Bachelor's thesis

Advanced Enterprise Issue Tracker

Zakir Abdalimov

Department of Computer Science
Supervisor: Ing. Petr Křemen, Ph.D.

May 24, 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Abdalimov** Jméno: **Zakir** Osobní číslo: **456946**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Pokročilý Firemní Issue Tracker

Název bakalářské práce anglicky:

Advanced Enterprise Issue Tracker

Pokyny pro vypracování:

1.Udělejte rešerši existujících systémů pro issue tracking, např. Redmine, JIRA, apod.
2.Vytvořte doménově nezávislý obecný model popisující společné rysy analyzovaných systémů.
3.Na základě vytvořeného modelu formulujte funkční a nefunkční požadavky na firemní issue tracker umožňující :
a)vytvářet a klasifikovat issues pomocí dynamických slovníků (RDF,SKOS)
b)vytvářet a klasifikovat řešení těchto issues pomocí dynamických slovníků (RDF,SKOS)
4.V jazyce UML navrhněte jednoduchý systém pro pokročilý issue tracking dle bodů 2 a 3.
5.Systém implementujte jako webovou aplikaci.
6.Demonstrujte jeho použitelnost na ukázkovém scénáři pro software issue tracking a porovnejte jeho vlastnosti proti systémům v bodě 1. Systém vhodně otestujte.

Seznam doporučené literatury:

[1] Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2013. Situational awareness: personalizing issue tracking systems. In Proceedings of the 2013 International Conference on Software Engineering (ICSE '13). IEEE Press, Piscataway, NJ, USA, 1185-1188.
[2] https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems
[3] http://www.uml.org
[4] https://www.w3.org/2004/02/skos

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Křemen, Ph.D., Skupina znalostních softwarových systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Petr Křemen, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank my supervisor Ing. Petr Křemen, Ph.D. who guided me throughout this work.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on May 24, 2018 .....

Czech Technical University in Prague
Faculty of Electrical Engineering

Citation of this thesis

Abdalimov, Zakir. *Advanced Enterprise Issue Tracker.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Electrical Engineering, 2018.

Abstrakt

Současný trh systémů pro sledování řešení poskytuje širokou nabídku aplikací, které se liší mezi sebou svým zaměřením a účelem. Cílem této práce je navrhnout a naimplementovat takový systém, který by převzal charakteristiky a vlastnosti existujících aplikací pro sledování řešení a zároveň by poskytoval užitečnou funkcionalitu, která v těchto aplikacích chybí. Toto zahrnuje sledování řešení a kategorizace problém a jejích řešení pomocí RDF slovníků.

Klíčová slova sledování problémů, sledování řešení, kategorizace problémů, kategorizace řešení, RDF.

Abstract

The current market of the issue tracking systems offers a wide range of choices which vary from each other regarding their focus and usage. The aim of this work is to design and build such system which inherits characteristics and design choices from existing issue tracking applications and at the same time offers new useful features that they are lack of. The latter include solution tracking and categorization of issues and their solutions using RDF dictionaries.

Keywords issue tracking, solution tracking, issue categorization, solution categorization, RDF.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goal

The goal of this work is to build a system that provides issue and solution tracking and categorization for the enterprise. It is highly inspired by already existing issue tracking systems but has its own unique features and characteristics as well such as supporting solution tracking and advanced issue and solution categorization.

1.2 Outputs

The outputs of this work are:

1. Analysis of existing issue tracking systems.
2. Creation of generic domain model for analyzed systems.
3. Definition of the requirements according to the created domain model for the new system which allows:
   a) Creation and categorization of issues using dynamic dictionaries such as RDF and SKOS.
   b) Creation and categorization of solutions for the issues using dynamic dictionaries such as RDF and SKOS.
4. Design of the new system in UML according to point 2 and 3.
5. Implementation of such system in form of web application.
6. Demonstration and evaluation of usability of the implemented system by trying several use cases out on it and comparing it to systems described in point 1. The system must be properly tested.

1.3 Issue Tracking

Issue tracking is a process of collecting, assigning, resolving and archiving problems. It helps teams, projects, companies and enterprises achieve more effective and consistent approach to solving emerged issues. Originally, issue tracking was a term that is tightly coupled with software projects, especially ones that involves bug tracking [1]. Currently, issue tracking is incorporated in other fields as well such as marketing, finance, customer service, management and others.

1.4 Motivation

1.4.1 Current Situation

Currently market of software systems provides a solid range of choices for issue tracking. They vary from simple task management applications to complex systems that are oriented on massive enterprises. Some of them are strictly adjusted for a particular use and others can be customized and configured to be applicable in any domain.

The problem with existing issue tracking systems that they are missing solution tracking feature and evaluation despite its usefulness (see section 1.4.2.2).

1.4.2 Suggested solution

The suggested system provides enterprise issue and solution tracking and categorization that is usable for any arbitrary domain.

1.4.2.1 Issue Tracking

The motivation for issue tracking is straightforward - it is used for:

1. delimiting responsibility between project members for completing tasks,
2. tracking state of issues,
3. keeping task history for future revision and use

The usefulness of mentioned points can be observed by looking at vast usage and supply of issue tracking systems on the market.

The motivation for applicability of this application in an arbitrary domain is justified by following means: each organization has different requirements and specifics for issue tracking - software companies have a necessity of tracking bugs, airlines must track incidents on the runway, etc. Each of those issues is defined differently: bug should contain information about the version of the system where it was found, an incident report should be connected with the identification number of the runway where it took place. So every organization will be able to provide their own definitions (types, custom fields and statuses) of issues without the need of changing application itself. Yet the nature of issues is similar across different systems. Issues tend to report situations that need to be handled (bugs resolved, tasks fulfilled, etc.) thus making it possible to unify the approach of their tracking despite their differences.

The categorization of the issues is usable for a couple of reasons. Firstly it adds more meaning to the issue. Depending on how an issue is categorized users are given information about the purpose of the issue, its possible contents and circumstances where or when it

took place. Secondly, it improves its searchability in a long run because it is an additional characteristic to be considered for search. The same applies to the solution categorization as well.

1.4.2.2 Solution Tracking

Solution tracking is a similar concept to issue tracking except that tracking matter is solutions of the issues. Solutions can be categorized and managed in the same fashion as the issues. To understand usefulness consider the following example in the context of a common issue tracking system.

When some type of issue is getting solved as a rule the solution is written as a comment on the issue. Then after some time passes by another issue emerges which has different contents but can be resolved by applying the same solution. The problem is that this particular solution would be difficult if not impossible to find in case of a massive enterprise which can have thousands of issues tracked at the moment. The search for this or any other solution would be problematic for several reasons:

- Solutions are indistinguishable from comments.
- Solutions are firmly coupled with its issues, thus in order to find a particular solution corresponding issue must be found first.
- Solutions are not tracked separately, therefore they are not categorized and cannot be filtered in a straightforward matter.

The usability of such functionality can be observed on a question and answer website Stack Overflow which is widely known in software development community [2]. When somebody is asking a question on Stack Overflow it is a common practice that the answer to this question is a reference to another answer which was given in a different question.

If the question is a duplicate of another question then the relevant answer can be easily found and referenced but often there are times when questions are different and the probability of finding the relevant solution and referencing it, in this case, is very low. Therefore tracking answers or in case of this application solutions separately from issues is beneficial - it makes re-usage of those solutions easier.

1.4.2.3 Example

Consider following example. Two separate teams are performing integration and manual testing on the system X inside the financial enterprise. The manual testing team faced a problem with displaying information about saving account of type “Saving+”. They submit a corresponding defect report to the issue tracking system. Then one of the colleagues from deployment team mentions in the comment section that several services of the system X are not available right now because they are struggling with certain error and are not able to deploy them. He then mentions when approximately the services would available so the manual testing team can adjust their test schedule accordingly thus solving the issue.

Meanwhile, integration testing team is struggling with other problem - one of the REST services returns error status when sending data to the URL with address *“/invoices”*. The error is caused by the same undeployed services that the first team is faced with but the integration testing team is unaware of that because the first team mentions a different

problem in their defect report and it cannot be spotted as a duplicate issue. By not spotting the first teams defect report second team certainly would not find the desirable solution mentioned by the colleague from the deployment team. Thus the integration testing team submit their own report to the issue tracking system and receives the same comment about the deployment issues.

Conclusively there are two answers instead of one and wasted time on waiting for the answer in case of the second team. Moreover, the answers mentioned in the comments are not particularly reusable because they are difficult to find.[1]

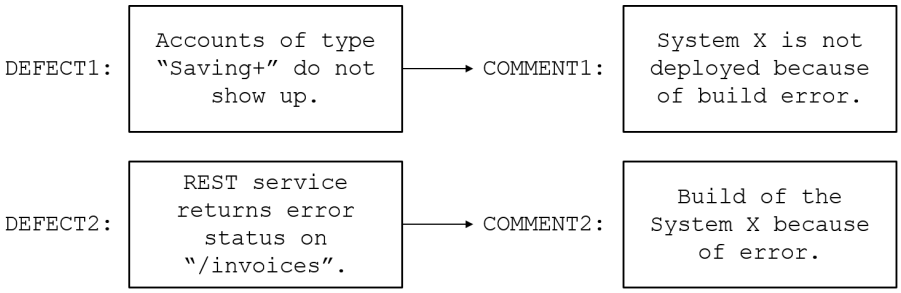

Figure 1.1: Defects and their answers (comments) emerged from example above

[1] More detailed possible solutions to this situation are discussed in section 6.1

Chapter 2

Background

This chapter introduces the information about the context of this project including technologies, methods and principles used.

2.1 RDF

The Resource Description Framework (RDF) [3] is a model to represent data on the internet. The data in RDF is represented by so-called *triples* that consist of three parts: *subject*, *predicate* and *object*. Subject and object are nodes that represent some kind of resource while *predicate* represents a relationship between them. By aggregating triples, such formation of data can be viewed as a graph structure. Each node (resource) and edge (relationship) in RDF graph is uniquely labelled by a uniform resource identifier (URI) to clearly identify each resource.

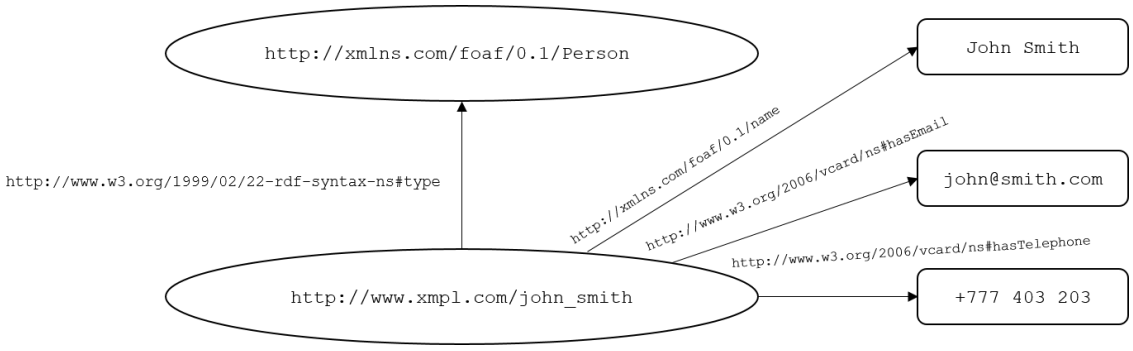

Figure 2.1: RDF graph describing John Smith

Figure 2.1 is an example of simple RDF graph. It represents a person named John Smith that has an email address *"john@smith.com"* and telephone number *"+777 403 203"*. Predicates and resources of this graph are represented by URIs. There are also so-called literals that determine certain values of the properties such as name, email and telephone. They could also be defined by URIs but it is more readable to use text instead.

2.2 REST

Representational State Transfer (REST) [4] is an architectural style for web services which is based on the HTTP protocol. It is used to provide uniform interface thus enabling clients to interact with the web application in a more standardized way than using arbitrary and application-specific approach. The interactions within RESTful web services are considered stateless meaning that the requests and responses are not dependent on previous messages interchanged [5][6]. REST services are final endpoints for communication with the implemented web application.

2.3 Technology Stack

Principal technologies used in the system developed as a part of this thesis:

1. Spring Boot [7], as core of the application.
2. Maven [8], as build and package manager.
3. JPA [9], as object-relational mapping tool and persistence framework for relational database.
4. JOPA [10], as object-relational mapping tool and persistence framework for RDF database.
5. SPARQL [11], as language to query the RDF store.
6. Spring Data [12], as data access provider.
7. Hibernate framework [13], as persistence provider.
8. Spring Data REST [14], as REST API provider.
9. PostgreSQL [15], as relational database system.
10. Spring Security [16], as security module.
11. JUnit [17], as testing framework.
12. Mockito [18], as mocking framework.
13. JMeter [19], as performance testing tool.

2.3.1 Spring Boot

The whole lifecycle of the application is managed by Spring Boot. It provides same features as vanilla Spring Framework such as:

- application context,
- dependency injection,
- aspect-oriented programming interface,
- transaction management,

- exception handling,
- tools for REST controllers definition,
- integration with contemporary Java frameworks and libraries.

The philosophy of Spring Boot is to exempt its users from as many configurations as possible by having default ones. At the same time, it offers flexibility by providing a possibility of rewriting those configurations that are applied by default. Another principal point of Spring Boot applications is ease of starting which is achieved by using embedded web server thus eliminating the need for deployment of WAR files.

2.3.2 Maven

Maven is build and package management tool that is widely used is Java community. It allows configuration of the build cycle which include testing, validation, compilation and deployment of the application. It is also responsible for dependency management of the application.

2.3.3 JPA

JPA is used as an object-relational mapping tool. It maintains consistency between Java objects and relational database. It also used by Spring Data module which defines appropriate JPA repository for each entity.

2.3.4 Spring Data

Spring Data provides an interface to access data layer with a minimum amount of effort. It uses JPA repositories that allow executing CRUD operations for defined JPA entities with an addition of sorting and pagination. The main advantage of the Spring Data JPA repositories that it is defined by interfaces and there is no need to write concrete implementations for custom functions. Their behaviour is either defined by JPQL-like queries or by the name of the functions themselves.

2.3.5 Hibernate framework

Hibernate framework is implicitly used by Spring Data as the persistence provider to perform operations on the database. It also offers an extension to standard JPA annotations.

2.3.6 Spring Data Rest

Spring Data Rest is a logical continuation of Spring Data module. It exposes Spring Data JPA repositories by providing auto-generated REST controllers for basic CRUD operations which get rid of many lines of boilerplate code in controller layer of the application. For every defined method in the repository, there is automatically mapped controller method in REST API.

2.3.7 SPARQL

SPARQL is a SQL-like language used to perform queries on RDF stores and remote SPARQL endpoints. In the context of this application, it is used to create native queries using JOPA entity to retrieve RDF data as plain Java objects.

2.3.8 JOPA

JOPA is a relational mapping tool and persistent framework. It used in several ways in the application:

1. to query RDF triple store by executing basic CRUD as well as custom operations using SPARQL language,

2. to maintain consistency between Java objects and records within the RDF triple store,

3. to serialize objects to JSON-LD format which is used when sending RDF data from REST controllers.

2.3.9 PostgreSQL

PostgreSQL is a relational database which offers all features that object-relational database is expected to offer. Those include views, procedures, triggers, constraints, indexes and more others. The main reason for choosing PostgreSQL is that it is open-source and easily integrable in Java applications because of an extensive amount of frameworks that are compatible with it.

2.3.10 Spring Security

Security of the application is managed via Spring Security module. It serves several purposes:

- Manages authorization, authentication and user roles.
- Manages user session and cookies.
- Manages permissions for requesting certain resource (URL).
- Provides interface for CORS configuration.
- Provides interface for password hashing and salting.

2.3.11 JUnit

Testing is managed by JUnit framework which is integrated with Spring Boot. The purpose of this integration is to provide:

- application context during test execution,
- transaction management during integration tests,
- interface for configuration of the test environment.

When integration tests take place embedded H2 database is used for faster test execution and separation from development and production database.

2.3.12 Mockito

Mockito is a mock framework used for unit testing. It allows the mocking behaviour of the objects in order to get rid of the necessity of their instantiation which can be troublesome and complex at times. Another benefit of using mock framework is a possibility of testing certain layer of the application isolatedly without the need of actually using the logic of the adjacent layers. It is integrated with Spring Boot framework which allows mocking of beans and components without any further configuration.

2.3.13 JMeter

JMeter is a tool which allows execution of performance tests on web services. In this case, it is used to generate HTTP requests that are sent to the REST services in order to test the responsiveness and reliability of the application in heavy load conditions. Each performance test case is defined by a script with HTTP requests and number of their executions specified within them. Scripts are created by the separate JMeter application that is accessed by a normal graphic user interface. Then scripts are imported into the project itself and executed by JMeter maven plugin.

Chapter 3

Analysis

3.1 Existing Solutions

This chapter describes existing issue tracking systems and their functionality regarding issue tracking, categorization of the issues, project structuring, permission management and solution tracking. Other aspects of the systems mentioned here that are not the focus of this work would be skipped.

Each system described here is used differently and has its own purpose. The chosen candidates were selected in such way that there is one of every kind thus making a set of systems that have a different focus. The main criteria for choosing a candidate within the group of systems that have similar focus is widespreadness of use.

In this chapter there are the general description, the summary of provided functionalities and conclusions stated for each of the systems. More detailed description that did not make to this chapter for each of the described applications can be found in section A.

3.1.1 Jira

Jira[2] is an issue tracking and project management tool that offers a complete solution for both enterprise scale projects as well as small teams. It is highly configurable which allows aligning its behaviour and contents to fulfil specific requirements for the vast range of domains. Jiras configurations include but not limited to:

- Issue types that determine which custom fields can be the part of the issue.
- Issue statuses that are dependent on chosen issue type.
- Workflows that restrict issue status transition depending on current issue status or user role.
- Issue priorities that have certain order depending on importance.
- Custom fields with a substantial amount of basic and advanced field types to choose from. They also can be configured to change behaviour depending on other custom field or current issue status (for example validation rules).
- Issue links to express a relationship between issues.

[2]The business version of Jira which is not specifically oriented on software projects is called Jira Core but for the convenience, it would be referred as Jira in this document.

Configuration options mentioned above are shared across the application but in order to use them in a specific project, it must be explicitly stated. It is done through so-called *screen* which is a set of configurations that can be applied to a project. This way Jira gets rid of defining same configurations per projects but at the same time set restrictions to the global configurations applied to projects to make it usage effective and more clear. All the configurations are managed by the single global administrator of the application which is defined at the beginning of the first deployment of the application on the server.

In order to spare users time on configuration, Jira offers a variety of templates when creating a project which includes set of predefined configurations for common project types.

Issue categorization in Jira is realized via issue types.

Each project in Jira can be divided into so-called *modules* which can be considered a subset of issues of the parent project. It is used for additional delimitation of responsibilities in the project.

Regarding permission management in a project, Jira offers user roles and groups. They define what action user within certain group or role can do. Permissions go beyond standard checkboxes and can be configured to be depending on for example state of the issue.

Overall Jira is a complete solution that can satisfy demands of a wide variety of projects because of its flexibility. On the other hand, such configuration depth comes with the complexity of its use. In order to simply add a custom field to an issue type, there are several layers of configuration abstraction that need to be passed. As well as other issue tracking systems Jira does not provide solution tracking.

3.1.2 Redmine

Redmine is an open source project management tool and an issue tracking system [20]. Same as Jira it can be used both for software and business projects but it is more lightweight regarding configurations and is not designed to cover every possible type of project.

Regarding issue tracking Redmine offers following configurations:

- Issue types that define its custom fields.
- Workflows that can be based on user role or current issues status.
- Issue statuses depending on issue types.
- Custom fields with several basic field types to choose from. Additional validation can be set to them by using a regular expression.

Configurations mentioned above are not shared across the application and need to be defined in each project separately. There is a possibility to copy a whole project with its configurations which can save time in some cases.

Redmine does provide a categorization of issues by issue types.

The project structure in Redmine is represented by the tree of projects and sub-projects. Each project can have divided into sub-projects. This structure has no restriction on depth.

Permission management in Redmine is addressed by user roles which every user has only one of. Basically, every role has set of permitted operations that user can perform on

an issue. It not as flexible but quite straightforward to use because it requires only one step to pass.

To conclude Redmine is more generic regarding configurations in comparison for example with Jira. It favours universal behaviour rather than richness and flexibility of configurations. Thus it is easier to use but can be restricting regarding configurations. It also does not provide solution tracking.

3.1.3 YouTrack

YouTrack is an issue tracking and project management which is specifically designed for developers [21]. It offers the creation of custom:

- Issue types which define custom fields for issues.
- Fields with a decent amount of basic and enumerated data types to choose from. There is also the possibility to make fields conditional meaning that they can change their behaviour depending on other fields (e.g. visibility, validation).
- Workflows that are defined by scripts which expands their configurability to another level which is not yet introduced in other issue tracking systems.
- Statuses depending on the issue type.
- Issue links that represent a certain relation between given and another issue.

Mentioned configurations are shared across the projects but in order to use them, they must be explicitly applied to a project. It is done by separately adding each of the desired configurations to a project.

Regarding categorization of the issues, YouTrack offers tags to label issues which are used to find those issues more easily. Moreover, there is a possibility of defining enumerated data types for custom fields which also can be used to categorize the issues. The categorization is also held by issue types.

Projects in YouTrack cannot be divided or structured in any kind of hierarchy.

YouTrack offers flexible permission management by using user groups and roles. A user role is a set of permissions that are applied to a user group or particular user. User groups are set of user roles that can be applied to a user.

In conclusion, YouTrack is a powerful issue tracking system for software developers. It can be theoretically used in non-software projects but most of the time it would be an overkill. It does not provide solution tracking.

3.1.4 Basecamp

Basecamp is a web-based application which is used for project management. Its design is focused on communication and task management within the team.

It offers a tool for managing so-called *todos* - tasks that need to be done in the project. Even though it is not an issue tracking system, the process of defining, assigning and completing tasks can be considered as a form of issue tracking.

In Basecamp todos are statically defined, meaning that users can not add custom fields to them nor change the behaviour of default ones. Each todo can have an assignee - the person who is responsible for the task, list of people in the current project that would be notified when the task is done, note and due date.

Speaking of categorization, todos are divided into groups called boards. Each of those is shown in project tasks section, so participants can find the appropriate tasks more easily. When all tasks are done on the board it can be archived, so it would not be shown any more on the page. Such an approach to a categorization of the tasks serves rather visual and temporary orienting purpose than the goal of giving a task more meaning and improving searchability in a long term.

Basecamp has one-level project structure meaning that is not possible to divide projects into smaller parts in any way.

Regarding permission, management Basecamp offers several user roles that are statically defined and can not be changed in any way. Those include client user which has permissions to read and comment on todos and other submissions, non-client users which are able to create projects, delete and update their submissions and lastly administrators that on top of mentioned privileges can manage project participants and any submissions.

To conclude - Basecamp is a simple tool which is straightforward to use. It is general enough to be usable for a wide range of organizations. On the other hand, it is not configurable, thus missing domains that require more specific and advanced issue tracking.

3.1.5 Summary

	Jira	**Redmine**	**YouTrack**	**Basecamp**
1. Domain	Arbitrary	Arbitrary	Software development	Limited arbitrary[3]
2. Permissions	User roles, user groups	User roles	User roles, user groups	User roles[4]
3. Configurations	Issue types, issue statuses, custom fields, workflows, issue priorities, issue links	Issue types, issue statuses, custom fields, workflows	Issue types, issue statuses, custom fields, workflows, issue links	Absent
4. Categorization	Issue types	Issue types	Issue types, tags	Boards
5. Solution tracking	Absent			

Table 3.1: Summary of the analysed systems

The above-given table summarizes the characteristics of the analysed systems described in details in previous sections. Each of the above-mentioned parameters has following meaning:

1. In which domains is the system applicable.
2. By what means permission management of users is realized.
3. By what means the configuration of issues is achieved.
4. By what means categorization of issues is implemented.
5. If the solution tracking is present.

Each of the analysed systems differs in their focus, applicability and characteristics. The common trait that they have is the absence of the solution tracking feature. Design choices, practices and features of the mentioned applications are used as the basis for the suggested system which is discussed in section 4.3.

[3]It can be applied in a range of different domains but can not satisfy those that require more specific issue tracking.

[4]Unlike in other analysed systems user roles are statically defined and can not be changed in any way.

Chapter 4

Design

4.1 Business Requirements

This section describes the specification for the developed application. It is divided into two sections. First one describes functional requirements which define what features it is able to provide to the user. In the second one non-functional requirements are defined which specify technical details and operation of the system itself rather than its functionality.

4.1.1 Functional Requirements

4.1.1.1 Project Management

The application provides management of the projects which is performed by the administrator. This includes creating, updating and deleting projects. Each project is a collection of participants and their submissions which includes issues, solutions and comments. The administrator is able to add and remove participants from the projects. The administrator is considered global throughout the application and is only one user of a kind.

4.1.1.2 Issue Tracking

This application will collect issues submitted by participants of projects. Issues can be commented on and have proposed solutions to them which is done by the project participants.

Issues are not statically defined and can contain a variety of different information depending on the given business model of an organization which means that they can be configured. This includes the definition of issue types, field types and statuses.

Configurations can be created locally using services of the suggested application as well as imported from remote endpoints reusing already defined issue types, field types and statuses (described in more details in 4.1.2.1).

Each issue must have a type which defines its purpose. Depending on the chosen issue type issue acquires custom fields which are the part of the issue that represents a particular piece of information in the given issue. Moreover depending on the issue type issue has a certain set of statuses to be in.

Issues are then categorized in order to add more meaning to them and improve their searchability in long run. Categorization is held by assigning categories and issue types to the issues. Categories work as labels or keywords that issues can be searched with. Their usage differs from assigning an issue type. Issue types define what kind of task,

problem or question it is and what its contents are whereas category gives information about circumstances of the issue. Consider following example in a context of the software project. There is an issue with type *status meeting* which defines this issue as "meeting where project lead informs the management about the state of the project". Because the status meeting is taking place during *summer release of 2018* and is connected with a requirement *user migration to CRM 2.0* it can be labelled with corresponding categories.

Categories, issue types, custom fields and statuses are added and updated by the administrator. Mentioned configurations are shared across the application and can be used in any created project without explicit permissions. The design of the configuration process is focused on the ease of use rather than complexity and flexibility of offered functionality.

An issue can be searched by its type, current status, creation/modification date, author, any of the defined attributes or categories that are assigned to the issue. All of the mentioned parameters can be combined with each other in order to create more specific queries if needed.

4.1.1.3 Solution Tracking

Each participant of the project can propose solutions to issues created within the given project. Reported solutions will be tracked, categorized and could be commented on. When proposing solution participant can offer their own newly created solution or reuse existing one. If the participant wants to propose an already existing solution it must be found first. The search can be done by title, description, the original author, creation date, rating or categories of the solution.

After solutions were proposed author of the issue can choose which one of them actually solved the issue and mark it as *the solution* to the issue.

Solutions can be categorized in the same fashion as issues are. The categories for solutions and issues are shared because often issues are categorized by the same or similar set of the categories as solutions are.

Solutions can be rated by the project participants thus showing their relevance, popularity and effectiveness. Rating associated with the solution is merely a number that shows how many participants gave "thumbs up" to the solution.

4.1.2 Non-functional Requirements

4.1.2.1 Usage of RDF

The application must be able to operate with RDF data in order to provide a categorization of the issues and solution by dynamic dictionaries. There are following advantages to this.

Firstly RDF is not only used for storing data but also defining the meaning of the data in an unambiguous way. This is realized via assigning URI to each of the defined concepts. Each of the concepts is then can be exposed online to be accessible globally. This way the meaning of the data is defined in an exact and clear matter which reduces the probability of its inconsistent usage.

Secondly, *"RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed."* [22]. This opens the possibility to integrate and reuse other data models and dictionaries to define issue tracking models for different organizations.

And finally, RDF data can be easily exported in a consistent matter to be migrated to and reused in other applications.

These particular characteristics are taken advantage of to import configurations from remote SPARQL endpoints that provide openly accessible RDF data on demand. There is a variety of those available on the internet [23] and they can be accessed via URL. Even though imported configurations can be defined by concepts that differ with local ones RDF allows merging of this models. Consider following example. All the issue types in local repository are defined by concept *"http://example.com/issueType"*, whereas issue types to be imported from the remote repository are defined by *"http://example.com/bugIssueType"*. Each imported instance of *"bugIssueType"* acquires property *"http://www.w3.org/2000/01/rdf-schema#subClassOf"* with value *"http://example.com/issueType"*. Mentioned property is commonly used to represent relationship between two concepts in this case meaning that *"bugIssueType"* is now considered an *"issueType"*. From now on the application is able to use imported issue types as if they were described by the local model.

4.1.2.2 Data Consistency

Data submitted through the application should be consistent across the whole system. It is the main non-functional requirement because the application model is based on two different data models: static which is represented relational SQL database and dynamic which is represented RDF store. It is crucial to focus on endorsing consistency between them because of their different nature and usage.

4.2 Characteristics and User Classes

The current section describes application design and offered functionality in more details. This includes the definition of the user roles, possible interactions with the application and entities used in the system.

4.2.1 Use Case Model

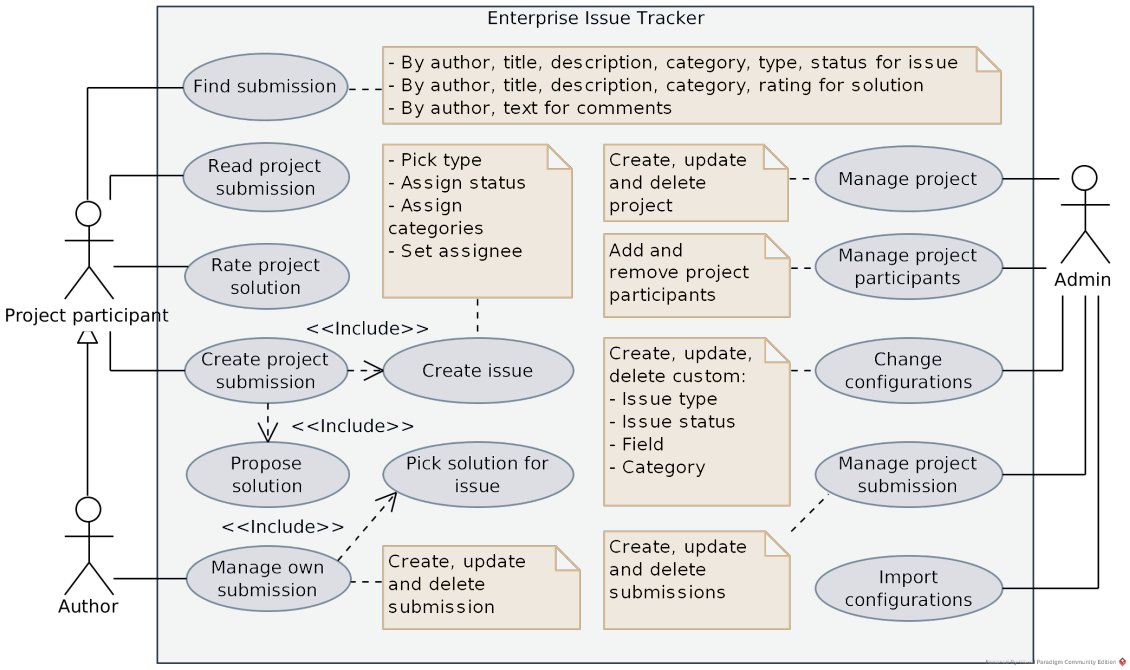

Figure 4.1: Use case model of the application

The above-given diagram shows possible user roles and their interactions with the given system. There are three roles defined in the system: *Admin* (or so-called administrator), *Project participant* and *Author*.

The project participant role is acquired by being invited to a project. Within the given project participant can find, read and create submissions which includes creating issues, proposing solutions and adding comments. Each participant of the project can rate solutions thus expressing an opinion about its effectiveness and relevance.

When project participant creates some kind of submission he or she becomes the author of the created submission which gives a possibility to update and delete it. In case of an issue author also acquires right to pick a solution to it.

The purpose of admin role is to manage projects and their participants - he is able to create, update, delete projects, invite and remove participants from them. Admin is also the only user that can configure issues meaning that he can create, update and delete issue types, custom field types, statuses and categories. These configurations are mutual across the projects meaning that they can be used in any created project. Admin also has rights import configurations from remote endpoints in order to reuse existing issue types, field types and issue statuses defined there. Admin has the privilege to manage submissions in any projects same as if he or she was the author of every submission created. There is only one admin defined throughout the application instance.

4.2.2 Business Domain Model

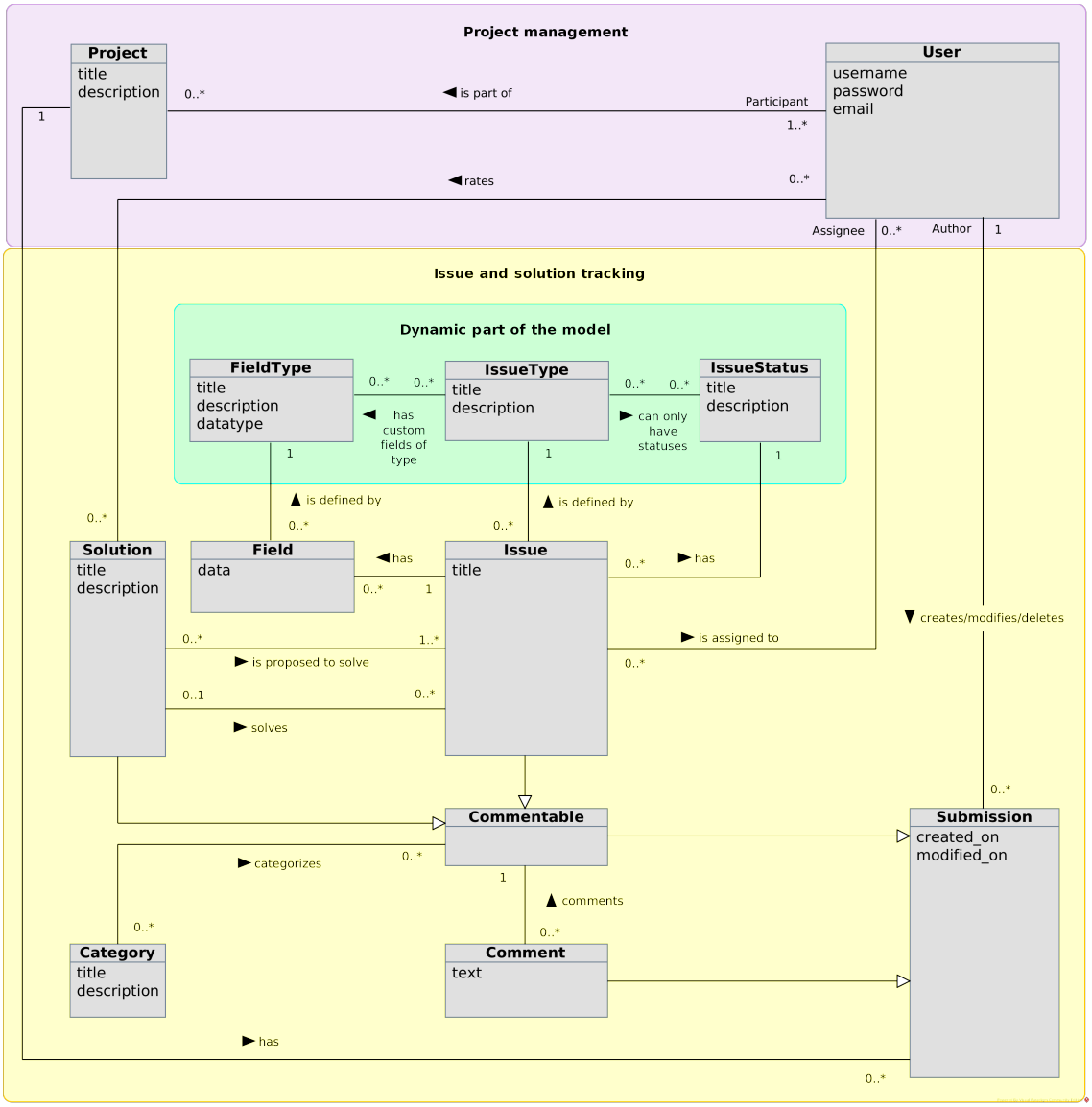

Figure 4.2: Business domain model of the application

The above-given diagram gives an overview of entities which are incorporated in the system. It can be divided into several sections.

The *Project management* section is representing the entities that are tightly coupled with project and participant management. It consists of *Project* and *User* entities. Projects can be defined as a set of users and their submissions that they interact with. All users in the application are participants of at least one project with exception of the administrator which is, in fact, creates projects thus can exist without participation in any project.

The *Issue and solution tracking* section consists of entities that responsible for the tracking of solutions and issues. Those core entities of this section are *Issue*, *Solution*, *Category* and *Comment*. Issues can have several proposed solutions to it but only one

that has actually solved the issue. Issues and solutions are categorized by the same set of the categories. There is no need to define those for issues and solutions separately because quite often they would be categorized by the same or similar list of categories. Issues and solutions can be commented on thus participants of the project can express their opinions. A solution can be rated by participants of the given group which is shown as a number of times it was rated or in more simple terms "liked".

Other entities of the *Issue and solution tracking* section, namely *Commentable* and *Submission*, are only used for generalization of the entities and reduction of duplicity in the model.

The dynamic section represents configurable part of the model which is closely coupled with the issue entity. Each issue must have issue type which defines custom fields types and statuses for the issue. Each field instance that is connected to the issue is defined by field type which gives a field title, description and data type. Because the dynamic part is stored as RDF all the simple data types that are defined in XML can be used in field type [24].

These configurations are mutual throughout the application meaning that they can be used in any project without any explicit permissions.

Each entity of the dynamic model is described by RDF dictionaries. This is held by assigning a unique URI to each property of the entity as well as to an entity itself. Each URI represents a concept in an RDF dictionary which is globally defined. The definition can be accessed online in order to see its exact description. This opens the possibility of re-usage already existing concepts for other organizations.

Let's take an example of the following instance of this model for better understanding of its applicability. Consider a small company which is oriented on software development. It has several departments: HR, finance, professional services and others. Each of the departments has their own respective project created. Each project contains relevant employees which participate in the operation of the respective department.

Projects also contain issues which are submitted by their participants. Each department has different responsibilities and specifics, therefore, dealing with different issues. HR department uses such issue types as *interview* and *team building*. Former contains information about interviewer, candidate, desired position, place and time while latter contains information about participants, place and time. Each particular piece of information is stored as a separate field within the issue and defined by appropriate field type. Once created field types, such as previously mentioned place and time, can be reused in several issue types.

Interview can acquire certain issue statuses such as pending when interviewer or candidate did not confirm their participation yet, confirmed when both sides are agreed on their participation with time and place arranged, *succeeded* when the candidate passed the interview and got desired position and *unsucceeded* when the candidate did not pass the interview.

The solution to the interview issue can be a reference from the interviewer based on which the issue is resolved.

Both interview issue and its solution can be commented on by other participants of the HR department. In this case, it is used by the organizer of the interview in order to communicate with interviewer regarding details of the interview.

Issue is categorized by categories *"summer recruitment campaign 2018"* that determines within which recruitment campaign interview is organized and *"professional services recruitment"* because candidate desired for position in corresponding department.

4.3 Reflection on Existing Solutions

The design of the application described in previous sections is highly inspired by practices applied in existing solutions.

The project management part is inherited from YouTrack and Basecamp. The project structure is only one-levelled as in YouTrack and Basecamp. The permission management is statically defined by user roles as in Basecamp. This approach is a most direct and require no configurations.

The configuration of the issue is based on the model of Jira, YouTrack and Redmine. It offers the definition of issue types, field types and statuses as in all of them. The global availability across all projects is taken from Jira and YouTrack but the complexity of the configuration management is more similar to Redmine. It gives the possibility of configuring issues for any domain but at the same time, the approach is more simple and straightforward.

Issue categorization is inspired by YouTrack which uses tags as well as issue types to categorize issues. It gives a more advanced approach for categorization in comparison with other systems because it takes into account both contents of the issue by its type as well as its circumstances by tags. In the current model, tags are defined as categories.

Unlike other systems suggested model offers solution tracking and categorization separately from issues and their comments which enables straightforward search and re-usage of the solutions.

Chapter 5

Implementation

This chapter describes the actual implementation of the system including its architecture, interactions between inner layers and concrete usage of the technologies described in section 2.3.

Note: the main goal of this work is to build back-end of the application according to suggested design. The front-end part of the application is used only for presentation purposes and is not focus of this work, therefore, is not described here.

5.1 Application Architecture

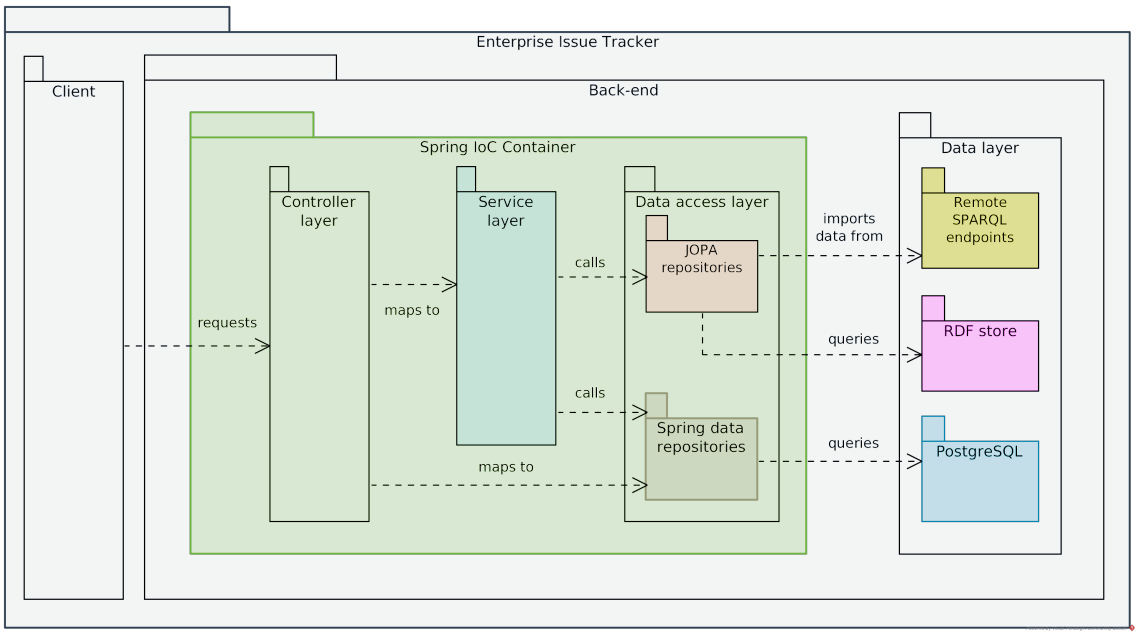

Figure 5.1: Application architecture diagram

This diagram showcases the different components and modules within the system and how they interact with each other. The structure of the application is divided into several layers that communicate with each other, basically following the classic N-tier architecture of enterprise applications.

5.1.1 Controller Layer

The controller layer mainly consists of Spring Data REST controllers which are provided automatically when Spring Data JPA repositories are defined. It gets rid of boiler code associated with implementing REST API for basic CRUD operations. Spring Data Rest also automatically provides an API for custom queries and methods defined in the repositories.

For more complex requests that require actual business logic to take place, it is necessary to define standard Spring Rest controllers as well. Those have services injected into them in order to execute business logic of the application.

REST controllers communicate with front-end in form of JSON. There are certain endpoints that return data in JSON-LD because of its RDF origin - for example for issue and field types. JSON serialization is managed by Jackson object mapper.

When a controller receives an entity as request body its gets validated by JSR validation mechanism described in section 5.1.3.

Exception handling is configured by *controller advice* component which defines responses and status codes to be returned as a reaction to certain types of exceptions. Thus there is no need to define this logic separately in each controller.

The controller layer respects the "Hypermedia as the Engine of Application State" (HATEOAS) [25] constraint of the REST application architecture. This allows clients to interact with REST interface without the need of knowing its structure in advance. It is accomplished by including hypermedia links within the responses.

For example in order to discover the base structure of the REST API of the application the client needs to send GET request to the root of the context path (in this case it is URL *"/api"*). It returns following data in JSON format:

```
{
  "_links":{
    "solutions": {
      "href": "http://localhost:8080/api/solutions{?page,size,sort}"
    },
    "fields": {
      "href": "http://localhost:8080/api/fields{?page,size,sort}"
    },
    "categories": {
      "href": "http://localhost:8080/api/categories{?page,size,sort}"
    },
    ...
  }
}
```

Now the client is able to locate each of the available resources by their URLs. When accessing needed resource the client would get locations of available sub-resources as well. For example when accessing *"/api/solutions"* server can return following data:

```
{
  "_embedded":{
    "solutions":[
      {
        "id":5,
        "title":"Do retest",
        "description":"Try to perform the whole test again, maybe you
            missed something.",
        "_links":{
          "self":{
            "href":"http://localhost:8080/api/solutions/5"
```

```
            },
            "proposedIssues":{
              "href":"http://localhost:8080/api/solutions/5/proposedIssues"
            },
            "createdBy":{
              "href":"http://localhost:8080/api/solutions/5/createdBy"
            },
            ...
        }
      ]
    },
    ...
}
```

In this case, the client received information about existing solution instances as a well as locations of other resources that relate to them. The structure of the resource hierarchy in REST API represents suggested business domain model and organized for each entity similarly as the example mentioned above. Each of the resources can be accessed or manipulated with the POST, GET, PUT, DELETE HTTP methods that represent standard CRUD operations.

5.1.2 Service Layer

The service layer has several purposes. It manages:

- Application business logic by using the combination of injected JPA repositories to perform more complex operations rather than just one query.
- Usage of utility classes.
- Logging by using Simple Logging Facade for Java. It is basically an interface for logging which in this case is implemented by Logback – a logging framework that Spring Boot uses by default.
- Transaction management by using Spring declarative transaction implementation.
- Exception handling and creation which is then delegated to the REST controllers.

5.1.3 Data Access Layer

Data access layer is represented by Spring Data module that uses JPA repositories which offer predefined CRUD operations combined with pagination. Each JPA repository is based on corresponding JPA entity that is mapped to a table in relational database. To reduce redundant duplicity of columns in database joined inheritance is used. In order to write custom queries, either JPQL or query creation mechanism is used [26].

Spring Data requires some persistent provider to be able to make transactions to the database. It can be configured by defining the entity manager bean. In case of this application it is not defined, so Hibernate persistent provider is used by default.

For accessing RDF data JOPA semantic framework is used. It is used to map Java object to appropriate concepts in RDF database. All the repositories for accessing RDF data are manually written using JOPA entity manager.

Each JOPA entity type, as well as its properties, has URI assigned to uniquely identify it. Each URI represents a certain concept that describes an attribute or an entity type.

Let's take an example: consider the issue type entity type from the figure 4.2 which has description attribute. Because issue type is stored used RDF description attribute has a URI assigned to it - namely *"http://purl.org/dc/elements/1.1/description"*. Because this concept is adopted from an already existing widely used dictionary, which is often the case, it is exposed online and can be accessed by URI through the browser. There is following information shown on the page on the request:

```
@prefix dcterms: <http://purl.org/dc/terms/>
...
<http://purl.org/dc/elements/1.1/description>
dcterms:description "Description may include but is not limited to: an
    abstract, a table of contents, a graphical representation, or a free-
    text account of the resource."@en ;
...
```

It means that resource with above-mentioned URI has property *"dcterms:description"* with corresponding value[5]. There are other properties that define this resource as well.

Each other attribute or entity type in the dynamic model can be accessed in the same fashion thus giving them an unambiguous and descriptive definition that is available at any time.

Data access layer is also responsible for the import of the RDF data from remote SPARQL endpoints. Those services that are called by the application but are not managed or deployed under it. Endpoints are read-only and resulting data retrieved from them are stored in the local RDF storage. Imported data are then merged into the local model by adding certain properties in order to treat it as if it was defined by the local model (described in more details in section 4.1.2.1).

When trying to persist or update any of the entities JSR validation is triggered that checks constraints defined by annotations in order to stop invalid entities to enter the database layer. The validation is extended on entities that are connected both with the static and dynamic model to check if the consistency between them is preserved. For example when persisting *field* the validator checks if it has valid reference (key) to a corresponding *field type*. It is the main mechanism behind enforcing consistency between the RDF and relational data models.

5.1.4 Data layer

The data layer consists of 2 parts: static and dynamic.

Static part is represented by tables in relational PostgreSQL database which stores entities that are not designed to be described by RDF dictionaries in this application. Those include instances of projects, users, submissions, fields and categories.

Dynamic part is represented by records in RDF storage. It stores information that is designed to be characterized by RDF ontologies. This includes entities that define their corresponding instances in the static model such as field types and issue types. Another criterion depending on which entities belong to RDF store or not is the sensitivity of information that they contain. Because the concept of RDF is tightly coupled with open data its security aspect is currently questionable [27]. The abstract definitions of field types, issue types and issue statuses do not contain any sensitive information about the organization itself thus it is reasonable to store them in RDF store.

[5] dcterms is alias for the URI *"http://purl.org/dc/terms/"* in this case

Dynamic part is also extended by usage of remote SPARQL endpoints which provide RDF data on demand. The import of data from remote services and local repository is intermediated by data access layer.

Entities from both of parts have a set of constraints that are defined by JPA / JOPA annotations restricting saving or updating entities in the invalid state.

5.1.5 Interactions Between Layers

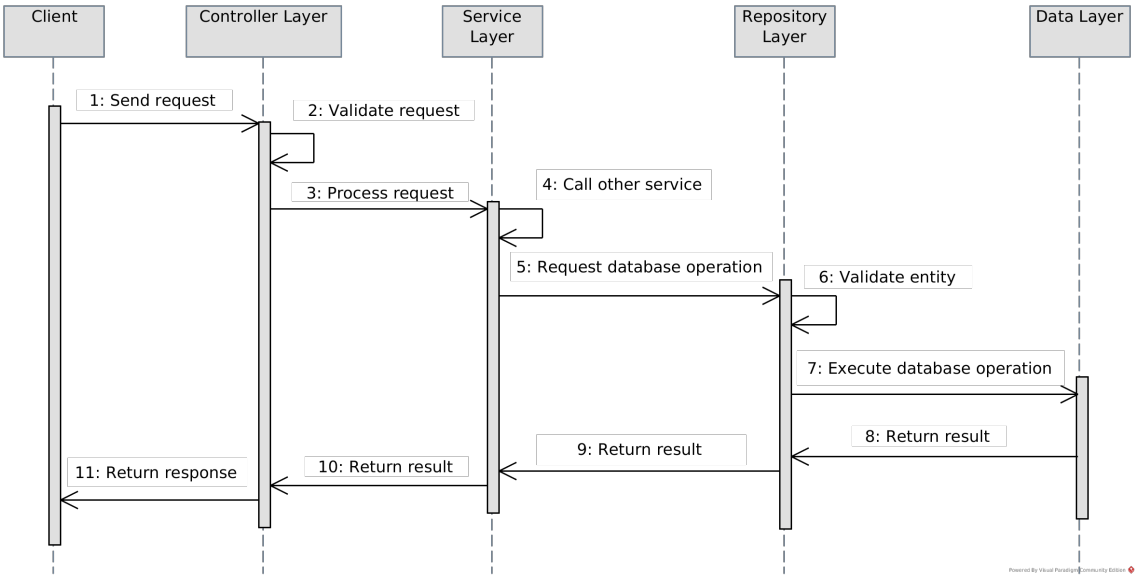

Figure 5.2: Application sequence diagram

The above-given diagram generalizes the communication between the application layers and the client. The process is always initialized from the client that sends an HTTP request to a particular REST endpoint in order to execute certain operation inside the system. The request is then being validated in the controller layer against REST point requirements regarding requests headers and parameters. If the request contains body data it is being deserialized to Java object and then validated by JSR validation in order to filter out invalid entities before they reach underlying layers.

Then the controller calls the service layer to execute an operation based on the received request. The call is being processed by service layer that often involves several layers in the way.

Each of the involved services calls corresponding repositories in order to perform database operations. If the operation involves some kind of input entity it is being validated by the same JSR validation mechanism that takes place in the controller layer. The reason behind this is that the input data from the service layer is not always necessarily come the corresponding REST controller and is not properly filtered out.

After the possible validation takes place the repository layer calls the data layer to perform a certain action in the database. If the action has a result in form of an entity it is propagated to the above-lying layers until it reaches the controller layer where the result is serialized and send in form of JSON to the client.

If there is an error in any of the above-mentioned steps the exception is thrown and propagated till the controller layer where is it resolved and corresponding HTTP response with appropriate status and an error message is generated and send to the client.

Chapter 6

Evaluation

6.1 Usability

In this section suggested implemented system is evaluated by considering several real-life scenarios and how they can be solved by the functionality of the implemented system to demonstrate its usability.[6]

6.1.1 Configurability

The implemented system provides the possibility of configuring issue types, field types and statuses. This way users are able to configure the issue tracking process according to their domain model and requirements.

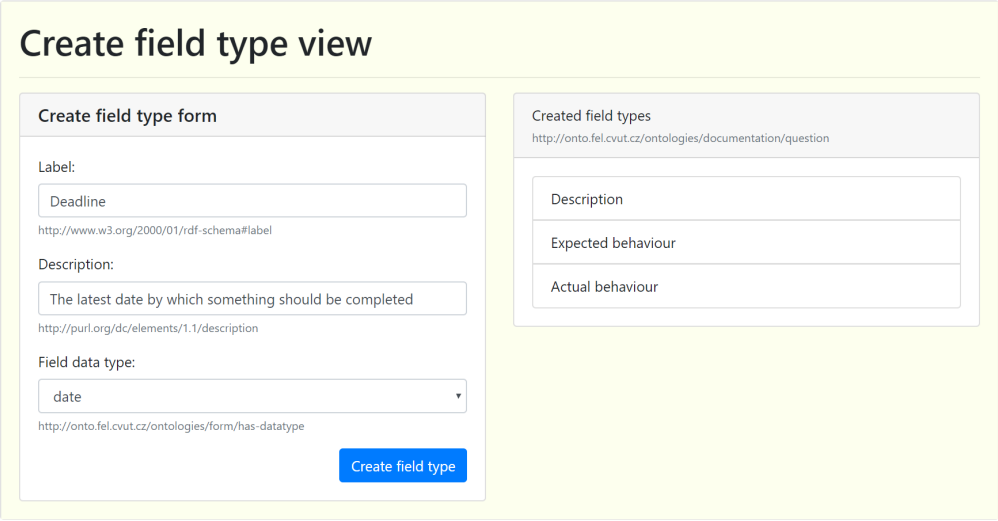

Figure 6.1: Example of *deadline* field type creation

[6]Note: the presented user interface in this chapter is fully operational with implemented back-end services. However, it does not cover all the use cases provided by back-end services and is used only for the presentation of the issue and solution lifecycle thus cannot be considered a suitable solution for real-life usage by users.

Consider the following example of creating a simple issue of type *task*. Firstly it is necessary to define appropriate field types. This includes defining title, description and data type of the field. The titles purpose is to label field so the user can quickly identify what is it for. Description helps a user to understand meaning, purpose and usage of the field in more detail. The data type defines what kind of information does the field contain (for example number, plain text, date et cetera). In case of *task* issue type *description* and *deadline* field types would be created. The *description* field could be described as "The detailed description of the steps that need to be done in order to complete the task" and *deadline* as "the latest time or date by which something should be completed".

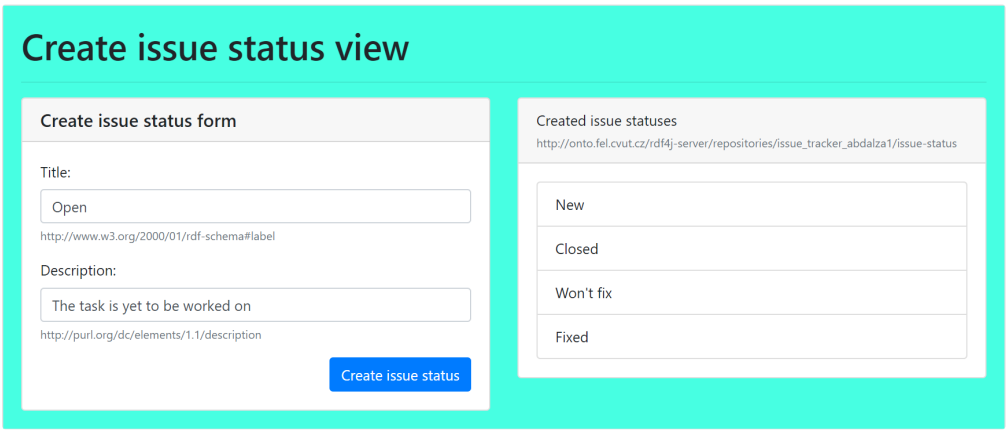

Figure 6.2: Example of *open* issue status creation

Subsequently relevant issue statuses need to be defined. Since *task* issue type is getting created it would be sufficient to create *open*, *in progress* and *closed* statuses. Each of those would also have a relevant description written to them.

Now the issue type itself can be created by defining a title, description[7], set of field types that would be part of the issue and possible statuses that issue can be in. After that instances of this issue type can be created. Created field types and statuses can also be reused by assigning them to different issue types.

The process of defining an issue type in the implemented system is highly inspired by analysed solutions. Nevertheless, it has its own specifics in comparison for example with Jira and YouTrack. The process of the defining a complete issue type is more simple, intuitive and quick.

However, there is a disadvantage that comes with the simplicity of this approach. Because configurations are shared for all projects across the application instance some of the issue types, field types and statuses would be irrelevant in some projects. Consider following scenario. There are two projects created in the application: one for the IT services team and one for the HR team. After carrying out appropriate configurations for both teams in the scope of one application both of the team start to create issues in their corresponding projects. When creating an issue one of the members of IT services team is offered *candidate*, *contract* and *team building* type of issues to choose from amongst relevant IT related issues. The same goes for HR team. It is a minor inconvenience for the user but it can be confusing when using the system.

[7]as a description of the type itself and not as the field in the instance

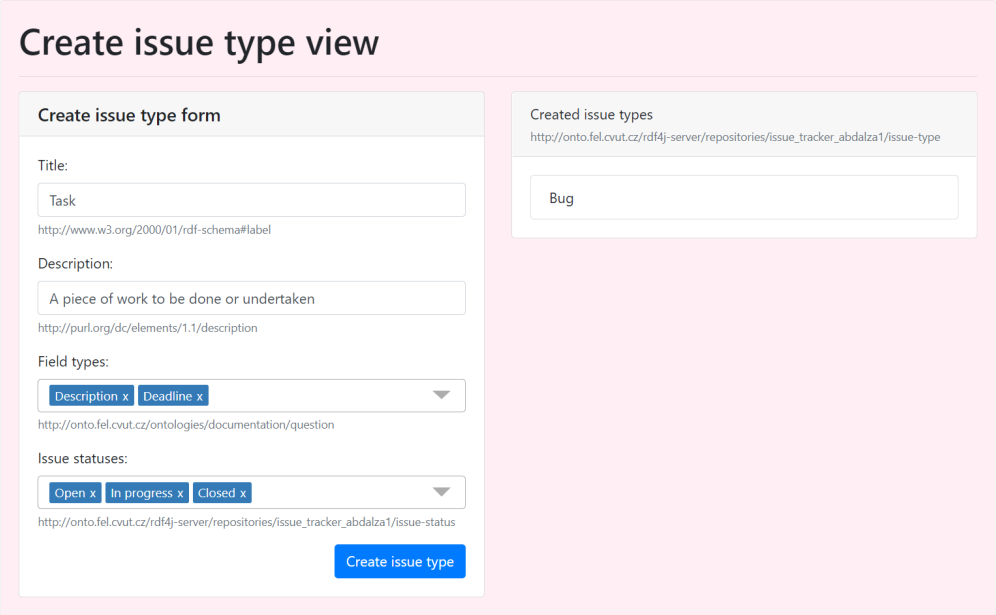

Figure 6.3: Example of *task* issue type creation

Figure 6.4: Example of *task* issue instance

6.1.2 Solution Tracking

The usability of the solution tracking is discussed in the section 1.4.2.3 where as an example situation with defect reports inside the financial enterprise is taken. In current section mentioned problem is addressed in more detail with concrete possible proceedings offered using implemented suggested system.

Consider manual testing team already submitted their defect report and got an answer from the deployment team. In case of this application, the answer of the deployment team is considered a *proposed solution* which means that it is a possible solution (or answer) to the manual testing teams problem. From now on this solution can be found in the same fashion as the issues are found - for example by searching by title, description, author,

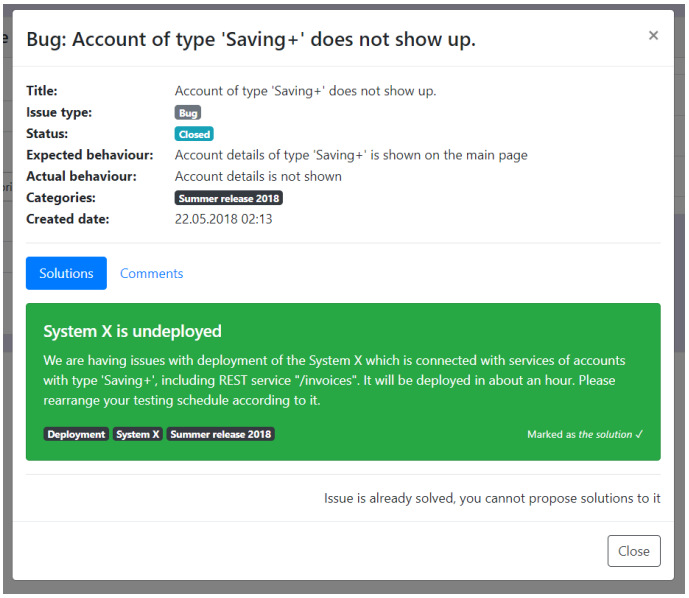

Figure 6.5: Example of solved issue

date submitted and so on. It also can be proposed to other issues making it reusable.

After receiving possible solution from the deployment team the manual testing team can decide whether it is *the solution* to the issue meaning that it is the desired answer to a problem. In this case, the team found the proposed solution reasonable cause to their problem so they mark it as the solution to the issue and then closing the issue.

The second team which is performing integration tests facing an error when calling one of the REST endpoints. Then they submit the relevant defect report to the issue tracking system. One of the procedures that can be done by the integration testing team to find the relevant solution is to search it by today's date. It is reasonable action because integration testing teams are often dealing with other systems unavailability and they might as well check if someone did not mention any build problems in today's solutions. After finding the relevant solution they can propose it to their on the issue and mark it as the solution. In case they did not succeed in finding the solution the same procedure would be performed by deployment team.

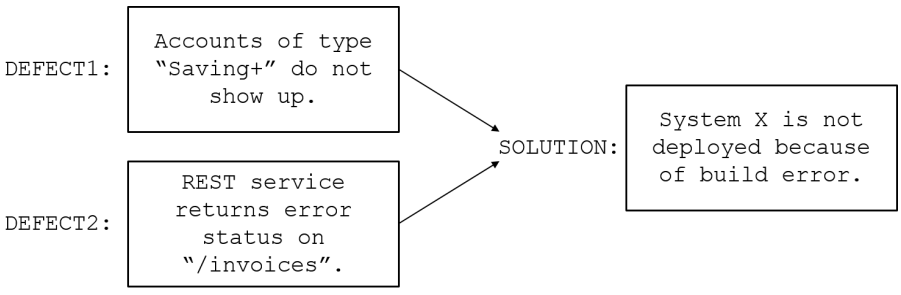

Figure 6.6: Defects and their corresponding solution

Conclusively there is only one answer that is referenced in two issues that is tracked, searchable and ready to be reused in other problems. In case if one of the any analysed issue tracking systems were used the current scenario would end up in the following way:

- The integration testing team had a low probability of finding the relevant answer on their own.
- Two separate answers would be created that hold the same information instead of one.
- In order to find those answers corresponding issues must be found making it a lot less reusable.
- They would be indistinguishable from other comments that were posted to the corresponding issues.

6.1.3 Categorization

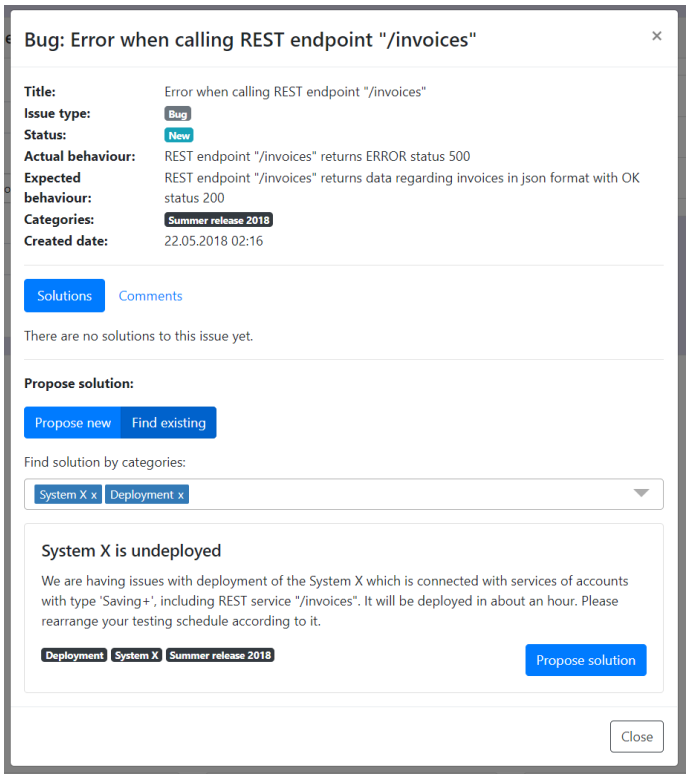

Figure 6.7: Example of finding an existing solution by category

Consider the example of defect reports that was discussed in section 1.4.2.3 at the moment when the integration testing team is trying to find the relevant solution by today's date. In case of the larger enterprise probability of finding the solution by date is quite low because such query could return hundreds of records. That is where issue and solution *categorization* comes into play. s One of the benefits of the categorization of the solution is that it then can be used as attribute in a search query. So when trying to find the information about possible build issues with system X, which again is a reasonable action for a team that works with integration with other systems, they can search by categories such as *"build"*, *"deployment"*, *"System X"* in addition to today's date. Aforementioned query significantly narrows the amount of returned solutions by filtering out irrelevant

ones. Of course the prerequisite for this use case is that team member of the deployment team categorized the solution in a reasonable manner.

In case of using one of the analysed systems, for example YouTrack which is in comparison with other systems offers categorization and advanced search features, the search would need to be applied in following ways because of the absence of the solution feature:

1. By searching the corresponding issue by the same categories and check if the comments contain information about build issues.

2. By searching comments directly if their contents contain one of the keywords.

The first approach might be effective in case corresponding issue was categorized properly but indirect since there is a second step of checking comments manually. The second approach is more reasonable but potentially can return a lot of irrelevant records since comments and solution are indistinguishable within this system.

Another aspect of the categorization are issue types which are used in similar fashion as in other systems. The main difference is the usage of RDF which is described in previous sections.

6.1.4 Configuration Import

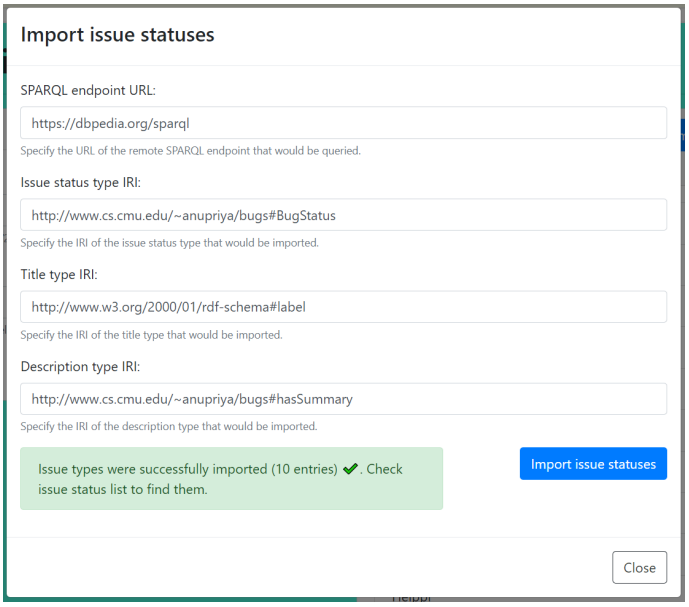

Figure 6.8: Example of importing issue statuses from the remote endpoint

In order to use reuse already existing issue types, field types and statuses defined externally the configuration import feature can be used. Consider the following example of importing issue status definitions. In order to import the configurations URL of the remote SPARQL endpoint must be provided. Then it is necessary to specify the URI which represents the concept of the entity type, same applies to every attribute of the specified entity type. In this particular case, it is sufficient to specify the concepts for the title and description attributes that are used by issue status concept in the remote repository. The import mechanism then finds all defined issue status instances by querying

remote SPARQL endpoint. Each found issue status has its respective title and description properties with values assigned to them. When saving this data to the local RDF repository imported issue statuses preserve the concept that represents their type (in this case *"http://www.cs.cmu.edu/ anupriya/bugs#BugStatus"*) as well as their URIs that represent each instance as a resource (for example *"http://www.cs.cmu.edu/ anupriya/bugs#Closed"*). Nevertheless, the application is able to use this data as if they were defined under the local model (in case of this application the issue statuses are defined by concept because of the mechanism described in section 4.1.2.1. Now users are able to use imported issue statuses for the definition of issue types and assignment to issue instances.

6.2 Testing

In this section suggested implemented system is evaluated by executing unit, integration and performance tests to examine if the system was reasonably designed and implemented regarding technical aspects.

6.2.1 Integration and Unit Testing

The main goal of integration and unit testing is to verify the behaviour of each of the three layers of the application: data, service and controller.

Testing of the data layer is quite trivial because most of its logic is provided by Spring Data module which has its own tests written that can be executed on the demand. Another factor that adds to a simplicity of testing data layer is its genericity - most of the logic is concentrated in parent repository class and is shared for its children thus in order to test all the repository logic it is sufficient to pick one of the children classes to test.

Nevertheless, there is a necessity of writing tests for this layer because firstly some of the Spring Data queries are custom and secondly significant part of the data layer are operations with RDF repository which are all manually implemented.

Most of the service layer consists of operations delegated from the data layer combined with business logic and exception handling on top of that. It has same generic characteristics as the data layer thus testing approach for this layer is similar.

Controller layer is implemented in the same generic fashion as the above-mentioned layers. Nevertheless, it is tested quite differently by using mocks instead of actual services. As a result data setup for tests is more straightforward because it is not as nested as it would be without it. The main concerns of the testing the controller layer is inspection and verification of exception handling, response codes and content types of the body.

Another part of the application logic that was tested is set of utility and validation classes. They are used across all above-mentioned layers of the application.

The resulting code coverage by tests for every part of the application is summarized in the table 6.1.

	By methods	**By lines**
Repository layer	93%	74%
Service layer	96%	95%
Controller layer	62%	75%
Validator	94%	96%

Table 6.1: Summary of the code coverage by tests

6.2.2 Performance Testing

The goal of the performance testing is to verify if responsiveness and reliability of the application are reasonable. It also inspects its operability because the test case mirrors an actual use case of the application that covers the whole lifecycle of the issue and solution tracking.

It begins with the creation of the issue which includes a definition of corresponding issue type, fields types and issue statuses. Then it continues by proposing and picking *the solution* to the issue. After that, it continues by assigning categories to the issue and solution. At the end of the scenario, the issue and solution are being searched in several ways by its categories, contents and other attributes.

The defined scenario is realized via execution of consecutive HTTP requests that represent each step of the lifecycle. It is applied to the applications final endpoints - REST services which are then interacting with underlying layers. The whole process is repeated a certain number of times and is parallelized by the corresponding number of threads thus making it an emulation of the real-life situation when a dozen of users are using the application simultaneously.

The resulting test set consists of 24 operations which are being executed by 100 users in a span of 10 seconds which leads to 2400 operations in total. Users are being added to emulation gradually which makes the heaviness of the load highest at the end of the test. Another factor that adds to the intensity of the load at the end is the accumulation of data - each user creates new issues and solutions thus making it more costly to find those for subsequent users.

The system is able to process all the requests of the performance test without errors with an average response time of 0.8 seconds and Apdex index of 66% in such conditions. The Apdex index shows what percentage of users would be satisfied with the given response time [28]. It is calculated with toleration threshold of 0.5 and frustration threshold of 1.5 seconds.

The test was executed on the application that runs on JVM with a maximum of 750 megabytes of memory allocated and machine with a 2.90GHz processor.

Chapter 7

Conclusion

The accomplished work consists of various parts and achieves several goals.

It contains an analysis of existing issue tracking systems that describe their design choices, functionality and characteristics. On the basis of the performed research, the design of new suggested system emerges which defines its business requirements and characteristics. Subsequently, the suggested system is implemented according to the proposed design. It provides unique features such as solution tracking and categorization of issues and their solutions by RDF dictionaries as well as inherits design choices and practices from already existing systems. It is properly tested and its usability is demonstrated via several real-scenarios mentioned throughout the report.

However, there is a lot of space for improvement regarding the functionality of the application. This includes but not restricted to: more advanced solution tracking with solutions types, workflows, flexible permission and configuration management, issue links and configurable behaviour of field types. Moreover, the front-end part can be fully implemented as the logical continuation of this project.

Appendix A

Existing Solutions - Detailed Examination

A.1 Jira

When creating a project in Jira user is offered to a variety of templates to choose from. Some of them are labelled as “Software Project” and others as “Business project”. The difference between those, apart from the list of templates, is that “Software Project” offers agile boards, integration with development tools and release hub for software versions on top of the default features of “Business project”. [29]

The first type includes such templates as “Scrum”, “Bug tracking”, “Kanban” and “Agility”. The second one includes “Project management”, “Recruitment”, “Procurement” and several others. It is also possible to create a custom template. [30]

The purpose of the templates is to apply pre-defined configurations to a newly created project. After creating project user can add custom configurations or change ones from a template as well. One of those configurations includes issue types which would be used in the project. Each issue type has its own name, icon, description and set of fields which would be contained in an issue of the given type. [31]

For example, if a user has chosen “Recruitment” project template, “Candidate” issue type would be defined in configurations and would have such fields as “Job role”, “Hiring department”, “Hiring manager” et cetera.

In Jira user has a possibility of configuring a custom field. Each custom field includes name, description and custom field type. A user can choose from a variety of field types including standard inputs such as text fields, number fields, date pickers, checkboxes, radio buttons and more advanced ones such as user pickers, group pickers, project pickers, issue participants and others. Then it can be added to certain issue types and configured to be required or even change its properties depending on other issue fields.[32]

When creating an issue in Jira user can link another issue to it. The meaning of the link is based on the chosen link type. Some examples of default link types include “Blocks”, “Duplicates”, “Relates” and “Cloners”. Custom link types can be created and configured in the same matter as custom fields. [33]

Another configurable element of the issues in Jira is issue priority. Basically, it is titled label that has colour, icon, description and level of importance compared to other priorities. [34]

Other important features of Jira issue tracker are issue statuses. Those work as labels

as well, meaning that they have title, colour, icon and description, but on the top of that they are the core concept of workflows. Workflow is a set of statuses and transitions between them. It is used for constraint the number of available statuses to transition to depending on the current issue status. For example, in order to give to an issue "Reopened" status, it has to acquire "Closed" status first. This way user can define custom workflows that represent business processes of the given enterprise. [35]

Projects in Jira issue tracker can be divided into components. Those are used to divide the project into several parts and group issues in them. [36]

Jira offers flexible issue permission management by introducing issue security and permission schemes. By applying a scheme to a project user sets a certain set of rules for actions that can be applied to issues and comments inside the given project. Those rules can be based on a variety of factors: user role, user group, project role, specifically chosen users and more others. [37][38]

A.2 Redmine

When creating a project in Redmine user need to choose which tracker would be used in this project. There are 3 default types of those: *bug*, *feature* and *support*. Trackers are used to divide issues into different types and required to be referenced when creating an issue. There is a possibility of defining custom tracker as well. [39]

Each tracker is required to have a workflow which defines the lifecycle of the issues. It is basically a set of constraints that restrict transitions between certain issue statuses. Workflows are configurable, meaning that user can define custom issue statuses and make constraints for certain transitions. There is also a possibility of defining constraint depending on how a user is related to an issue. For example, only an author of the issue can change its state to "Done" or only an assignee of the issue can make it "In progress" and so on. [39]

Apart from default fields such as subject, description, status, priority and assignee issue can have custom ones. Redmine offers a fair amount of basic field types such as "Boolean", "Date", "Float", "Integer", "Url", "Text" and several others. When creating a custom field a validation can be set to it by using a regular expression. Moreover, a user can define custom fields for projects and user accounts as well. [40]

For managing permissions in projects, user roles are used. When defining user role there is a list of permissions to set to a given role regarding actions that can be applied to an issue. Those permissions are static and not customizable, meaning that they can't be based on issue fields. Thus either certain action is permitted or forbidden unconditionally for a given role. Nevertheless, such an approach is justifiable, because of its straightforwardness and ease of use. [41]

Each project in Redmine can have sub-projects. Those have the same features as a usual project, except they would be referenced on the main page of the parent project and issues of the sub-projects are listed in the parent project. Sub-project can also be divided into sub-projects (and this can be applied an arbitrary number of times) basically creating a tree structure.

A.3 YouTrack

YouTrack provides the possibility of creating custom fields [42]. A user can choose from simple field types such as "string", "date and time", "period", "integer" et cetera and enu-

merated field types such as “enum”, “group”, “user”, “state” and so on[43]. Furthermore, a custom field can be conditional, meaning that it would be shown only if the certain value would be selected on another field. [44][45][46]

YouTrack offers highly customizable workflows, which can be based not only on issue status, arbitrary custom field value or even passed time since the creation of the issue. However, to define a workflow there is need of writing and uploading scripts written in domain language invented by JetBrains which requires at least some familiarity with programming in order to write one. [47]

An issue can be linked to other issues in YouTrack. Some of the default link types are “relates to”, “is required for”, “depends on”, “parent for”, “subtask of” and others. A user can create custom link types as well. [48]

Each issue can be voted for by users of the given project with exception of the author of the issue. When user votes for the issue its vote number goes up by one and user is added to a list of voters that can be viewed under the issue details [49].

When creating an issue YouTrack offers similar issues depending on summary and description user entered in. It is a quite useful feature for linking other issues or avoiding making duplicate ones. [50]

A.4 Basecamp

In order to start tracking tasks in Basecamp, a project must be created. There are two types of projects - teams and projects. After creating a couple of those they would be shown on the main page in two different sections. Basecamp how-to guides state that teams are used to group people with similar roles, whereas projects are used for *“for every major thing you have going on — like a marketing campaign, a new product launch, or a project for a client”* [51]. Apart from the formal meaning and visual appearance on the main page, there is no difference between teams and projects regarding of offered functionality - the task management is the same in both of them.

Appendix B

Application Setup Instructions

In order to run application there are following software required to be installed on the machine:

- Java 8 (1.8.0_171) or higher
- Apache Maven 3.5.0 or higher
- npm 3.10.10 or higher
- Node.js 6.11.4 or higher
- PostgreSQL (tested on 9.6.5)

Then it is necessary to create the user with the corresponding database in PosgreSQL. Once logged in *psql* application in the command line, run following commands:

```
CREATE USER issue_tracker WITH PASSWORD 'issue_tracker';
GRANT ALL PRIVILEGES ON DATABASE 'issue_tracker' TO issue_tracker;
```

Alternatively *application.properties* can be changed in *enterprise-issue-tracker* project in order to set user with different username and password. The RDF store is initialized as an in-memory database for convenience so there is no need to setting it up or dealing with the remote connection. This can also be changed in *application.properties* file. Once the database is set the back-end of the application can be started by execution of *start-be* file. When starting the back-end both SQL and RDF databases are cleared out and filled with sample data which can be accessed via REST services or front-end. After that front-end of the application can be started by executing *start-fe* file. Subsequently, the user-interface can be accessed on *http://localhost:4200* address in the browser. The REST services are available at *http://localhost:8080*.

To execute performance tests firstly start the back-end of the application by executing *start-be* file once the application is ready execute *start-performance-test* file. Resulting test report is available as HTML page at:

```
/enterprise-issue-tracker/target/jmeter/reports/{mostRecent}/index.html
```

Note: after executing of performance test the database is full of test data that might be overwhelming when accessing front-end of the application. In order to clean the database just restart the back-end of the application.

Appendix C

List of Abbreviations

API Application Programming Interface

CRUD Create, Read, Update and Delete

DAO Data Access Object

HTTP Hypertext Transfer Protocol

JOPA Java OWL Persistence API

JPA Java Persistence API

JPQL Java Persistence Query Language

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation for Linked Data

JSR Java Specification Request

JVM Java Virtual Machine

RDBMS Relational Database Management System

RDF Resource Description Framework

REST Representational State Transfer

SKOS Simple Knowledge Organization System

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

WAR Web Application Resource / Web Application Archive

XML Extensible Markup Language

Bibliography

[1] Bertram, D.; Voida, A.; Greenberg, S.; et al. Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, ACM, 2010, pp. 291–300.

[2] Stack Overflow Developer Survey 2017. Stack Exchange, Inc., [cit. 2018-04-27]. Available from: `https://insights.stackoverflow.com/survey/2017`

[3] Klyne, G.; Carroll, B., Jeremy edited by Mcbride. Resource description framework (RDF): Concepts and abstract syntax. 2006, World Wide Web Consortium, [cit. 2018-04-25]. Available from: `https://www.w3.org/TR/rdf-concepts/`

[4] Fielding, R. T.; Taylor, R. N. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000, 76-86 pp.

[5] Web Services Architecture. February 2004, World Wide Web Consortium, [cit. 2018-05-03]. Available from: `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest`

[6] Masse, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media, Inc., 2011, 5-6 pp.

[7] Spring Boot Reference Guide. Pivotal Software, Inc., [cit. 2018-04-25]. Available from: `https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/`

[8] Porter, B.; Zyl, J. v.; Lamy, O. Maven – Welcome to Apache Maven. Apache Software Foundation, [cit. 2018-05-03]. Available from: `https://maven.apache.org/`

[9] Java Persistence API. [cit. 2018-04-25]. Available from: `http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html`

[10] Ledvinka, M.; Kostov, B.; Křemen, P. JOPA: Efficient Ontology-Based Information System Design. In *International Semantic Web Conference*, Springer, 2016, pp. 156–160.

[11] SPARQL Query Language for RDF. January 2008, World Wide Web Consortium, [cit. 2018-05-03]. Available from: `https://www.w3.org/TR/rdf-sparql-query/`

[12] Gierke, O.; Darimont, T.; Strobl, C.; et al. Spring Data JPA - Reference Documentation. Pivotal Software, Inc., [cit. 2018-04-25]. Available from: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

[13] Hibernate ORM. Red Hat, Inc., [cit. 2018-05-03]. Available from: http://hibernate.org/orm/

[14] Brisbin, J.; Gierke, O.; Turnquist, G. Spring Data REST - Reference Documentation. Pivotal Software, Inc., [cit. 2018-04-25]. Available from: https://docs.spring.io/spring-data/rest/docs/current/reference/html/

[15] PostgreSQL: About. PostgreSQL Global Development Group, [cit. 2018-04-26]. Available from: https://www.postgresql.org/about/

[16] Spring Security Reference. Pivotal Software, Inc., [cit. 2018-04-26]. Available from: https://docs.spring.io/spring-security/site/docs/5.0.3.RELEASE/reference/htmlsingle/

[17] JUnit - About. [cit. 2018-04-26]. Available from: https://junit.org/junit4/index.html

[18] Mockito framework site. [cit. 2018-05-03]. Available from: http://site.mockito.org/

[19] Apache JMeter™. Apache Software Foundation, [cit. 2018-04-26]. Available from: https://jmeter.apache.org/

[20] Redmine. Jean-Philippe Lang, [cit. 2018-04-27]. Available from: https://www.redmine.org/

[21] YouTrack: Issue Tracking and Project Management Tool for Developers. JetBrains s.r.o., [cit. 2018-04-27]. Available from: https://www.jetbrains.com/youtrack/

[22] RDF. March 2014, World Wide Web Consortium, [cit. 2018-04-23]. Available from: https://www.w3.org/RDF/

[23] SparqlEndpoints. World Wide Web Consortium, [cit. 2018-05-23]. Available from: https://www.w3.org/wiki/SparqlEndpoints

[24] XML Schema Datatypes in RDF and OWL. March 2006, World Wide Web Consortium, [cit. 2018-05-08]. Available from: https://www.w3.org/TR/swbp-xsch-datatypes/

[25] Alarcon, R.; Wilde, E.; Bellido, J. Hypermedia-driven RESTful service composition. In *International Conference on Service-Oriented Computing*, Springer, 2010, pp. 111–120.

[26] Gierke, O.; Darimont, T.; Strobl, C.; et al. Spring Data JPA - Reference Documentation. Pivotal Software, Inc., [cit. 2018-05-03]. Available from: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

[27] Thuraisingham, B. Security issues for the semantic web. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, IEEE, 2003, pp. 633–638.

[28] Application Performance Index – Apdex Technical Specification. January 2007, Apdex Alliance, Inc., cit [2018-05-10]. Available from: http://apdex.org/documents/ApdexTechnicalSpecificationV11_000.pdf

[29] Jira applications overview - Atlassian Documentation. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/jirasoftwarecloud/jira-applications-overview-764478250.html

[30] Create a Template - Atlassian Documentation. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/doc/create-a-template-296093779.html

[31] Issue types. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/adminjiracloud/issue-types-844500742.html

[32] Configuring a custom field. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/adminjiraserver/configuring-a-custom-field-938847235.html

[33] Linking issues. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/jiracoreserver073/linking-issues-861257339.html

[34] Defining priority field values. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/adminjiraserver071/defining-priority-field-values-802592398.html

[35] Working with workflows. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/adminjiraserver071/working-with-workflows-802592661.html

[36] Defining a Component. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/jira064/defining-a-component-720412395.html

[37] Managing Project Permissions. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/jira064/managing-project-permissions-720412504.html

[38] Configuring issue-level security. Atlassian Corporation Plc, cit. [2018-04-28]. Available from: https://confluence.atlassian.com/adminjiraserver073/configuring-issue-level-security-861253265.html

[39] Redmine - Issue tracking system. Jean-Philippe Lang, cit. [2018-04-28]. Available from: http://www.redmine.org/projects/redmine/wiki/RedmineIssueTrackingSetup

[40] Redmine - Custom fields. Jean-Philippe Lang, cit. [2018-04-28]. Available from: http://www.redmine.org/projects/redmine/wiki/RedmineCustomFields

[41] Redmine - Roles and permissions. Jean-Philippe Lang, cit. [2018-04-28]. Available from: http://www.redmine.org/projects/redmine/wiki/RedmineRoles

[42] Custom Fields. JetBrains s.r.o, [cit. 2018-05-08]. Available from: https://www.jetbrains.com/help/youtrack/standalone/Custom-Fields.html

[43] Custom Field Types. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/Supported-Custom-Field-Types.html

[44] Manage Custom Fields. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/Manage-Custom-Fields-Per-Project.html

[45] Conditional Custom Fields. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/conditional-custom-fields.html

[46] Custom Field Settings. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/Field-Properties.html

[47] Workflows. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/incloud/Workflow-Guide.html

[48] Issue Link Types. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/Link-Types.html

[49] Vote for Issues. JetBrains s.r.o., [cit. 2018-04-28]. Available from: https://www.jetbrains.com/help/youtrack/standalone/vote-for-issues.html

[50] Andrianova, V. Similar Issues in YouTrack 5.0: Just Avoid Duplicates. July 2013, jetBrains s.r.o., [cit. 2018-04-28]. Available from: https://blog.jetbrains.com/youtrack/2013/07/similar-issues-in-youtrack-5-0-just-avoid-duplicates/

[51] Teams and Projects. Basecamp, [cit. 2018-04-28]. Available from: https://3.basecamp-help.com/article/25-teams-and-project