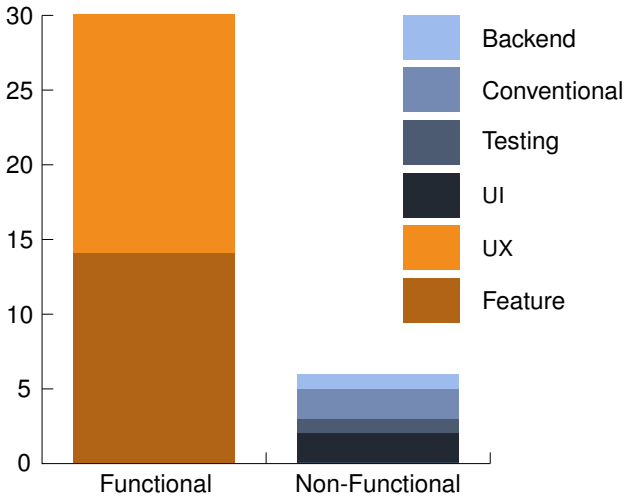

Figure 3.5: Overview for functional ($n = 30$) and non-functional ($n = 6$) requirements by their category.

Nevertheless, the other non-functional categories (i.e., *UI, Testing, Conventional, and Backend*) describe elements that are significant for the development process; most prominently, the integration into an existing infrastructure (NFR_6).

Overall, the listed requirements set a framework for a prototypical application state. The priority of functional requirements, however, have more significance compared to non-functional requirements. The requirement levels for most items are either *MUST* or *SHOULD*, as can be seen in Table 3.4.

Category	Requirement Levels				
	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT
Feature	9	2	1	1	1
UX	12	4	-	-	-
UI	2	-	-	-	-
Testing	-	1	-	-	-
Conventional	-	2	-	-	-
Backend	1	-	-	-	-
Functional	21	6	1	1	1
Non-Functional	3	3	-	-	-
Sum	24	9	1	1	1

Table 3.4: Requirements overview by their requirement level (see Bradner, 1997).

There is only one requirement that must not be specified, that is the ability to drop cards on adjacent lanes (see FR_5). Moreover, the subject of column and lane order and repositioning (as outlined in FR_6) has a level of *SHOULD NOT*, since it requires further research (as described in chapter 7).

4 State of the Art

As described in chapter 1, the main contribution of this work is to provide a novel approach allowing to modify specific RDF values by moving cards over a Kanban board. Since there is no comparable previous work, this chapter provides an overview of the visualization of RDF graphs (section 4.1), as well as existing Kanban board solutions (section 4.2).

4.1 Graph Visualization

As mentioned in the introduction, graphs are usually visualized by a set of ellipses that are interconnected by arrows, representing the graphs' nodes and edges, respectively. Nodes, however, may also be shaped as rectangles depending on their context. A notable application for editing and visualizing RDF resources is *Protégé*.[21] Since its initial release in 1999, it has become the leading ontological engineering tool. It is an open-source project, and its plugin system allows other developers to contribute to the project (Gaševic et al., 2009, p. 62).

Another more recent work in the field of visualizing linked data (especially ontologies) has been conducted by Lohmann and colleagues (2016). The authors introduced a visual language for OWL ontologies (*VOWL*) aiming to provide a comprehensive visualization that is comprehensible by *"[…] casual ontology users with only little training"* (Lohmann et al., 2016, p. 1). VOWL is implemented as a Protégé plugin as well as a web application.[22] The web application is publicly available as an open-source project[23] and can be embedded by other applications. For example, VOWL is integrated within eccenca's front-end (i.e., the *DataManager*).

Furthermore, Lohmann et al. elaborated on existing graph visualization in their work, most of which are available as Protégé plugins; for example, *TGViz*,[24] *NavigOWL*,[25] and *SOVA*.[26] A comprehensive review of knowledge graph visualizations is provided in section two in the article (Lohmann et al., 2016, p. 2).

It should be again pointed out, that this work does not attempt to visualize an entire graph. This approach would primarily lead to a clash of paradigms. It would map a graph to a relational model since a Kanban board can be considered as a table structure. However, when a user defines the structural components of the board (i.e., the resources which should be used for cards, columns, and lanes within the board configuration, see FR_1), they explicitly dissect the graph, and therefore, bypassing the clash. In other words, preselecting the board components creates a table-compatible subset graph.

[21] Initial Release: 1999. GitHub: https://github.com/protegeproject/protege. Web-version: https://webprotege.stanford.edu/.
[22] *VOWL* home page: http://vowl.visualdataweb.org/.
[23] More information: http://vowl.visualdataweb.org/webvowl.html
[24] Protégé Wiki: https://protegewiki.stanford.edu/wiki/TGViz.
[25] Protégé Wiki: https://protegewiki.stanford.edu/wiki/NavigOWL.
[26] Protégé Wiki: https://protegewiki.stanford.edu/wiki/SOVA.

To illustrate the mapping and visualization process of the RMB. Figure 4.1 (A1) compares a 'traditional' graph visualization (using VOWL)[27] with a mockup of the Resource Management Board (B). Both images (A1) and (B), depict a small section of the FOAF graph, whereas (A2) reveals details about a selected node from (A1); in this instance, the FOAF term *knows* got selected. On the other hand, (B) depicts a small section from the first use case of this work (i.e., FOAF Term Status). As stated before, the RMB only targets selected card classes and maps their inherent properties to the columns and lanes of the board. Thus, in contrast to other graph visualizations, the RMB provides a selective perspective on a graph.

To comprehend the mapping and visualization process of the RMB, compare the contents of image (A2) and (B) of Figure 4.1. In the first use case, the prototype's aim was to target three specific properties of the FOAF graph in order to generate a board. These three properties (i.e., cards, columns, and lanes) were defined as *board component resources*, as they structurally define the board components. The card classes were defined as one of `owl:Class`, `owl:ObjectProperty`, or `owl:DatatypeProperty`. Therefore, the term *knows* appeared as a card on the board, as in image (B) and listed in (A2). The type property was also used to separate the cards by lane. Finally, the cards got distributed over columns by their inherent `vs:term_status` value, as depicted in (B) and listed in (A2).

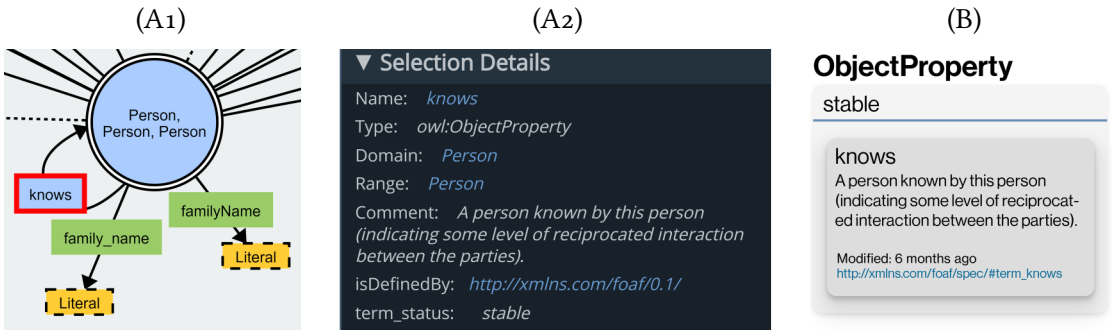

Figure 4.1: Visualization approaches of VOWL (A) and the RMB (B). VOWL provides a visualization for the entire graph, whereas the RMB targets a specific selection.

4.2 Kanban Board Solutions

The initial idea of this project was to develop an in-house Kanban board from scratch. However, considering that the Kanban board acts only as a tool for reaching the actual project goal, we decided to use the existing Kanban board solution react-trello as underlying basis. This decision was mainly based on the assumption that developing a board from scratch would go beyond the scope of a Master's thesis. Compared to an in-house application, a third-party solution would allow to develop a vertical prototype at a faster pace while focusing on the actual goal of the project. Furthermore, an in-house board solution would bear the risk of stagnation once the basic requirements were satisfied.

To explain why we chose react trello as basis, the current section reviews existing implementations of Kanban board solutions. Our review was based on the following project-relevant criteria: First and foremost, a permissive software license (such as MIT or ISC) is essential for a component that is supposed to be integrated into a commercial software product. Moreover—having the technology

[27] The VOWL FOAF visualization can be accessed under `http://www.visualdataweb.de/webvowl/#foaf`.

stack in mind—the eccenca's user interface is created with React, meaning that the Kanban board needed to be React-based. React can be used with both languages JavaScript and TypeScript. Although both languages allow to develop an independent software component, JavaScript represents the preferred option, since eccenca's build workflows and front-end codebase is entirely written in and adjusted towards JavaScript. Another crucial criterion was the injectability of data. As mentioned previously, our project did not aim to create data from scratch. Instead, existing data needs to be integrated into the board.

We only considered active projects, that is projects with the last commit date no longer than one year ago. As an indicator of popularity and quality, we used the star and fork count. Table 4.1 provides an overview of publicly available Kanban board solutions. Projects set in bold type are fulfilling most of the just criteria.

Project, *License*, Author GitHub Page	JS/TS React Version	Injectable? Source	Lanes?	Stars/Forks
react-kanban, *MIT*, Lourenci, Leandro `https://github.com/lourenci/react-kanban`	**JavaScript** **>16.8.5**	**Yes** **JSON-like**	**No**	**61/19**
React Kanban DND, *MIT*, Besen, Lucas `https://github.com/lucasbesen/react-kanban-dnd`	TypeScript >16.5.2	Yes JSON-like	No	95/10
react-trello, *MIT*, Ramachandran, R. `https://github.com/rcdexta/react-trello`	**JavaScript** **>15.4.2**	**Yes** **JSON-like**	**No**	**781/192**
Kanban Board App, *ISC*, Shellyl, N. `https://github.com/shellyln/kanban-board-app`	TypeScript >16.9.0	No CouchDB	Yes	16/5
React Kanban, *MIT*, Englund, Markus `https://github.com/markusenglund/react-kanban`	JavaScript >16.2.0	No MongoDB	No	1,400/140

Table 4.1: Comparison of React Kanban board solutions based on their development language, data injectability, swimlane support, star, and fork count.

As illustrated in Table 4.1, react-kanban (first commit on March 19, 2019) and react-trello (first commit on January 24, 2017) turned out to be the most promising projects to provide a foundation for this work. Both React projects are under active development, and share the advantages of being JavaScript-based, and offering an interface to inject data. On the other hand, both projects, by design, do not support swimlanes. A central advantage of react-trello is, that it has higher popularity and active contributor count compared to react-kanban. Moreover, at the time of developing, eccenca's React codebase is at version 15.x, making it an excellent bedrock, since its minimal supported version is React 15.4.2. Furthermore, react-trello seems to be a sophisticated project with active contributions from over 20 users, and active maintenance regarding its issue management on GitHub.

5 Specifications

This chapter is structured in the following way: First, section 5.1 specifies the graph model for the board configuration and provides screenshots of its representation in eccenca's *DataManager*. Hereafter, section 5.2 introduces the data model for the external component react-trello, followed by the adjustments and specifications for the prototype's model and interface; and lastly, section 5.3 provides an overview of the query strategy for sending and fetching data between the RMB and eccenca's SPARQL endpoint.

5.1 Board Configuration

This section introduces the model for the board configuration, which allows users to control the resources required to display a board within the prototype (e.g., board and card component resources). The board configuration is part of the *DataManager*, which is eccenca's front-end allowing users to author and explore semantic content.

5.1.1 Config Definition

The graph-based board configuration is defined by different terms, primarily RDF(S), OWL, and SHACL. However, this subsection only highlights key aspects in prefixed Turtle notation; the entire graph can be found in the appendix of this work. Fundamentally, the RMB is defined as an `owl:Ontology` and the board configuration as an `owl:Class`, as defined in Code 5.1.

```
rmb:
  a owl:Ontology ;
  rdfs:label "Resource Management Board" .
rmb:BoardConfig
  a owl:Class ;
  rdfs:label "Board Configuration" .
```

Code 5.1: Exemplary sample of the RMB and the board configuration as an OWL ontology and class.

As requested by the first functional requirement (FR_1), the board configuration contains various properties to describe a board. Code 5.2 illustrates the definition of the board component resources.

```
rmb:cardsClass
  a owl:ObjectProperty ;
  rdfs:label "Cards Class(es)" ;
  rdfs:domain rmb:BoardConfig .

rmb:cardsColumnProperty
  a owl:ObjectProperty ;
  rdfs:label "Column Property" ;
  rdfs:domain rmb:BoardConfig .

rmb:cardsLaneProperty
  a owl:ObjectProperty ;
  rdfs:label "Lane Property" ;
  rdfs:domain rmb:BoardConfig .
```

Code 5.2: Exemplary sample of the definition for the board component resources.

SHACL is used to validate the graph by certain conditions. Moreover, eccenca's *DataManager* relies on these shapes to render their front-end UI. Code 5.3 illustrates the use of SHACL to specify the `cardsClass` property further. For example, the property `sh:nodeKind` (line 4) has a value of `sh:IRI`. This constraint means that nodes confirming to `rmb:cardsClass` must be of type IRI. Moreover, it can be observed that `sh:minCount` (line 6) is defined, while `sh:maxCount` is not. This is on purpose since cards are allowed to refer to multiple classes (or resources). In contrast, the shape specifications for column and lane properties would contain both conditions, since a `sh:minCount` of `1` defines a property as mandatory, whereas `sh:maxCount` defines a maximum number of allowed instances.

```
1 rmb:cardsClassSHACL
2   a sh:PropertyShape ;
3   sh:class owl:Class ;
4   sh:nodeKind sh:IRI ;
5   sh:path rmb:cardsClass ;
6   sh:minCount 1 .
```

Code 5.3: Sample SHACL specification for the property `cardsClass`.

5.1.2 Config Properties and Relations

The board configuration consists of ten properties, as requested by the first functional requirement (FR_1 on page 17). Table 5.1 provides a more specific overview of all properties as defined by the graph for board configuration. Note that the last column, RMB Defaults, is referring to default values being used by the RMB, as requested by FR_2.

Board Config Object ⟶ `sh:nodeKind`	Object Description	*required*/RMB Default
General Board Properties		
`rdfs:label` ⟶ `sh:Literal`	The name of the board	*required*
`dct:description` ⟶ `sh:Literal`	Descriptive text about the board's intention	`""`
`rmb:cardsGraph` ⟶ `sh:IRI`	Graph, which is used to get the cards from	*required*
`rmb:boardLimit` ⟶ `sh:Literal`	Integer to set the card display limit on a board	100
Board Component Resources		
`rmb:cardsClass` ⟶ `sh:IRI`	Card resources (multiple)	*required*
`rmb:cardsColumnProperty` ⟶ `sh:IRI`	Column resource (mutation property)	*required*
`rmb:cardsLaneProperty` ⟶ `sh:IRI`	Swimlane resource	`""`
Card Component Resources		
`rmb:cardsDescriptionProperty` ⟶ `sh:IRI`	The property to fill the card body	`dct:description`
`rmb:cardsAdditionalFieldProperty` ⟶ `sh:IRI`	Resources referring to additional properties	`""`
`rmb:cardsModifiedProperty` ⟶ `sh:IRI`	Property used to save modified timestamps	`dct:modified`

Table 5.1: Board Configuration Properties.

To illustrate the structure, relations, and cardinalities of the board configuration, the class diagram in Figure 5.1 highlights the class for the board configuration along with its properties. To transfer existing conventions from the traditional UML class diagram to the domain of RDF, the work by Tong and colleagues (2015) is used as a reference. Tong et al. propose a model that allows a *"Construction of RDF(S) from UML Class Diagrams."* Although they neither included OWL nor SHACL in their model, it is a helpful guide that is applicable in both directions. For example, they provided two tables that map the main elements and datatypes of UML to RDF(S) and vice versa (Tong et al., 2015, pp. 241, 243).

Nevertheless, in the following, the most important key aspects for mapping the board configuration to a UML class diagram are outlined. Furthermore, based on their publication, the corresponding transformation rule and page will be used: First and foremost, RDF classes are mapped to regular UML classes (Rule 2, p. 241). This rule applies to the board configuration since it is defined as an `owl:Class` (see Code 5.1). RDF properties whose values are literals (i.e., specified by an xsd datatype) become UML class attributes (Rule 4, p. 241), as exemplarily depicted by the `rdfs:label` within classes. Lastly, properties whose values are resources, and their `rdfs:domain` refers to their parent class become an UML aggregation (Table 1, p. 241 and Rule 6, p. 242).

Since Tong et al. did not incorporate OWL and SHACL, they concluded that cardinality could not be expressed in their model (p. 244). However, both OWL and SHACL allow to define cardinalities. Specifically, OWL has a dedicated property (i.e., `owl:minCardinality` and `owl:maxCardinality`), whereas SHACL uses the previous mentioned `sh:minCount` and `sh:maxCount` properties which may also be used to describe cardinality. Figure 5.1 depicts the corresponding cardinalities between each relation. For this prototypical configuration, `rmb:cardsClass` and `rmb:cardsAdditionalFieldProperty` are the only elements that allow multiple instances (as outlined in FR_1 and FR_{17} resp.).

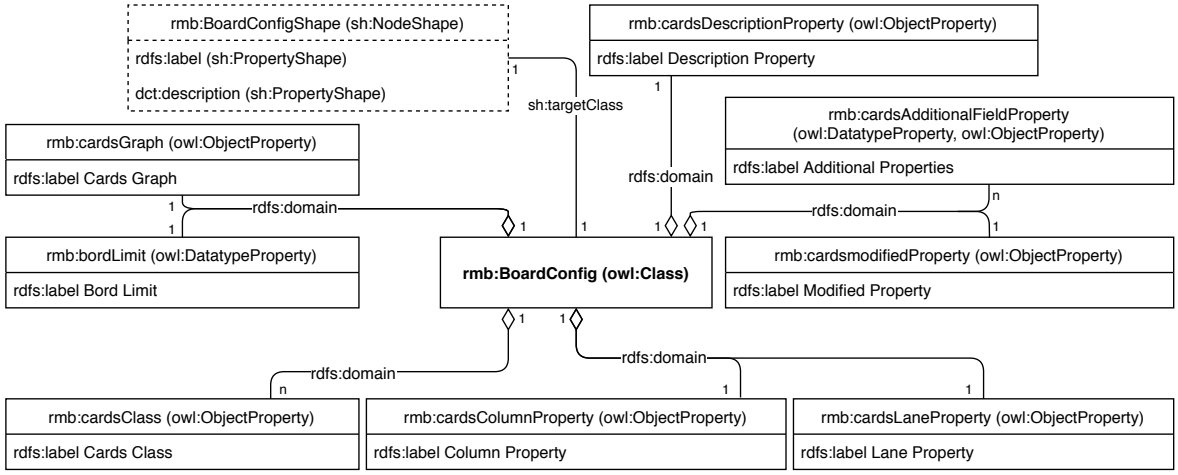

Figure 5.1: The prototypical board configuration graph represented as a UML class diagram.

5.1.3 Config Instance & Usage

For demonstration purposes, Code 5.4 defines a particular instance of a board configuration with the data requested by the first use case (for reference see the board and card component resources on page 9). Note that neither developers nor users need to define a new board configuration in this manner; instead, users define a board using eccenca's *DataManger* which will generate a similar specification:

```
rmb:foaf-term-status
  a rmb:BoardConfig ;
  rdfs:label "FOAF Term Status" ;
  dct:description "Manages terms by their status in the FOAF namespace" ;
  rmb:cardsGraph foaf: ;
  rmb:cardsClass owl:Class, owl:DatatypeProperty, owl:ObjectProperty ;
  rmb:cardsColumnProperty vs:term_status ;
  rmb:cardsLaneProperty rdf:type ;
  rmb:cardsDescriptionProperty rdfs:comment .
```

Code 5.4: A board configuration instance of the first use case (FOAF Term Status).

The *DataManager* relies on a graph that contains shape definitions to render its UI. In other words, a (compatible) graph gets translated to a user interface. When importing the entire board configuration graph (see appendix) into eccenca's *DataManager*, the front-end will render the graph as described in Figure 5.2. While Figure 5.2 (A) illustrates the interface when defining a new board configuration, Figure 5.2 (B) shows the configuration for the first use case. After creating a new board configuration, as in (B), this particular instance will be added to the board configuration graph. In other words, exporting the board configuration graph of Figure 5.2 (B), would lead to a graph with similar data as Code 5.4.

The properties listed in Figure 5.2 (A) correspond to all properties requested in the first functional requirement (see FR_1) or as specified in Table 5.1. Moreover, as image (A) shows, there are four required fields: the board's name, graph, card class(es), and column property. Thus, the UI requires the user to add a resource. This UI functionality is derived from a resource `sh:minCount` property set to value 1, making it mandatory to define a resource, as mentioned earlier.

Figure 5.2 (B) depicts the setup for the *FOAF Term Status* use case (see page 9 for the board and card component resources). The class(es) field contains multiple resources since a `sh:maxCount` is not defined for this property.

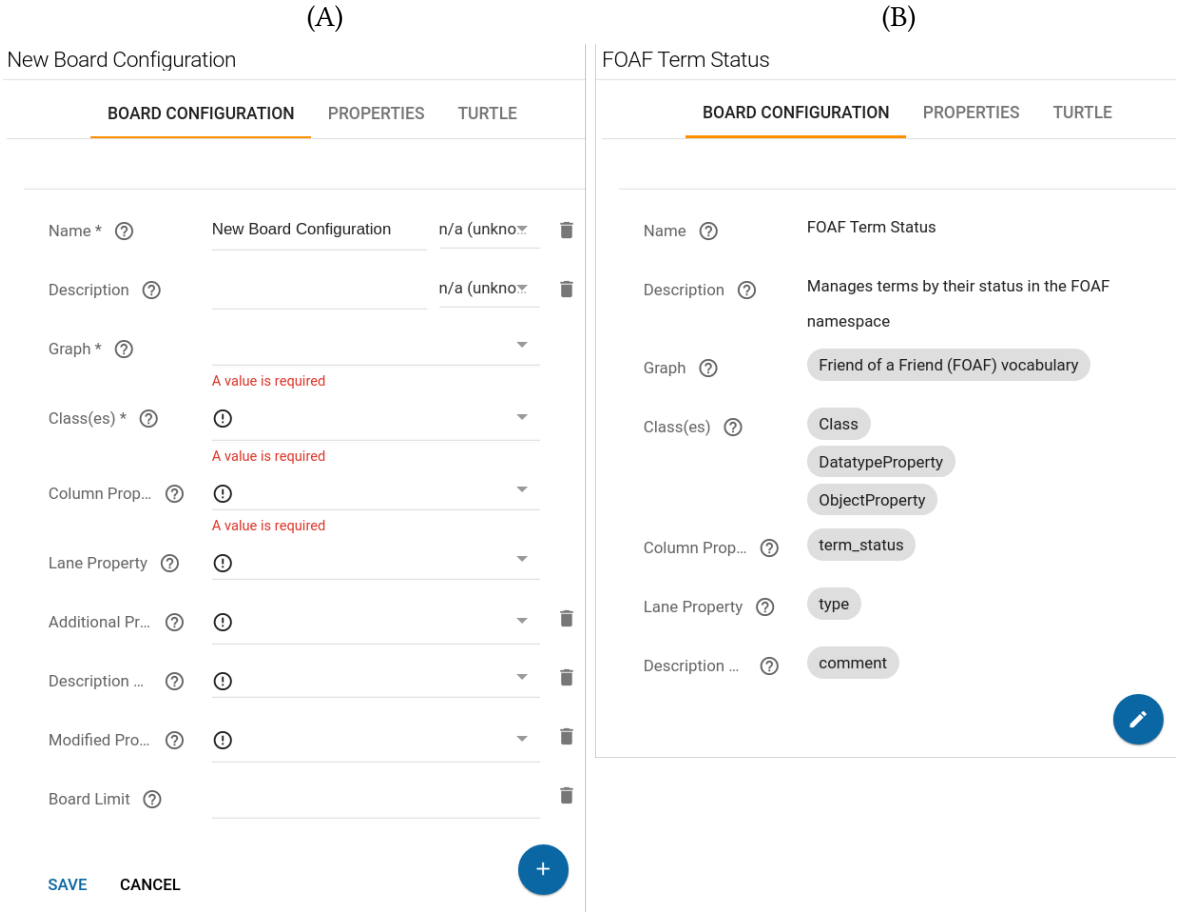

Figure 5.2: Board configuration graph represented in eccenca's *DataManager*. (A) is showing all defined fields of the board configuration. (B) is showing an actual instance of a board configuration (i.e., the first use case).

5.2 Board Specifications

The following subsection 5.2.1 describes the specifications, limitations, and workarounds for the third-party component react-trello (Ramachandran, 2017). Then subsection 5.2.2 provides the design specifications for the prototype.

5.2.1 react-trello

This subsection will highlight the most important aspects of react-trello to understand its general usage. Since react-trello is one of the few JavaScript-based React projects that allows to inject data (see Table 4.1), the following segment will introduce its data model. It is important to point out that—regardless of how the intermediate data model is shaped—the resulting model needs to satisfy the required target data model by react-trello in order to render a board.

Target Data Model

The data is stored in a JSON format, containing a column array, which itself contains a card array. Code 5.5 provides an overview of the target data model that react-trello expects in order to render a board flawlessly. The code depicted below describes a board that consists of two columns (i.e., testing and stable) and three cards (i.e., depiction, knows, and personal mailbox). Note that react-trello uses the misleading term *lanes* to describe the *columns* of the board (see line 2). Although I reported the idiosyncratic terminology.[28] It is unlikely that the term will be changed in the future since renaming elements within the data model would introduce breaking changes[29] for a variety of users. For the sake of clarity, I will continue using the term column when referring to react-trello's lanes.

```
1  {
2    "lanes": [{
3        "id": "testing",
4        "title": "testing",
5        "cards": [{
6            "id": "http://xmlns.com/foaf/spec/#term_depiction",
7            "title": "depiction",
8            "description": "A depiction of some thing)."
9          }]
10     }, {
11       "id": "stable",
12       "title": "stable",
13       "cards": [{
14           "id": "http://xmlns.com/foaf/spec/#term_knows",
15           "title": "knows",
16           "description": "A person known by this person (indicating some level of [...]).",
17           "modified": "2019-06-14T16:49:21+02:00"
18         }, {
19           "id": "http://xmlns.com/foaf/spec/#term_mbox",
20           "title": "personal mailbox",
21           "description": "A personal mailbox, ie. an Internet mailbox associated [...]"
22         }]
23     }]
24 }
```

Code 5.5: Target Data model of the react-trello component defining two columns and three cards.

react-trello's data model—similar to its namesake product *Trello*—does not support swimlanes. This is important to note since this feature is demanded by all use cases and as it is a functional requirement

[28] https://github.com/rcdexta/react-trello/issues/126.
[29] That means it will require users to make a corresponding change in their code as well.

(FR_4). Nonetheless, the data model fulfills two structural requirements; (1) columns and cards have an `"id"` key, which is convenient since a URI is an excellent id by definition, and (2) React, the JavaScript library for creating the front-end, works more efficiently with arrays rather than objects, which furthermore have drawbacks with regard to sorting their elements.

The specification to use the component within React is demonstrated below in Code 5.6. The object `data` (line 4) is the only mandatory attribute and refers to a JSON structure similar to previously introduced in Code 5.5.

```
1 import Board from 'react-trello';
2 /* React Component Structure ... */
3 return(
4   <Board data={data} />
5 )
```

Code 5.6: Minimal React code to render a board using react-trello.

Figure 5.3 shows the default visual representation of react-trello. The underlying data is taken from previous Code 5.5 that is similar to the data from the first use case, yet limited regarding its board features (see the mockup of use case 1 on page 10 for comparison).

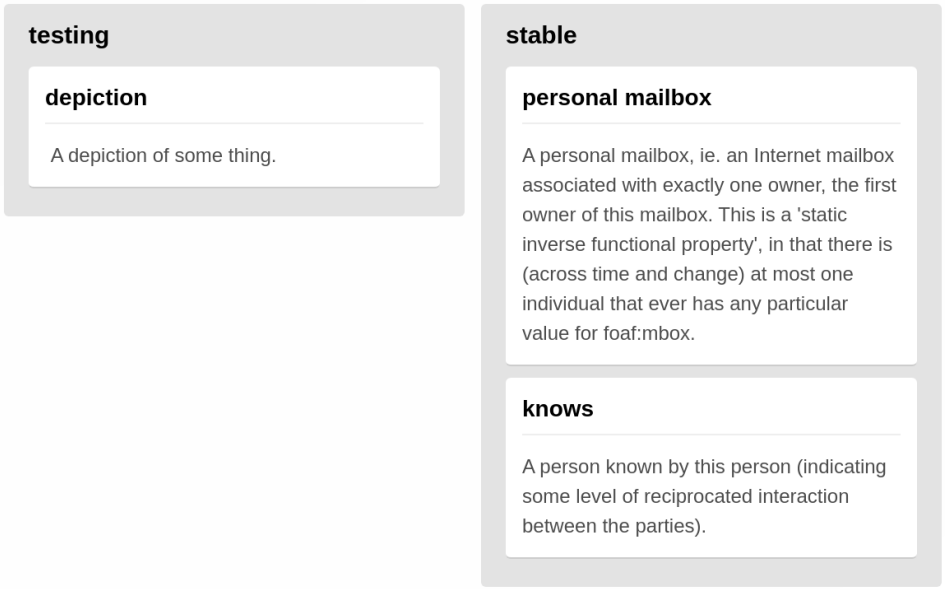

Figure 5.3: Default react-trello board visualizing the data from Code 5.5.

Bypassing Limitations

Without any modifications, the default card element of react-trello (as shown above) is limited to a predefined amount of elements it can possibly depict on its surface. However, react-trello allows developers to use their own card component. This provides two fundamental benefits: (1) Developers have no boundaries regarding card styling and positioning of elements, and (2) every value of a key that is specified within the card object of the data model can be depicted on a card. For example, the key `modified` at line 17 in previous Code 5.5 can be used as a visual element within a custom card component (see line 8 in Code 5.7).

The following two code samples demonstrate the specification for a custom card component (Code 5.7) and the necessary changes within the react-trello component (Code 5.8). Note that the

properties used in Code 5.7 (e.g., `modified` in lines 2 and 8) need to match exactly the key names defined within the data model in Code 5.5

```
1  /* Custom Card Function at CustomCard/CustomCard.js */
2  export default ({title, description, modified, id}) => {
3    return (
4      <div>
5        <h1>{title}</h1>
6        <p>{description}</p>
7        <hr />
8        <p><small>{modified}</small></p>
9        <p><a href={id}>{id}</a><p>
10     </div>
11   )
12 };
```

Code 5.7: Example of a custom card component.

In order to use the prior defined custom card components the main react-trello component requires the attribute `customCardLayout` (line 6) and the child element `<CustomCard \>` (line 7) to be set.

```
1  /* Main React Component */
2  import Board from 'react-trello';
3  import CustomCard from './CustomCard/CustomCard';
4  /* ... */
5  return(
6    <Board data={data} customCardLayout>
7      <CustomCard />
8    </Board>
9  );
```

Code 5.8: Usage of custom cards in react-trello.

At this stage, the former code samples describe the core building blocks in order to create and display a board with an arbitrary amount of columns and cards. Although react-trello's data model does not support swimlanes, this limitation can be bypassed by loading multiple instances of the board component, each containing a different data chunk, as depicted in lines 6 and 7 in Code 5.9. This means that, conceptually, each react-trello board can be considered as a swimlane. To obfuscate the semantic mapping from board to swimlane, the import statement might by renamed from `Board` to `Lane` (see lines 2, 6, and 7).

```
1  /* Main React Component */
2  import Lane from 'react-trello';
3  /* ... */
4  return(
5    <div id="board-container">
6      <Lane data={data1} />
7      <Lane data={data2} />
8    </div>
9  );
```

Code 5.9: Example of multiple swimlanes.

Additionally, the existing data model needs to be adjusted to include swimlanes. Code 5.10 specifies a JSON template that acts as a superset of react-trellos's data model. The benefit of this design is that it is (a) able to manage swimlanes, and (b) when destructuring the object by swimlanes, the resulting object satisfies the requested target data model by react-trello (lines 5 to 24 and 28 to 52). The adjusted model below consists of two swimlanes (lines 2 and 26) and two columns (lines 7f., 19f.,

and 30f., 42f. resp). While the first lane contains one card (line 9ff.) which is located in the column labeled `[Column Title#1]` (line 7f.), the second column within this lane has no card (line 21). The second lane contains two cards, each of which are grouped in a different column (lines 32ff. and 44ff.)

```
1  {
2    "[SWIMLANE IRI#1]": { // swimlane identifier
3      "title": "[Swimlane Title#1]",
4      // start of the react-trello format
5      "lanes": [ // "columns" within the current swimlane
6        {
7          "title": "[Column Title#1]",
8          "id": "[Column IRI#1]",
9          "cards": [
10           {
11             "title": "[Card Title#1]",
12             "id": "[Card IRI#1]",
13             "description": "...",
14             "modified": "[ISO 8601 Timestamp]"
15           }
16         ]
17       },
18       {
19         "title": "[Column Title#2]",
20         "id": "[Column IRI#2]",
21         "cards": []
22       }
23     ]
24     // end of the react-trello format
25   },
26   "[SWIMLANE IRI#2]": {
27     "title": "[Swimlane Title#2]",
28     "lanes": [
29       {
30         "title": "[Column Title#1]",
31         "id": "[Column IRI#1]",
32         "cards": [
33           {
34             "title": "[Card Title#2]",
35             "id": "[Card IRI#2]",
36             "description": "...",
37             "modified": "[ISO 8601 Timestamp]"
38           }
39         ]
40       },
41       {
42         "title": "[Column Title#2]",
43         "id": "[Column IRI#2]",
44         "cards": [
45           {
46             "title": "[Card Title#3]",
47             "id": "[Card IRI#3]",
48             "description": "...",
49             "modified": "[ISO 8601 Timestamp]"
50           }
51         ]
52       }
53     ]
54   }
55 }
```

Code 5.10: Template specification for multiple swimlanes within the data model.

5.2.2 RMB Specification

This subsection provides the specification for the UI components that are used by the Resource Management Board. As outlined in the non-functional requirement NFR_2, eccenca has elaborated a style guide for its visual components based on Material Design Lite (MDL).[30] The corresponding specifications are publicly available[31] and will be referenced throughout this section.

Cards

To conform to eccenca's visual appearance and to allow an arbitrary amount of elements to be depicted on a card's surface, eccenca's card component[32] replaces react-trello's default card component.

Figure 5.4 illustrates the design and positioning specification for the card component. Figure 5.4 (A) showcases all defined card component resources that a user may specify within the board configuration. However, the only mandatory elements on a card are its title and its resource identifier, as depicted by Figure 5.4 (B). This means, if an optional element (e.g., the description) is not defined within a resource, the card does not reserve a blank area. In other words, the card's height matches its content. Moreover, the card specification contains different sections. For this work the following sections are relevant: Figure 5.4 (1) defines the `<CardTitle>`, (2) `<CardContent>` is used for the descriptive text, (3) and (4) use the element `<CardActions>`, since by design it creates a horizontal divider, and is set in a smaller font size.

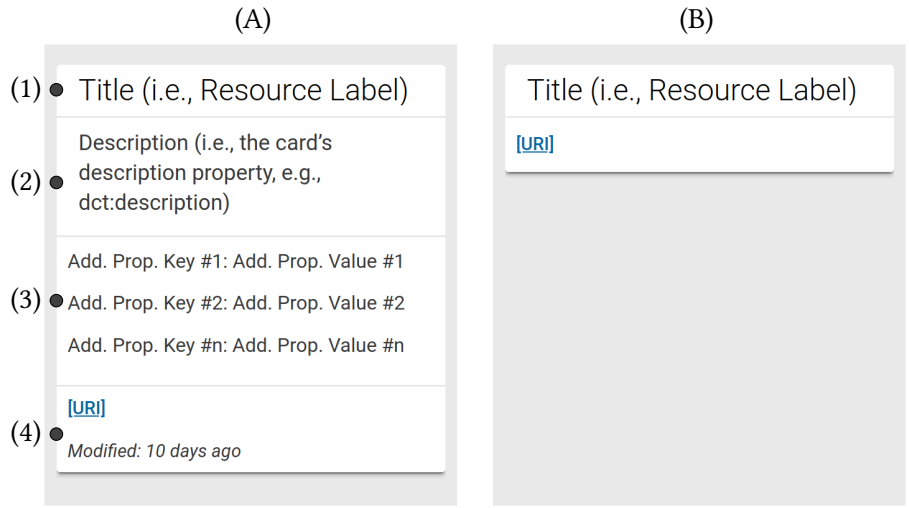

Figure 5.4: Card style and positioning specification.

SHACLINE Whenever a user clicks a card, a modal window[33] appears and provides more information about the clicked resource. More specifically, the board requires an event handler that listens to card on-click events. This means, that the handler should receive the card's resource identifier and the corresponding card's graph. These two information are provided to eccenca's SHACLINE component, which creates a SHACL-defined document-like view on the provided resource and allows the user to manipulate data safely. The SHACLINE component should be embedded within the modal window.

[30] MDL is a UI component library by Google. GitHub: `https://github.com/google/material-design-lite`.
[31] eccenca UI Repository: `https://github.com/eccenca/ecc-gui-elements`.
[32] Card specs. `https://github.com/eccenca/ecc-gui-elements#card`.
[33] A modal window is *"[…] a window overlaid on either the primary window or another dialog window. Windows under a modal dialog are inert. That is, users cannot interact with content outside an active dialog window." (W3C, 2015)*

Board Overview

Figure 5.5 specifies the structural layout of the Resource Management Board using the adjust swimlane model and eccenca UI elements. Foremost, Figure 5.5 (1) uses the `<SelectBox>` element[34] allowing users to view all boards by their title. Moreover, the initial state of the application only displays the `<SelectBox>` element waiting for the user to select a board. After a board got selected, the remaining elements (2) to (6) appear below the selection box. Specifically, the elements (2) and (3) display the board's title and description. The placeholder (4) should provide information or warnings about the state of the board (using the `<InfoBox>`[35] element). Element (5) provides the label of the current swimlane. If swimlanes are not defined, this element does not appear. Lastly, the swimlane container (6) iterates over all swimlane objects defined in the data (at least once). It groups cards and columns in the way as specified before.

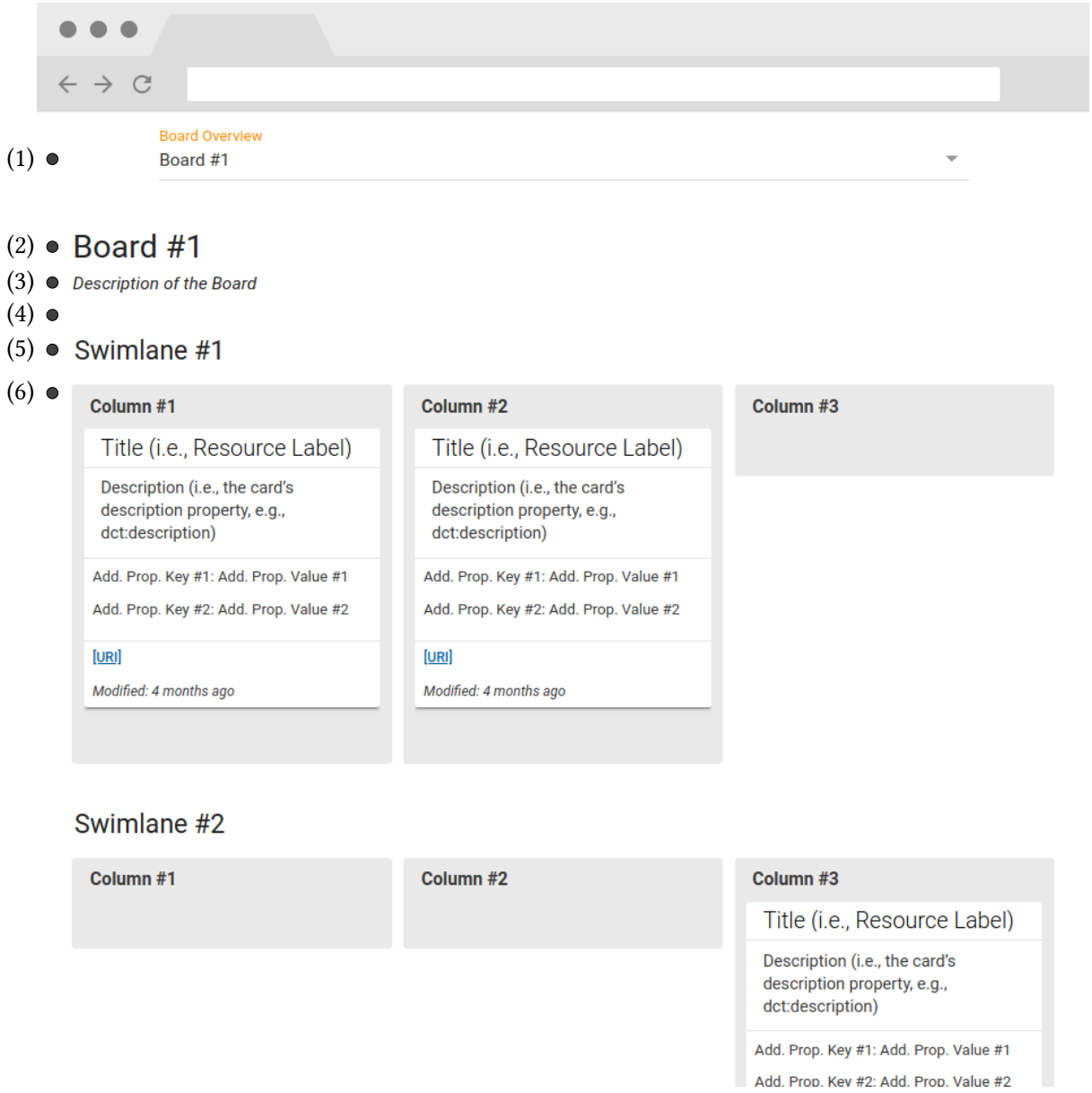

Figure 5.5: Layout Specification for the RMB using react-trello and MDL cards.

[34] SelectBox specs. https://github.com/eccenca/ecc-gui-elements#selectbox.
[35] InfoBox specs. https://github.com/eccenca/ecc-gui-elements#alert-error-info-success-and-warning.

5.3 Query Strategy

Figure 5.6 provides a high-level abstraction of the Resource Management Board. The process flow can be divided into three different views (i.e., SPARQL, data processing, and user interface). While the user interface has been outlined in the previous section, data processing will be introduced in the next section. The current section focuses on the SPARQL query strategy. When users interact with the RMB, the underlying SPARQL processes for requesting or updating data are composed automatically, since users are not supposed to form a query by themselves. This section provides the specifications for this SPARQL abstraction, specifically focusing on five central stages within the query process, as denoted by the letters $\mathcal{A}$ to $\mathcal{E}$ within the SPARQL lane in Figure 5.6.

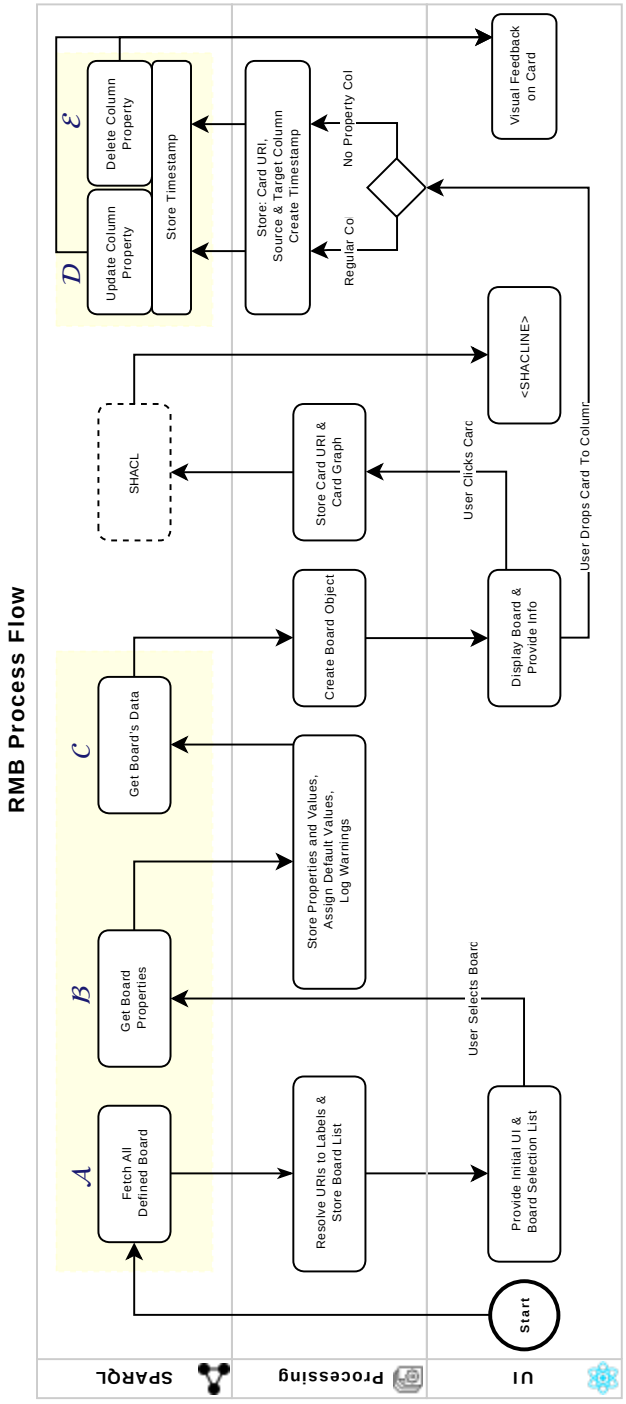

Figure 5.6: Process flow of the RMB separated by three views (i.e., SPARQL, data processing, and UI).

All queries will use two interfaces developed by eccenca; Foremost, the *SPARQL logic store* (or sparqlChannel in the following), which requires an input string that contains the actual SPARQL query, and returns a response object. The response object may be set to `responseJson` to receive a JSON structure as the output format. The second interface is the *TitleHelper logic store* (or titleHelper in the following) which—in its default configuration—requires a single resource (or an array of resources) as an input argument, and returns a key-value object, whereas a key is an IRI and value is the corresponding label.

The following three subsections represent the query stages (i.e., $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$) in order to receive and send the required data for rendering the board.

5.3.1 $\mathcal{A}$ — Fetch All Defined Boards

Initially, the prototype should send an asynchronous request to fetch all available boards (i.e., board configurations). Code 5.11 illustrates this query, which essentially requesting resources that are of the type `rmb:BoardConfig` (line 3), as earlier specified in the model for the board configuration. Line 4 requests the board's description if defined.

```
1 SELECT DISTINCT ?board ?descr
2 WHERE {
3   ?board a <https://vocab.eccenca.com/rmb/BoardConfig> .
4   OPTIONAL { ?board <http://purl.org/dc/terms/description> ?descr . }
5 }
```

Code 5.11: Requesting all boards that are of type `rmb:BoardConfig`.

Code 5.12 illustrates the response object that contains the board configuration of the first use case, as earlier defined in Code 5.4. The *titleHelper* would reveal the title as also defined in former board configuration (i.e., FOAF Term Status). Nevertheless, the list of board titles should be used in the `<SelectBox>` element, which was shown previously in Figure 5.5 (1).

```
1 [{
2   "board": {"type":"uri",    "value":"https://vocab.eccenca.com/rmb/foaf-term-status"},
3   "descr": {"type":"literal","value":"Manages terms by their status in the FOAF namespace"}
4 },{
5   // other configurations
6 }]
```

Code 5.12: Exemplary response object of the request in Code 5.11.

5.3.2 $\mathcal{B}$ — Get Board Properties

Selecting a board from the `<SelectBox>` immediately triggers another query requesting all properties within the selected board configuration. That is, the properties and their values as requested in the first functional requirement FR_1 or similar in Table 5.1. This query, however, requires the IRI of the board the user selected. Code 5.13 illustrates the request for all properties and values of a specific board, using the placeholder variable `boardIRI` in line 3.

```
1 SELECT DISTINCT ?p ?o
2 WHERE {
3   <boardIRI> ?p ?o .
4 }
```

Code 5.13: Requesting all defined properties of a specific board (i.e., the variable `boardIRI`).

Consider the scenario that the current board configuration contains the information of the first use case, as illustrated in Code 5.4 or Figure 5.2 (B), and the former request replaces the placeholder `boardIRI` with `rmb:foaf-term-status` (Code 5.13 line 3). As a result, the sparqlChannel would return a response object similar as depicted by Code 5.14.

This output provides two important information: (1) the developer knows what specific properties have been defined within the selected board configuration (i.e., lines containing `"p"`), and (2) what their corresponding object value is (i.e., lines containing `"o"`). In any case, the response object should match the board configuration regarding their contents, as can be seen in Code 5.14 and Figure 5.2 (B).

```
1  [{
2    "p": {"type": "uri",    "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"},
3    "o": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/BoardConfig"}
4  },{
5    "p": {"type": "uri",    "value": "http://www.w3.org/2000/01/rdf-schema#label"},
6    "o": {"type": "literal","value": "FOAF Term Status"}
7  },{
8    "p": {"type": "uri",    "value": "http://purl.org/dc/terms/description"},
9    "o": {"type": "literal","value": "Manages terms by their status in the FOAF namespace"}
10 },{
11   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsGraph"},
12   "o": {"type": "uri",    "value": "http://xmlns.com/foaf/0.1/"}
13 },{
14   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsClass"},
15   "o": {"type": "uri",    "value": "http://www.w3.org/2002/07/owl#Class"}
16 },{
17   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsClass"},
18   "o": {"type": "uri",    "value": "http://www.w3.org/2002/07/owl#DatatypeProperty"}
19 },{
20   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsClass"},
21   "o": {"type": "uri",    "value": "http://www.w3.org/2002/07/owl#ObjectProperty"}
22 },{
23   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsColumnProperty"},
24   "o": {"type": "uri",    "value": "http://www.w3.org/2003/06/sw-vocab-status/ns#term_status"}
25 },{
26   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsLaneProperty"},
27   "o": {"type": "uri",    "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"}
28 },{
29   "p": {"type": "uri",    "value": "https://vocab.eccenca.com/rmb/cardsDescriptionProperty"},
30   "o": {"type": "uri",    "value": "http://www.w3.org/2000/01/rdf-schema#comment"}
31 }]
```

Code 5.14: Exemplary response object of the request in Code 5.13. The response matches the board configuration in the *DataManger*. In this instance, the response object matches the config of Figure 5.2 (B).

At this stage, developers can check the existence of all properties and may add default values for missing properties, as specified in Table 5.1. For example, Code 5.14 does not contain a `boardLimit` property. However, as outlined in FR_2 (Default Values), it is beneficial to set a limit implicitly, since this will avoid loading too many cards on the board which would likely causes the browser to stall. A default limit of 100 can be considered as a reasonable maximum amount of cards. If this is not high enough, users may set an explicit board limit in the board configuration to override the default value. Lastly, the modified property was not part of the response object. However, since the modified property is used to store a timestamp in the event of dropping a card to a column, it is always justifiable to use an implicit modified property in case the response object lacks its definition. The property `dct:modified` is a suitable candidate for this task.

At this stage, it should also be checked if the board contains all the required elements in order to render the board. As indicated in FR_1 and Table 5.1, the response object should contain at least a name, the underlying card's graph, the card classes, and the column property. If anything is missing, the user should receive a warning provided in the `<InfoBox>` element, as shown in Figure 5.5 (4).

5.3.3 *C* — Get Board's Data

The last stage is requesting the actual cards that are displayed within the board, along with their requested properties. Code 5.15 provides a generic template specification to request this data. In the stage of data processing, the placeholder variables (depicted in red) need to be replaced with the corresponding resources of the former response object (i.e., Code 5.14). However, in the case of absent resources, the entire placeholder line within the where clause and the specific placeholder within the projection variables (lines 1 and 2) should be removed in order to form clean query.

The keyword `SAMPLE` (lines 1 and 2) is an aggregate function that returns an arbitrary value from a multiset passed to it. For example, if there are multiple column values (e.g., *stable* and *unstable*), an arbitrary single selection gets returned.[36] Although this event is likely an indication for broken or invalid data, it should be guaranteed that the response only contains atomic values.[37]

At first glance, using the keyword `OPTIONAL` for the `cardsColumnProperty` (line 7) may seem like an error; however, the optional clause is used on purpose, since only the board configuration requires the definition of that property. For a resource, on the other hand, it is not a necessity to contain the requested property. An example of this scenario is provided in the second use in this work (see page 12). In this example, the column property (i.e., `vs:term_status`) was defined. However, the UNESCO terms did not contain this property, as the purpose of this scenario was to assign it from scratch. Therefore, the *no property* column was listed as a functional requirement in FR_8. Note that this also applies to the `cardsLaneProperty` (line 8), as resources will fallback in the *Everything Else* swimlane (see FR_{10}).

```
1  SELECT DISTINCT ?card (SAMPLE(?columns) AS ?column){laneSample}
2    (SAMPLE(?descriptions) AS ?description)(SAMPLE(?modifieds) AS ?modified){additionalValues}
3  FROM <cardGraph>
4  WHERE {
5    ?card a ?class.
6    FILTER (?class IN (<cardsClass>)) .
7    OPTIONAL { ?card <cardsColumnProperty> ?columns. }
8    OPTIONAL { ?card <cardsLaneProperty> ?lanes. }
9    OPTIONAL { ?card <cardsDescriptionProperty> ?descriptions. }
10   OPTIONAL { ?card <cardsModifiedProperty> ?modifieds. }
11   {additionalPropertiesPlaceholder}
12 }
13 GROUP BY ?card
14 LIMIT boardLimit
```

Code 5.15: Template specification for requesting the content of the board depending on the selected board and card component resources. Variables that need to be replaced or removed are depicted in red color.

For illustration purposes, Code 5.16 demonstrates the final query after the replacement process of the former SPARQL template. As stated, the resources used in Code 5.16 stem from the prior response

[36] Thus, the variable names are mapped from plural to a singular form.

[37] This operation is similar to the process of achieving a *first normal form* (1NF) in the field of database normalization. In other words, a table only consists of atomic columns, which means all cells have a single value.

object of stage $\mathcal{B}$ (see Code 5.14). Moreover, the placeholders for additional properties were removed, since they were not defined in the response object.

```
SELECT DISTINCT ?card (SAMPLE(?columns) AS ?column)(SAMPLE(?lanes) AS ?lane)
  (SAMPLE(?descriptions) AS ?description)(SAMPLE(?modifieds) AS ?modified)
FROM <http://xmlns.com/foaf/0.1/>
WHERE
{
  ?card a ?class.
  FILTER (?class IN (
    <http://www.w3.org/2002/07/owl#Class>,
    <http://www.w3.org/2002/07/owl#DatatypeProperty>,
    <http://www.w3.org/2002/07/owl#ObjectProperty>
  )) .
  OPTIONAL { ?card <http://www.w3.org/2003/06/sw-vocab-status/ns#term_status> ?columns. }
  OPTIONAL { ?card <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?lanes. }
  OPTIONAL { ?card <http://www.w3.org/2000/01/rdf-schema#comment> ?descriptions. }
  OPTIONAL { ?card <http://purl.org/dc/terms/modified> ?modifieds. }
}
GROUP BY ?card
LIMIT 100
```

Code 5.16: SPARQL sample to retrieve the board's data of the first use case.

Code 5.16 returns a response object that contains all 75 FOAF terms.[38] Code 5.17 shows an excerpt of four resources, which are—regarding their content—similar to the mockup provided on page 10. Ultimately, this array of resources represents the cards of the Resource Management Board.

The subsection 6.2.1 (Data Transformation) will highlight important aspects when converting the final response object (i.e., Code 5.17) into the prior defined target data model (i.e., Code 5.10) in order to satisfy the required format of react-trello.

```
[{
    "card":       {"type":"uri",    "value":"http://xmlns.com/foaf/0.1/Document"},
    "column":     {"type":"literal","value":"stable"},
    "lane":       {"type":"uri",    "value":"http://www.w3.org/2002/07/owl#Class"},
    "description":{"type":"literal","value":"A document."},
    "modified":   {"type":"literal","value":"2019-09-06T09:24:37+02:00",
                                    "datatype": "http://www.w3.org/2001/XMLSchema#dateTime"}
  },{
    "card":       {"type":"uri",    "value":"http://xmlns.com/foaf/0.1/mbox"},
    "column":     {"type":"literal","value":"stable"},
    "lane":       {"type":"uri",    "value":"http://www.w3.org/2002/07/owl#ObjectPropert"},
    "description":{"type":"literal","value":"A personal mailbox, ie. an Internet mailbox..."}
  },{
    "card":       {"type":"uri",    "value":"http://xmlns.com/foaf/0.1/depiction"},
    "column":     {"type":"literal","value":"testing"},
    "lane":       {"type":"uri",    "value":"http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"},
    "description":{"type":"literal","value":"A depiction of some thing."}
  },{
    "card":       {"type":"uri",    "value":"http://xmlns.com/foaf/0.1/knows"},
    "column":     {"type":"literal","value":"stable"},
    "lane":       {"type":"uri",    "value":"http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"},
    "description":{"type":"literal","value":"A person known by this person (indicating..."}
}]
```

Code 5.17: Sample response object of the request in Code 5.16. The object contains four resources (i.e., cards) similar to the mockup of the first use case (see page 10).

[38] To replicate this scenario, you may visit `http://bit.ly/foaf-term-status`, which is linking to the LOV SPARQL endpoint containing the query provided in Code 5.16. The output can be toggled between a raw response (i.e., similar to Code 5.17), and a table view.

Update & Delete Column Properties

The following two subsections will provide the query specifications for column updates. Specifically, an event listener should trigger the process of a column update, whenever a card gets dropped to column. An informal algorithm that handles column updates would be:

(1) When dropping a card, check if the source column AND the target column is equal; then do nothing.
(2) If the source column AND target column are unequal, then check
(2A) if the target column equals a column value other than *no property*; then do update the resource's column property to the target column value (i.e., stage $\mathcal{D}$). If (2A) is not the case, check
(2B) if the target column equals the column value of *no property*; then do delete the resource's column property entirely (i.e., stage $\mathcal{E}$).

Set a Timestamp

Regardless of updating or deleting a property, both stages have in common that they store a timestamp in their process, as can be seen in Figure 5.6. Therefore, Code 5.18 specifies a timestamp template that is applicable in both stages. There are two steps involved for updating the value of the modified property: (1) deleting all existing values and (2) inserting the new value. Moreover, the depicted `DELETE/WHERE` approach (line 1 and 2) guarantees to delete a set of variables.[39] This essentially means that if there are multiple timestamps values, the query will delete all of them.[40] Similar to the prior templates, placeholder variables are depicted in red color. In order to update a value the query needs to know (in descending order): the resource's graph (i.e., `cardsGraph`), its resource identifier (i.e., `cardId`), and the property that needs to be adjusted (i.e., `cardsModifiedProperty`). The timestamp is provided in ISO 8601.[41]

```
1  WITH <cardsGraph>
2  DELETE { <cardId> <cardsModifiedProperty> ?o . }
3  WHERE  { <cardId> <cardsModifiedProperty> ?o } ;
4    INSERT DATA {
5      GRAPH <cardsGraph> {
6        <cardId>
7        <cardsModifiedProperty>
8        "ISO8601Timestamp"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
9      }
10   }
```

Code 5.18: SPARQL template to update the modified property of a resource.

[39] See SPARQL specification: https://www.w3.org/TR/sparql11-update/#deleteInsert.
[40] Although this should not occur though, due to sampling the projection variables beforehand (see Code 5.15).
[41] The ISO 8601 is a standardized time format, which has the following pattern: 2019-12-20T08:30:00+01:00. That is expressing the date of December 20, 2019, at a time of 8:30:00, and a one hour time offsets from UTC.

5.3.4 $\mathcal{D}$ — Update Column Property

The query for updating a timestamp has a similar pattern as the query for updating a column property. This means that all instances of `cardsModifiedProperty` will be replaced by `cardsColumnProperty`. However, in the stage of data processing, the resource for `newColumnValue` (line 8) should be type-checked, since it may either be an IRI or a literal. Hence, it is either surrounded by angle brackets (<>) or double quotes (""), respectively.

```
1  WITH <cardsGraph>
2  DELETE { <cardId> <cardsColumnProperty> ?o . }
3  WHERE  { <cardId> <cardsColumnProperty> ?o } ;
4    INSERT DATA {
5      GRAPH <cardsGraph> {
6        <cardId>
7        <cardsColumnProperty>
8        <"newColumnValue"> .
9      }
10   }
```

Code 5.19: SPARQL template to update the column property of a resource.

5.3.5 $\mathcal{E}$ — Delete Column Property

If a user drags a card to the *no property* column, the corresponding column property of that resource gets deleted using Code 5.20.

```
1  WITH <cardsGraph>
2  DELETE { <cardId> <cardsColumnProperty> ?o . }
3  WHERE  { <cardId> <cardsColumnProperty> ?o } ;
```

Code 5.20: SPARQL template to delete the column property of a resource.

6 Implementation

This chapter highlights aspects of the developmental process of the RMB (section 6.2), and after that, screenshots will demonstrate the final prototype showcasing the four use cases (section 6.3).

6.1 Technology Stack

The current project was encapsulated within eccenca's existing ecosystem. Due to the user-centric nature of this work, most technologies involved during the development of the prototype were all based around front-end frameworks and their corresponding build workflow. However, as the prototype relied on the solutions provided by eccenca's *Corporate Memory*, some interfaces were back-end based, such as the triplestore database, and authorization functionality.

As indicated in the specification section of the previous chapter, the prototype has been developed using React.[42] That is a JavaScript library for building user interfaces, having its strengths in performance, and its appealing approach to use JavaScript render the HTML DOM. Specifically, React refers to this method as Javascript XML (JSX). Code 6.1 illustrates the basic concept of React.

```
/* React Component */
render() {
  const myIRI = 'http://example.com/ns/rmb/thesis#react-example';
  const myJSX = <p>Hi, I am JSX! My IRI is: {myIRI}</p>
    return (
      <div>
        {myJSX}
      </div>
    );
}
```

Code 6.1: Basic React example.

The React coding guidelines, as well as the React community in general, encourage developers to use a modern JavaScript syntax. Therefore, the usage of JavaScript represented a fundamental requirement during the development process. The CSS was composed using the preprocessor language SCSS, and the entire front-end project was streamlined using Gulp, which is a build system and task runner[43] when compiling the code.

To establish (a) the underlying board configuration graph and (b) the query strategy, technologies around the *Semantic Web Stack*[44] were used. Both procedures were specified in the previous chapter. More detailed information around web technologies and the semantic web stack can be found here:

- https://developer.mozilla.org/en-US/docs/Web
- https://jena.apache.org/tutorials/index.html

[42] GitHub Repository: https://github.com/facebook/react.
[43] This involves tasks around minification, concatenation, cache busting, or linting.
[44] More information: https://en.wikipedia.org/wiki/Semantic_Web_Stack.

6.2 Development Process

The agile development process of the current project was held in an iterative and incremental manner. The development of the prototype was split into small and manageable pieces, undergoing multiple iteration cycles. In irregular meetings, novel features or enhancements were discussed and specified. In the step-wise development process, novel features were first tested and implemented before the next iteration cycle. The cycles were repeated until a satisfying prototypical application state was reached for this work.

Besides learning the involved technologies, one of the most time-consuming tasks of the current project was the data transformation process. That is the conversion from the response object of the board's data (Code 5.17) to the target data model (Code 5.10). The next subsection gives an outline of the main obstacles in this regard and illustrates the algorithm used to transform the data from the response object to the target data model.

6.2.1 Towards the Target Data Model

Comparing the column and lane resources in Code 5.17 (e.g., lines 3 and 4) reveals that both differ in their type: while columns are of type `literal`, lanes are of type `uri`. This implies that both require different processing steps. The target data model, for example, requires one title for each swimlane. This means that a single title will be used to label every individual segment of the swimlane container, as specified in Figure 5.5. However, given that swimlanes are resources—in this particular case—their label needs to be first resolved using eccenca's titleHelper. Moreover, if a certain resource does not contain a swimlane property while other resources do, even more processing steps are required.

To summarize this 'mixed-type issue,' Table 6.1 provides a lookup table for all possible combinations. The corresponding function[45] contains the transformation algorithm, and also refers to the notation depicted in the table. Ultimately, this function is responsible for converting the response object of stage C into the target data model. The 3×3 matrix only differentiates between column and lane types, since these are the only board component resources that can vary in their type.[46] In the table, the type null expresses the scenario that a resource does not contain the requested swimlane and/or column property. The boxes around the entries illustrate how the algorithm branches the processing.

		Column Type		
		Literal	Uri	Null (×)
Lane Type	Literal	LL	LU	L×
	Uri	UL	UU	U×
	Null (×)	×L	×U	××

Table 6.1: Lookup table for the mixed type processing algorithm. The boxes around the matrix entries illustrate explicit (black) and implicit cases (gray) and corresponding processing steps for a single resource, based on the response object provided in Code 5.17.

[45] That is the function within the file `getInitialBoardState.js` located at `src/util/` attached digitally to this work.
[46] Cards are always resources (that means they are always of type `uri`); therefore, they are not considered in this matrix.

To provide an example of how to read the lookup table, reconsider the lane and column types of Code 5.17 (i.e., `uri` and `literal`, resp.). The corresponding matrix entry in Table 6.1 explicitly yields to UL. However, if a resource does not contain the requested column or swimlane property, both entries at the horizontal and vertical margin of the table need to be considered implicitly. For the first use case this means, that if a FOAF term has no `term_status` property, it would be allocated in the *no property* column, meaning that it would be processed using the steps of U×. Similarly, if a FOAF term has no type property defined, it would be allocated in the *Everything Else* swimlane, meaning that it would be processed by the ×L condition. Furthermore, and regardless of the explicit case, if a resource lacks the requested column and lane property, the processing steps of ×× would be applied. That would, for example, be the case if a card is located in the *no property* column within the *Everything Else* swimlane.

In conclusion, the algorithm determines the correct processing step by selecting the explicit matrix entry and selecting the corresponding horizontal and vertical margin entries. Lastly, the case ×× should always be checked and, if applicable, executed. To give one final example for this case: If a single resource contains the properties corresponding to the explicit entry LU, then the following implicit cases are also possible in this particular board: L×, ×U, and ××.

Due to the complex structure of the branching process, various response objects have been used to test the reliability of the transformation process.[47]

[47] See the files `ll.js`, `lu.js`, `ul.js`, and `uu.js` located at `src/util/demos/`.

6.2.2 Project and Component Structure

This subsection provides an overview and brief description of the project structure. For the sake of clarity, some utility files were not included in this list. However, all files are accessible in eccenca's repository or on the attached microSD card.

src	
components	Grouping all React components
Lane	
CustomCard	
CustomCard.jsx	Custom card specs. (see Figure 5.4), text trimmings, apply MDL styles
Lane.jsx	Represents the lane container (see Figure 5.5), imports CustomCard
SPARQLView	
SPARQLView.jsx	Provides a UI component to review/edit the Board's query (see $FR_{12/13}$)
ShaclineModal	
ShaclineModal.jsx	Provides a modal dialog (i.e., eccenca's SHACLINE) on card click (see FR_{20})
util	
demos	
*.js	Test queries for mixed type cases (ll/lu/ul/uu.js, see Table 6.1, NFR_3)
sparql-mappings	
baseSPARQLStr.js	Template to query the board's data, similar to Code 5.15
generateBoardSPARQL.js	Replace former template placeholders with the requested resources (Code 5.16)
generateLookaheadBoardSPARQL.js	Minified version of the former query; yet, with an implicit limit of +1
getAllBoards.js	Requesting all defined board configurations, similar to Code 5.11
getBoardObjects.js	Fetch all defined properties within the selected board, similar to Code 5.13
JSONtemplate.js	JSON template for the target data model
boardDefaults.js	Define text elements (e.g.*Everything Else* lane or *no property* column)
deleteProperty.js	Function that gets triggered if the target column is the *no property* column
deletePropertyStr.js	Template to delete a property, similar to Code 5.20
getBoardBody.js	Creates a meta object containing the board's data and other information
getInitialBoardState.js	Transforms the former object to the target data model, see subsection 6.2.1
handleColumnUpdates.js	Checks source and target columns, and triggers the update/deleteProperty.js
promises.js	Wraps eccenca's sparql and titleHelperChannel in JavaScript's promise API
updateProperty.js	Function that gets triggered if the target column is a regular column
updatePropertyStr.js	Template to update a property, similar to Code 5.19
updateTimestampStr.js	Template to create a modified property for a resource, similar to Code 5.18
ResourceManagementBoard.jsx	Main React Container

6.3 Workflow

A user or resource manager needs to follow two steps in order to visualize and modify RDF resources within the prototype. First, they need to create or edit a board configuration along with the desired board and card component resources within eccenca's *DataManager*. Then, they can select the board in the prototype, and start to explore or edit resources. Editing resources can either be performed by relocating cards on the board (which would update the resources column property), or by clicking on a card (which would open eccenca's SHACLINE modal window). Since the board configuration was already introduced within section 5.1 (specification), including its visual representation (Figure 5.2), the following section will showcase the final stage of the prototype, as illustrated by a series of screenshots showing different UI components and use cases.

To begin with, Figure 6.1 shows the initial application state, waiting for the user to select a board configuration (i.e., one of the four use cases in the select box). This state corresponds to the process flow of Figure 5.6 after fetching all board configurations (*A*).

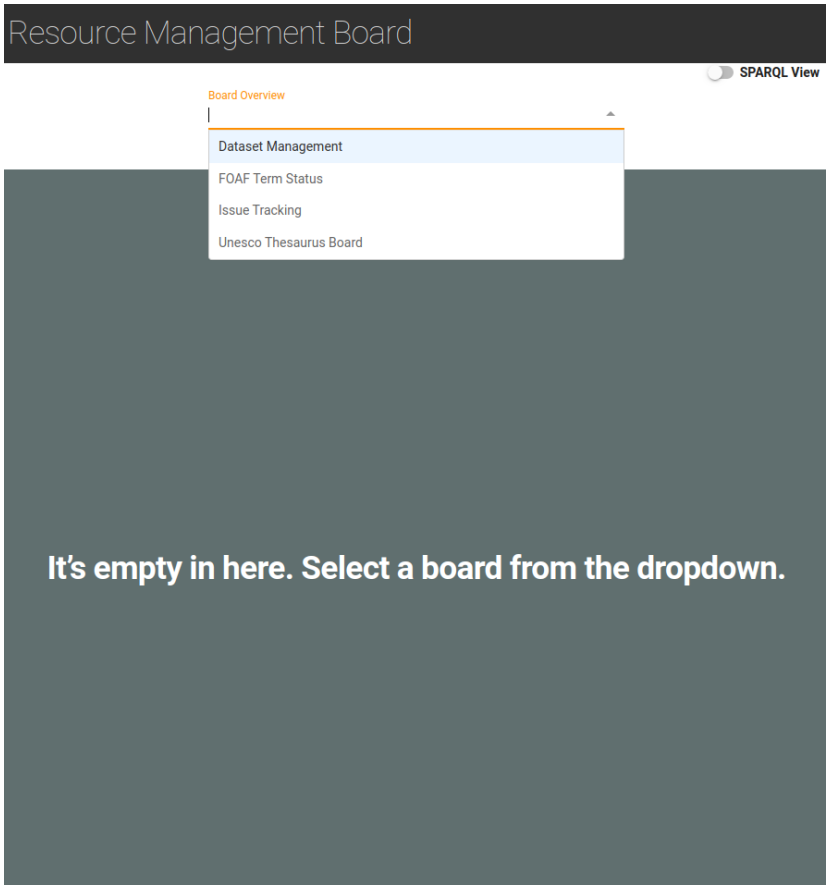

Figure 6.1: Initial board state of the RMB.

Figure 6.2 shows the SPARQL View component, which expands above the board selection box. To reveal this information, the user needs to toggle the SPARQL View at the top right corner. In this instance, the component contains the auto-composed query to request the data for the first use case (FOAF Terms Status). The query corresponds to stage *C* of Figure 5.6 and likewise to Code 5.16.

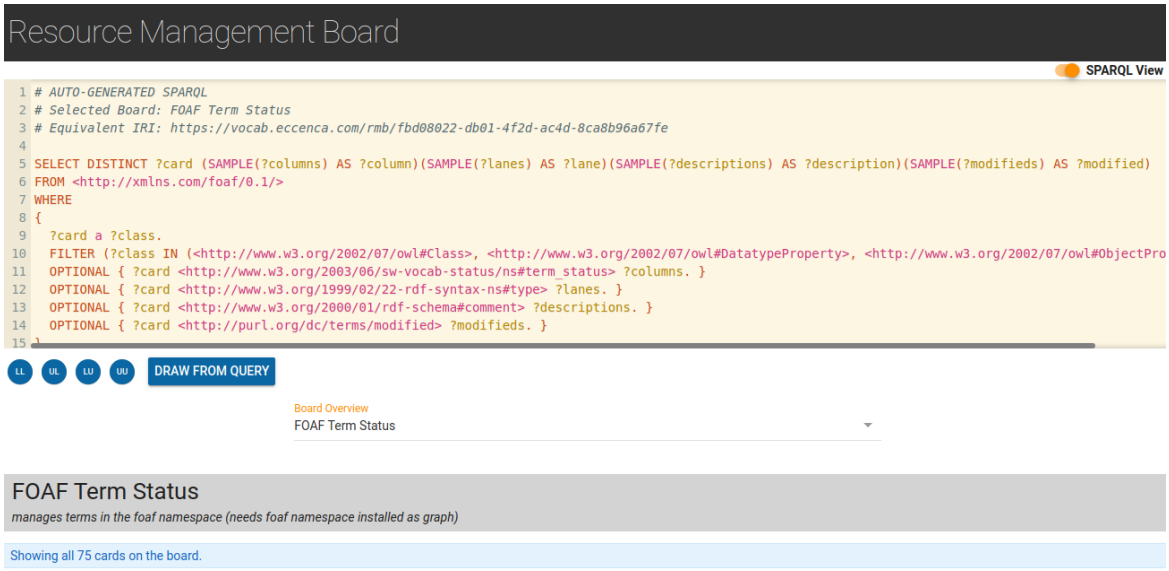

Figure 6.2: SPARQL View component with the auto composed query for the first use case.

Figure 6.3 shows the board for the first use case (FOAF Term Status). Since there was no explicit limit set, the implicit limit (100) is above the number of resources being depicted as cards. Therefore, the infobox provides the corresponding information.

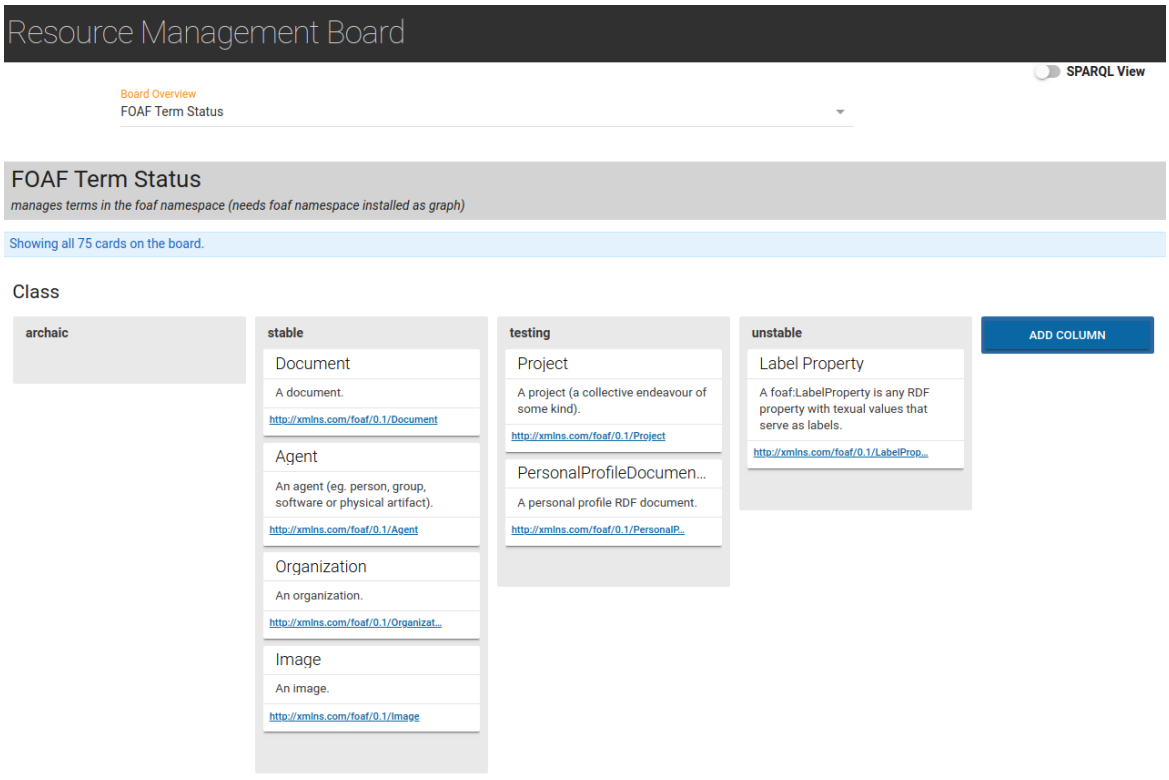

Figure 6.3: RMB of use case 1. Only the first swimlane is depicted.

Figure 6.4 shows the board for the second use case. Since the terms from the UNESKOS graph do not contain the defined column property `vs:term_status`, the cards are allocated in the fallback column *no property*. As outlined in this use case (see section 3.1.1), users may then start to create term statuses from scratch and assign cards correspondingly (as depicted).

Since there was no explicit limit defined, the implicit limit avoids loading over 4.000 cards on the board. However, if a user wants to increase the limit, they can either set a higher explicit limit in the board configuration or use the SPARQL View to increase the `LIMIT` and redraw the board.

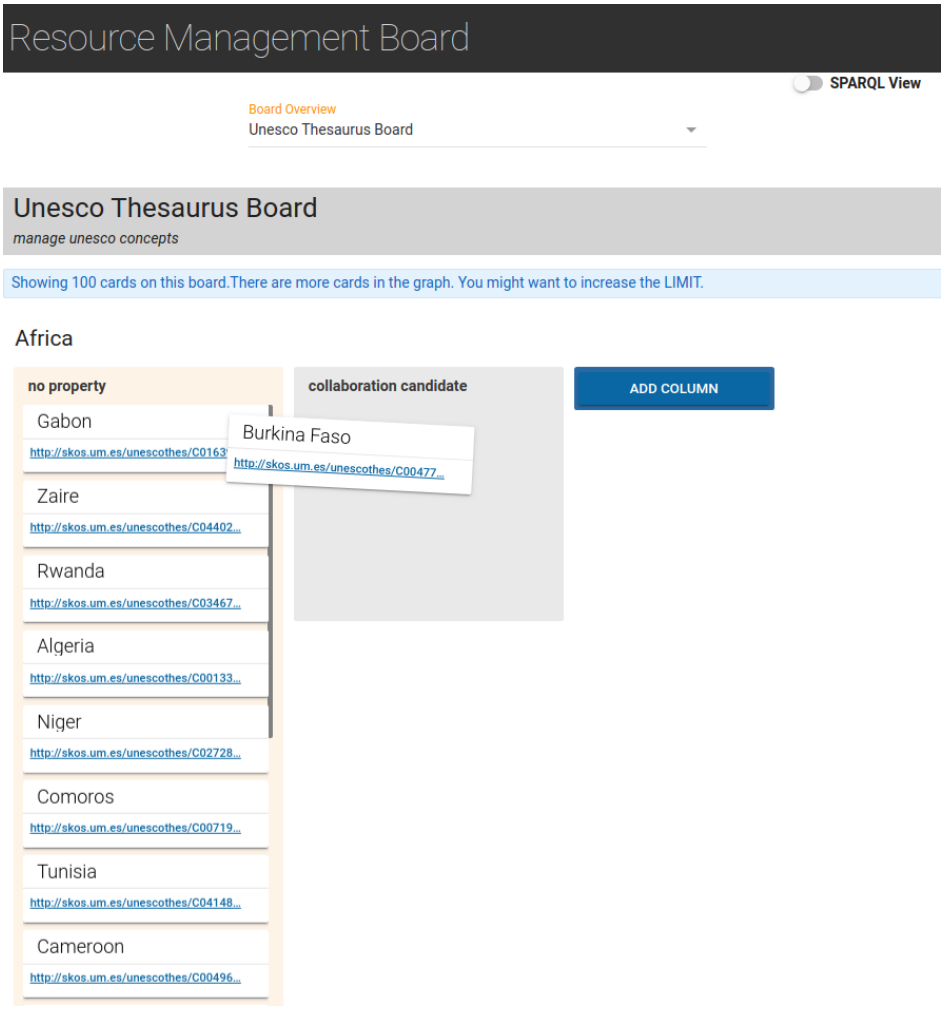

Figure 6.4: RMB of use case 2.

Figure 6.4 shows the board for the third use case. This use case focused on the use of additional properties, as outlined subsection 3.1.2. Additional properties allow to display an arbitrary amount of properties defined in the card's resource. Additional properties are displayed as key-value pairs, as specified in Figure 5.4 (A3).

The card *RND Spending* was moved from the column *published* to the column *needs approval*, which was triggering the real-time events to recolor the card and set the modified timestamp to *just now*, as requested in $FR_{26/27}$. Both events are only temporarily, meaning that after refreshing the board, the card will have its default background color and a relative timestamp (e.g., *Modified: a minute ago*).

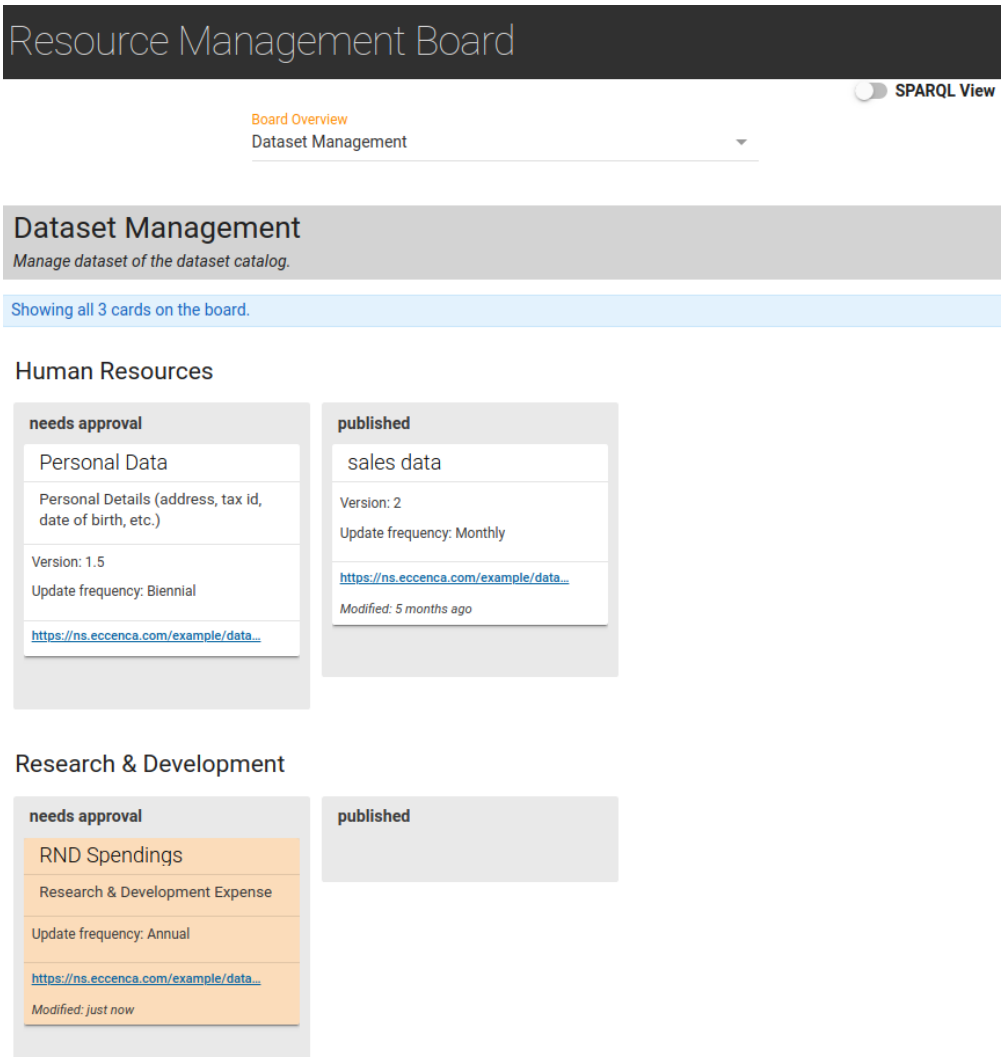

Figure 6.5: RMB of use case 3, demonstrating additional properties and real-time events.

Figure 6.4 shows the board for the last use case, using a subset of the DOAP vocabulary to specify columns and lanes, as outlined in section 3.1.2.

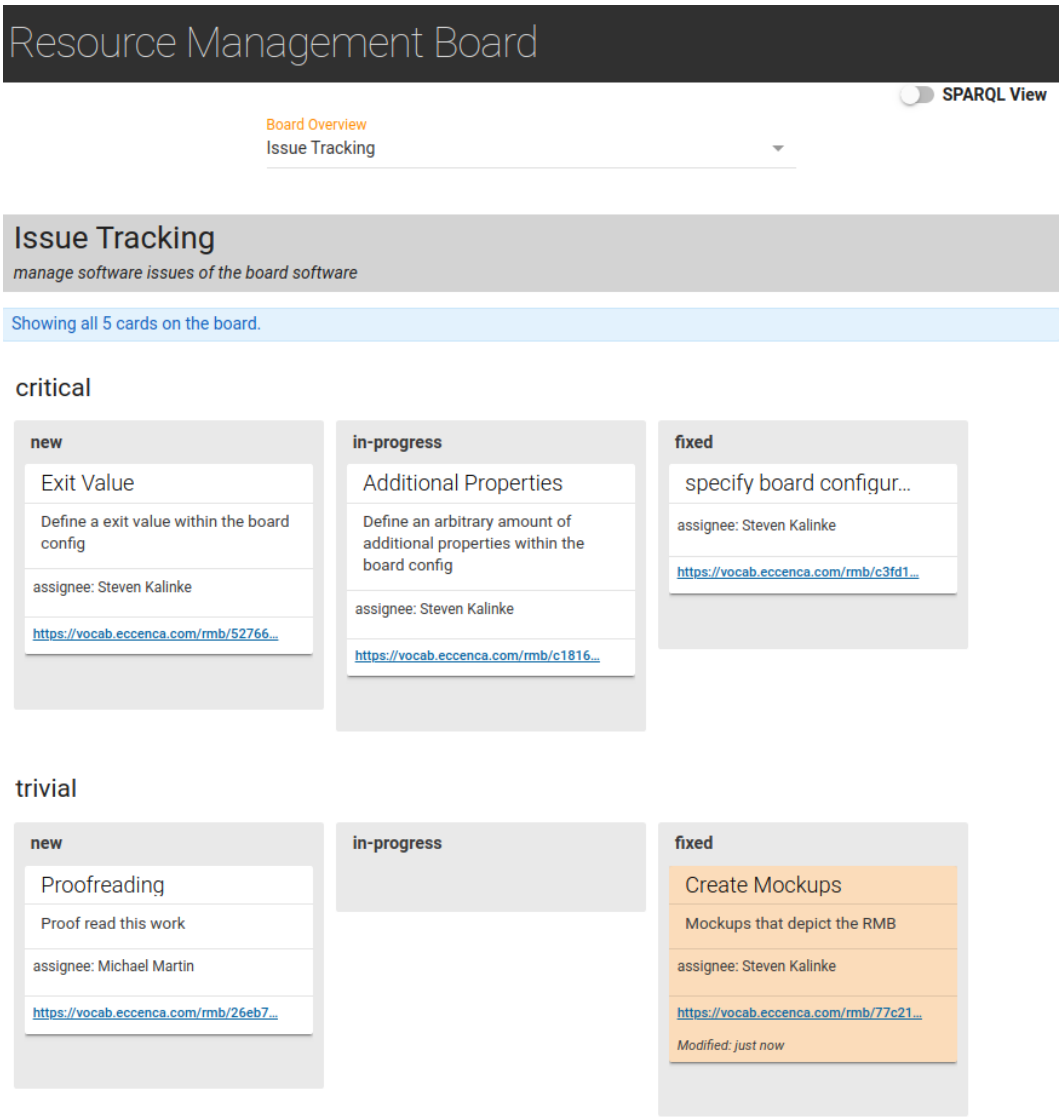

Figure 6.6: RMB of use case 4.

Figure 6.7 demonstrates the implementation of the SHACLINE modal dialog. The card *Exit Value* from Figure 6.6 has been clicked, and the editing mode enabled. The dialog allows to change various properties of the clicked resources. As depicted, the additional property *Assignee* gets changed. After saving the changes (bottom left button), the user can *close* the dialog or *close & redraw* the board (bottom right buttons) to see the changes after the board will automatically refresh. This implementation refers to $FR_{20/21}$.

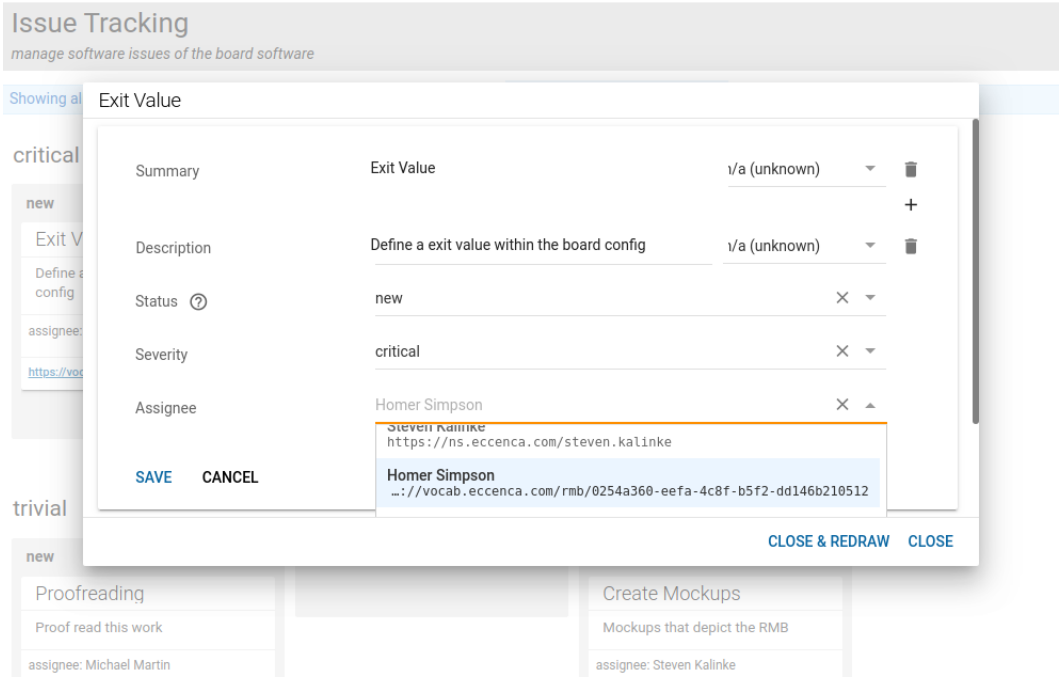

Figure 6.7: Implementation of the SHACLINE modal window. The card *Exit Value* got clicked and its Assignee value gets modified.

7 Evaluation

This chapter evaluates the prototype with regard to the functional and non-function requirements. Building upon this, limitations and future directions will be discussed.

7.1 Strengths of the Current Prototype

The aim of this project was to create a prototypical application that allows users to (1) visualize a certain section of a knowledge graph within a Kanban board, and to (2) modify a specific property by dragging a resource into another column.

I have successfully developed a prototype that reaches both goals and can be used within eccenca's infrastructure. The board configuration graph provides an interface to specify what resources should be displayed and what resources should be modified when relocating cards.

Overall, the Resource Management Board is a novel approach in the field of visualizing and modifying RDF resources. No previous work has so far combined the paradigm and functionality of an intuitive Kanban board environment with RDF visualization. This makes the project an innovative contribution to the field of semantic data exploration.

7.2 Limitations of the Current Prototype & Future Work

Despite the strengths of the developed prototype, the current application state has limitations. Coming back to the functional requirements provided in Table 3.2, the final prototype is to date mainly limited with regard to the following three requirements that have not been (or only partially) implemented:

- FR_7 — Disallow Column/Lane Repositioning
- FR_{11} — Create New Columns
- FR_{13} — SPARQL Editor

FR_7 Although alphabetical sorting might be an appropriate sorting strategy for swimlanes (as requested in FR_7), it is not immediately suitable for columns. In a Kanban board, column labels typically carry a semantic meaning about progress (or progress order). Humans are intuitively capable of sorting such labels. For example, the sequence | *Done* | *ToDo* | *Doing* | does not make direct sense to us, whereas | *ToDo* | *Doing* | *Done* | does.

Nevertheless, there are some use cases in which alphabetical sorting would be possible, namely when column labels do not contain any inherent sequencing information. This would, for example, be the case in the first mockup from the introduction (see Figure 1.2), with the column labels *White* and *Red*. The order of columns would be solely up the user's preference in this case. To give another example: if column labels would express blood types,[48] alphabetic sorting would be principally possible as different blood types do not carry any meaning of progress.

Similar to statistical datatypes, two different types of column semantics can be differentiated, namely categorical and ordinal columns. Categorical column labels are more likely to be reordered

[48] The work of Bursa et al. (2017) extends FOAF by a *Blood Ontology*, see https://www.researchgate.net/publication/319633737_BloodHealthFOAF_Extending_FOAF_with_Blood_Ontology.

as compared to columns with ordinal labels, as different users have different preferences. Therefore, in order to obtain a meaningful progress flow, users should be allowed to rearrange columns, which was performed in Figure 6.6 to maintain a meaningful flow. A central limitation of the prototype is that even though users can change the column order, it would not be persistent after refreshing the board, meaning that it would change back to the initial (alphabetical) order.

An important future direction would be an application that allows users to store their preferred column position, regardless of any column semantics. To address this current limitation, the *Ordered List Ontology* (OLO) could be used, as it provides an index property,[49] that uses a positive integer to store a position. The new column position could be retrieved by react-trello using the provided method `handleLaneDragEnd`.[50] The board configuration itself might be a suitable place to store this sequencing information, compared to cards, which would create unnecessary noise in card resources.

FR_{11} The current application state is furthermore limited in that it only allows to create new columns if the column value is of type `literal`. If column values are of type `uri`, the *Add Column* button will not be displayed (see Figure 6.3 and Figure 6.4 for corresponding examples with literal-typed columns, and Figure 6.5 and Figure 6.6 for examples with uri-types columns). While it is relatively effortless to resolve the label of a resource, it is challenging to do the opposite: A user enters a new column value as a string, and the corresponding resource needs to be found. This approach is prone to errors in many ways, as there is virtually an unlimited amount of possible character combinations a user might enter, and moreover, a single label may refer to multiple resources (e.g., `rdfs:Class` and `owl:Class` share the same label (i.e., *Class*), despite being different resources). An alternative solution would be to provide text suggestions to the user based on predefined column values.

FR_{13} If users want to review the SPARQL query, they can toggle the corresponding UI element to reveal the *SPARQL View* (see Figure 6.2 and FR_{12}). This component reflects the auto-composed query of stage C in Figure 5.6, which is responsible for retrieving the elements that are displayed on the board. The possibility to make changes in this field (FR_{13}) is in the current implementation restricted to the SPARQL `LIMIT` value. The current implementation works by a regular expression check for a change of the limit's value within the *SPARQL View*. If a user changes this value and clicks the *Draw from Query* button, the board will refresh while respecting the new board limit. Future work should extend this feature by regex-checking for other board and card component resources.

Exit Value, Exit Column, & Dwell Time

Another limitation of the current prototypical state is that it does not allow cards to disappear from the board. As an example, consider the use case of issue tracking (see Figure 6.6): When ordering the columns sequentially, the last column in this board has the label *fixed*. Since cards cannot progress beyond this last column, it would eventually clutter up with cards.

One possible approach to address this issue would be to define an *exit value* within the board configuration, whereby the property of the exit value needs to match the config's column property.

[49] http://purl.org/ontology/olo/core#index
[50] See https://github.com/rcdexta/react-trello/blob/master/README.md#callbacks-and-handlers.

The exit value, in this case, would be *fixed* (or rather the resource `dbug:fixed`), matching the config's column property (i.e., `dbug:status`). Conceptually, this board configuration would designate the '*fixed* column' to the *exit column*.

Furthermore, the board configuration needs to define a *dwell time* property,[51]. This property accepts an integer value, which expresses the number of days a card is allowed to dwell in a column before it is supposed to disappear. That means that the exit value, dwell time, and a card's modified property indicate whether a resource should be depicted on the board or not.

Information on Breaking Changes

The following two breaking changes need to be considered for the future development of the current prototype.

React 15.x As of 2019, eccenca's front-end codebase is below React version 16. This means that react-trello version 2.0.7 is the last supported version working with React 15.x. This is due to react-trello's dependency of styled-components, which is a library to write and manage CSS in JavaScript. Nevertheless, react-trello 2.0.7 requires styled-components 3.4.10 which, in turn, have a react peer dependency of: `"react": ">= 0.14.0 < 17.0.0-0"`,[52] (which is within the boundaries of React 15.x). This will change with react-trello version 2.0.8 since it requires version 4.0.3 of styled-components with a react peer dependency of: `"react": ">= 16.3.0"`,[53] which will break the board.

Custom Cards Update From react-trello version 2.1 to 2.2, there have been breaking changes regarding the definition of custom cards. The react-trello creator has released upgrade instructions containing the affected code lines and necessary changes, respectively: `https://github.com/rcdexta/react-trello/blob/master/UPGRADE.md`. Note that this breaking change could not be addressed in the project, as the prior breaking change has restricted development by forcing it to remain on version 2.0.7.

[51] Especially, a `owl:DatatypeProperty`, similar to the board limit definition.
[52] Line 153 at `https://github.com/styled-components/styled-components/blob/v3.4.10/package.json#L153`
[53] Line 135 at `https://github.com/styled-components/styled-components/blob/v4.0.3/package.json#L135`

Appendix

Board Configuration

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dct:  <http://purl.org/dc/terms/> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix rmb:  <https://vocab.eccenca.com/rmb/> .
@prefix dcat: <http://www.w3.org/ns/dcat#> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .
@prefix shui: <https://vocab.eccenca.com/shui/> .
@prefix dbug: <http://ontologi.es/doap-bugs#> .

<http://www.w3.org/ns/shacl#IRI> a <http://www.w3.org/ns/shacl#NodeKind> .
<http://www.w3.org/ns/shacl#Literal> a <http://www.w3.org/ns/shacl#NodeKind> .
rdfs:comment a owl:DatatypeProperty .
skos:Concept a owl:Class .


# RMB Class Definition

rmb:
  a owl:Ontology ;
  rdfs:comment "This includes shapes, classes, properties and configurations for the RMB." ;
  rdfs:label "RMB Configuration" ;
  rdfs:seeAlso
    <https://confluence.brox.de/display/ECCPRODMGMT/Resource+Management+Board>,
    <https://gitlab.eccenca.com/elds-ui/ecc-component-resourcemanagementboard> ;
  dct:creator
    <https://ns.eccenca.com/steven.kalinke> ,
    <https://ns.eccenca.com/stramp> .

<https://vocab.eccenca.com/rmb/BoardConfig>
  a owl:Class ;
  rdfs:label "Board Configuration" .


# General Board Properties

rmb:cardsGraph
  a owl:ObjectProperty ;
  rdfs:comment "single relation pointing to the named graph where the cards are loaded from" ;
  rdfs:label "cards graph" ;
  rdfs:domain rmb:BoardConfig .

rmb:boardLimit
  a owl:DatatypeProperty ;
  rdfs:comment "single integer value used to set the limit of shown cards on a board" ;
  rdfs:label "board limit" ;
  rdfs:domain rmb:BoardConfig .


# Board Component Resources

rmb:cardsClass
  a owl:ObjectProperty ;
  rdfs:comment
    "possible multiple relations pointing to the classes defining the cards on the board" ;
  rdfs:label "cards class" ;
  rdfs:domain rmb:BoardConfig .
```

```
rmb:cardsColumnsProperty
  a owl:ObjectProperty ;
  rdfs:comment "single relation pointing to the property which is used to arrange the cards in columns" ;
  rdfs:label "column property" ;
  rdfs:domain rmb:BoardConfig .

rmb:cardsLaneProperty
  a owl:ObjectProperty ;
  rdfs:comment "single relation pointing to the property which is used to arrange the cards in lanes" ;
  rdfs:label "lane property" ;
  rdfs:domain rmb:BoardConfig .


# Card Component Resources

rmb:cardsDescriptionProperty
  a owl:ObjectProperty ;
  rdfs:comment "single relation pointing to the property which is used for the body of the cards" ;
  rdfs:label "description property" ;
  rdfs:domain rmb:BoardConfig .

rmb:cardsModifiedProperty
  a owl:ObjectProperty ;
  rdfs:comment
    "single relation pointing to the property which is used to save the xsd:dateTime modified timestamp" ;
  rdfs:label "modified property" ;
  rdfs:domain rmb:BoardConfig .

rmb:cardsAdditionalFieldProperty
  a owl:DatatypeProperty, owl:ObjectProperty ;
  rdfs:comment
    "multiple relation pointing to the properties which are used to show additional fields on cards" ;
  rdfs:label "additional field property" ;
  rdfs:domain rmb:BoardConfig .


# SHACL SHAPES

rmb:boardConfigSHACL
  a <http://www.w3.org/ns/shacl#NodeShape> ;
  sh:name "Board Configuration" ;
  sh:property rmb:labelSHACL,
              rmb:descriptionSHACL,
              rmb:cardsGraphSHACL,
              rmb:boardLimitSHACL,
              rmb:cardsClassesSHACL,
              rmb:cardsColumnPropertySHACL,
              rmb:cardsLanePropertySHACL,
              rmb:cardsDescriptionPropertySHACL,
              rmb:cardsAdditionalFieldPropertySHACL,
              rmb:cardsModifiedPropertySHACL ;
  sh:targetClass rmb:BoardConfig ;
  shui:tabName "Board Configuration" ;
  rdfs:label "Board Configuration" .

rmb:labelSHACL
  a sh:PropertyShape ;
  sh:name "Name" ;
  sh:description "the name of the board" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:Literal ;
  sh:order 1 ;
  sh:path rdfs:label ;
  shui:showAlways true ;
  sh:minCount 1 ;
  rdfs:label "Name" .
```

```
rmb:descriptionSHACL
  a sh:PropertyShape ;
  sh:name "Description" ;
  sh:description "descriptive text on the intention of the board" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:Literal ;
  sh:order 2 ;
  sh:path dct:description ;
  shui:showAlways true ;
  sh:minCount 0 ;
  rdfs:label "Description" .

rmb:cardsGraphSHACL
  a sh:PropertyShape ;
  sh:name "Graph" ;
  sh:class owl:Ontology ;
  sh:description "The Knowledge Graph which is used to get the cards from." ;
  sh:maxCount 1 ;
  sh:nodeKind sh:IRI ;
  sh:order 3 ;
  sh:path rmb:cardsGraph ;
  shui:denyNewResources true ;
  sh:minCount 1 ;
  rdfs:label "Graph" .

rmb:boardLimitSHACL
  a sh:PropertyShape ;
  sh:name "board limit" ;
  sh:description "single integer value used to set the limit of shown cards on a board" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:Literal ;
  sh:path rmb:boardLimit ;
  sh:datatype xsd:integer ;
  rdfs:label "board limit" .

rmb:cardsClassesSHACL
  a sh:PropertyShape ;
  sh:name "Class(es)" ;
  sh:class owl:Class ;
  sh:description "which cards should be shown in the board" ;
  sh:nodeKind sh:IRI ;
  sh:order 4 ;
  sh:path rmb:cardsClass ;
  sh:minCount 1 ;
  rdfs:label "Classes" .

rmb:cardsColumnPropertySHACL
  a sh:PropertyShape ;
  sh:name "Column Property" ;
  sh:description "property used to arrange the cards in columns" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:IRI ;
  sh:order 5 ;
  sh:path rmb:cardsColumnsProperty ;
  sh:minCount 1 ;
  shui:uiQuery rmb:843c129c-6d65-4f23-886f-1a1a049c12b9 ;
  rdfs:label "Column Property" .

rmb:cardsLanePropertySHACL
  a sh:PropertyShape ;
  sh:name "Lane Property" ;
  sh:description "the property used to arrange the cards in lanes" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:IRI ;
  sh:order 6 ;
  sh:path rmb:cardsLaneProperty ;
  shui:showAlways true ;
  shui:uiQuery rmb:843c129c-6d65-4f23-886f-1a1a049c12b9 ;
  rdfs:label "Lane Property" .
```

```
rmb:cardsDescriptionPropertySHACL
  a sh:PropertyShape ;
  sh:name "Description Property" ;
  sh:class owl:DatatypeProperty ;
  sh:description "the property which is used to fill the card body (defaults to dct:description)" ;
  sh:maxCount 1 ;
  sh:nodeKind sh:IRI ;
  sh:path rmb:cardsDescriptionProperty ;
  rdfs:label "Description Property" .

rmb:cardsAdditionalFieldPropertySHACL
  a sh:PropertyShape ;
  sh:name "Additional Properties" ;
  sh:description "Relation pointing to the properties which are used to show additional fields on the cards" ;
  sh:nodeKind sh:IRI ;
  sh:path rmb:cardsAdditionalFieldProperty ;
  shui:showAlways false ;
  shui:uiQuery rmb:843c129c-6d65-4f23-886f-1a1a049c12b9 ;
  rdfs:label "Additional Properties" .

rmb:cardsModifiedPropertySHACL
  a sh:PropertyShape ;
  sh:name "Modified Property" ;
  sh:class owl:DatatypeProperty ;
  sh:description "the property which is used to save modified timestamps (defaults to dct:modified)" ;
  sh:nodeKind sh:IRI ;
  sh:path rmb:cardsModifiedProperty ;
  rdfs:label "Modified Property" .
```

Code 7.1: Board Configuration.

Bibliography

Ambler, S. W. (2014). *User Stories: An Agile Introduction.* URL: http://www.agilemodeling.com/artifacts/userStory.htm#InitialInformal, visited on 09/03/2019 (cit. on p. 7).

Anderson, D. (01/2016). *Scrumsplaining #2: There Is No Sense Of Urgency With Kanban.* URL: https://djaa.com/scrumsplaining-2-there-is-no-sense-of-urgency-with-kanban/, visited on 09/07/2019 (cit. on p. 6).

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). "DBpedia - A crystallization point for the Web of Data". In: *Journal of Web Semantics* 7.3. The Web of Data, pp. 154–165. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2009.07.002. URL: http://www.sciencedirect.com/science/article/pii/S1570826809000225 (cit. on p. 4).

Bradner, S. (03/1997). *Key words for use in RFCs to Indicate Requirement Levels (BCP 14).* RFC 2119. Harvard University, pp. 1–3. URL: https://www.ietf.org/rfc/rfc2119.txt (cit. on pp. 7 sq., 27).

Brickley, D. & Guha, R. V. (03/1999). *Resource Description Framework (RDF).* W3C Recommendation. W3C. URL: https://www.w3.org/TR/rdf-schema/ (cit. on p. 4).

Burris, E. (n.d.). *Capturing Requirements with Use Cases.* URL: http://sce2.umkc.edu/BIT/burrise/pl/usecasemodeling/, visited on 09/03/2019 (cit. on p. 8).

DCMI Usage Board (2002). *DCMI Metadata Terms.* Tech. rep. Dublin Core. URL: https://www.dublincore.org/specifications/dublin-core/dcmi-terms/ (cit. on p. 4).

Erickson, J., Maali, F., & Archer, P. (01/2014). *Data Catalog Vocabulary (DCAT).* W3C Recommendation. W3C. URL: https://www.w3.org/TR/vocab-dcat/ (cit. on p. 13).

Fensel, D. (2005). *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential.* Mit Press. MIT Press. ISBN: 9780262562126. URL: https://books.google.de/books?id=zQ34EoZO2IYC (cit. on p. 3).

Gargouri, F. (2010). *Ontology Theory, Management and Design: Advanced Tools and Models: Advanced Tools and Models.* IGI Global research collection. Information Science Reference. ISBN: 9781615208609. URL: https://books.google.de/books?id=8BDcycIkLdwC (cit. on p. 9).

Metadata and Semantics Research (2015). Communications in Computer and Information Science. Springer International Publishing. ISBN: 9783319241296. URL: https://books.google.de/books?id=foGBCgAAQBAJ (cit. on p. 11).

Gaševic, D., Selic, B., Bézivin, J., Djuric, D., & Devedžic, V. (2009). *Model Driven Engineering and Ontology Development.* Springer Berlin Heidelberg. ISBN: 9783642002823. URL: https://books.google.de/books?id=s-9yu7ubSykC (cit. on p. 28).

ISO/IEC/IEEE (12/2010). "ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary". In: *ISO/IEC/IEEE 24765:2010(E)*. Accessible at: https://www.smaele.nl/documents/iso/ISO-24765-2010.pdf, pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835. URL: https://ieeexplore.ieee.org/document/5733835, visited on 09/03/2019 (cit. on p. 7).

Jacobson, I., Spence, I., & Bittner, K. (12/2011). *Use-Case 2.0: The Guide to Succeeding with Use Cases*. Tech. rep. URL: https://www.ivarjacobson.com/sites/default/files/field_iji_file/article/use-case_2_0_jan11.pdf (cit. on p. 7).

Jailall, R. (04/2018). *It doesn't work: Story points, planning poker, software prediction*. URL: https://medium.com/@imprisonevery1/it-doesnt-work-story-points-planning-poker-software-prediction-2bfaefaf59ea, visited on 09/03/2019 (cit. on p. 7).

Kerievsky, J. (09/2012). *Stop Using Story Points*. URL: https://www.industriallogic.com/blog/stop-using-story-points/, visited on 09/04/2019 (cit. on p. 7).

Khosrow-Pour, M. (2006). *Dictionary of Information Science and Technology*. Dictionary of Information Science and Technology Bd. 1. Idea Group Reference. ISBN: 9781599043869. URL: https://books.google.de/books?id=KVQB9Mhx6d8C (cit. on p. 3).

Knublauch, H. & Kontokostas, D. (07/2015). *Shapes Constraint Language (SHACL)*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/shacl/ (cit. on p. 4).

Krimmer, D. (09/2017). *Why We Kicked Estimation Meetings (And Maybe You Should Too)*. URL: https://www.dkrimmer.de/2017/09/04/why-we-kicked-our-estimation-meetings/, visited on 09/04/2019 (cit. on p. 7).

Lohmann, S., Negru, S., Haag, F., & Ertl, T. (2016). "Visualizing Ontologies with VOWL". In: *Semantic Web* 7.4, pp. 399–419. DOI: 10.3233/SW-150200. URL: http://www.semantic-web-journal.net/content/visualizing-ontologies-vowl-0, visited on 09/30/2019 (cit. on p. 28).

McGuinness, D. L. & Harmelen, F. van (07/2004). *OWL Web Ontology Language (OWL)*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/owl2-overview/ (cit. on p. 4).

Miller, E. & Schloss, B. (10/1997). *Resource Description Framework (RDF)*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/rdf11-concepts/ (cit. on p. 3).

Miller, L. & Brickley, D. (01/2014). *FOAF Vocabulary Specification (0.99)*. Namespace. URL: http://xmlns.com/foaf/spec/ (cit. on pp. 4, 8 sq.).

Pastor-Sanchez, J.-A. (2015). *UNESKOS Vocabulary*. Namespace. URL: http://skos.um.es/TR/uneskos/ (cit. on p. 11).

Powers, S. (2003). *Practical RDF: Solving Problems with the Resource Description Framework*. O'Reilly Media. ISBN: 9780596550516. URL: https://books.google.de/books?id=VfcX9wJEH3YC (cit. on p. 3).

Ramachandran, R. (2017). "react-trello". In: *GitHub*. Initial Commit: https://github.com/rcdexta/react-trello/tree/ab72f5fc9b7f253ee8b22aace169c33a0efe5bc1. GitHub Repository. URL: https://github.com/rcdexta/react-trello (cit. on p. 35).

Shaposhnik, R., Martella, C., & Logothetis, D. (2015). *Practical Graph Analytics with Apache Giraph*. Apress. ISBN: 9781484212516. URL: https://books.google.de/books?id=FWb%5C_CgAAQBAJ (cit. on p. 4).

Stanley, M., Moussa, M., Paolini, B., Lyday, R., Burdette, J., & Laurienti, P. (2013). "Defining nodes in complex brain networks". In: *Frontiers in Computational Neuroscience* 7, p. 169. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00169. URL: https://www.frontiersin.org/article/10.3389/fncom.2013.00169, visited on 09/15/2019 (cit. on p. 3).

Stoica, M., Ghilic-Micu, B., Mircea, M., & USCATU, C. (12/2016). "Analyzing Agile Development – from Waterfall Style to Scrumban". In: *Informatica Economica* 20, pp. 5–14. DOI: 10.12948/issn14531305/20.4.2016.01. URL: http://www.revistaie.ase.ro/content/80/01%20-%20Stoica,%20Ghilic,%20Mircea,%20Uscatu.pdf, visited on 09/30/2019 (cit. on p. 6).

Tong, Q., Zhang, F., & Cheng, J. (2015). "Construction of RDF(S) from UML class diagrams". In: *Journal of Computing and Information Technology* 22, p. 237. DOI: 10.2498/cit.1002459. URL: https://www.semanticscholar.org/paper/Construction-of-RDF(S)-from-UML-Class-Diagrams-Tong-Zhang/fd18ef8838a8ca8ee57c6a806ad00479d615f717 (cit. on pp. 32 sq.).

UNESCO (1977). *UNESCO Thesaurus*. Thesaurus. UNESCO. URL: http://vocabularies.unesco.org/browser/thesaurus/en/ (cit. on p. 11).

W3C (2015). *WAI-ARIA Authoring Practices 1.1*. W3C Working Group Note 14 August 2019. W3C. URL: https://www.w3.org/TR/wai-aria-practices/#dialog_modal (cit. on p. 39).

W3C (n.d.). *Ontologies / Vocabularies*. URL: https://www.w3.org/standards/semanticweb/ontology.html, visited on 09/04/2019 (cit. on p. 3).

Wilder-James, E. (03/2004). *DOAP: Description Of A Project*. URL: http://usefulinc.com/ns/doap%5C#, visited on 10/02/2019 (cit. on p. 14).

❧ Colophon

This thesis is set in LaTeX 2_ε (*Leslie Lamport*, 1984) and was written using the web service *Overleaf*. Chapter, section, and figure titles are set in Helvetica (*Max Miedinger*, 1960). The body text is set in Linux Libertine (*Philipp H. Poll*, 2003). The typeface used for code sections is Bera Mono (*Walter Schmidt*, 2004). Figures and graphics were designed with *Illustrator* (*Adobe*, 1987) and *Mathcha.io* (*Bui Duc Nha & Phan Thi Minh Nhat*, 2019). Flow charts were sketched using the web service *draw.io*. JavaScript and React was written within *WebStorm* (*JetBrains*, 2010) and *VS Code* (*Microsoft*, 2015).

Acknowledgements

Foremost, I would like to thank my supervisors: Dr. Michael Martin and Dr. Sebastian Tramp. Michael, thank you for introducing me to eccenca GmbH and thus giving me the opportunity to apply my theoretical knowledge gained throughout my studies in practical field. A very special thank goes to Seebi for great support, discussions about the prototype, and overall for introducing me into the field of semantic web, and giving me to opportunity to learn about modern front-end development and workflows in a great enterprise housing even more sincere people! I'm also grateful for your great patience with me during the last year. Thanks to Jan Kaßel for helpful comments. Maleen Thiele! A special thanks for your outstanding support in various aspects, for you great language support, and for your great patience with me during the last months. Last but not least I want to thank my parents for their endless and unconditional support throughout my studies—it's over now.

Statement of Authorship

From the examination regulations:[54]

> Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.[55]

Leipzig, December 20, 2019 _____

Place, Date Signature of Steven Kalinke

Digital Version

The digital version of this work is supplied on the SD card below. The CRC-64 checksum of this file must match my following handwritten one:

CRC-64 _____

The zip file contains the PDF version of this work. In contrast to the print version, the digital version indicates all clickable references in blue color (e.g., literature references). It is otherwise identical to the printed version.

[54] http://studium.fmi.uni-leipzig.de/fileadmin/Studienbuero/documents/Formulare/HinweiseAbschlussarbeit.pdf

[55] Im Gegensatz zur Printversion zeigt die Digitalversion alle klickbaren Referenzen in blauer Schrift an (z. B. Literaturverweise). Ansonsten ist sie identisch mit der gedruckten Version.