TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Vladimir Kulagin 164225IAPB

WEB APPLICATION FOR CREATING REPORTS ON SOFTWARE DEVELOPMENT PROJECTS AND COLLECTING CUSTOMER FEEDBACK

Bachelor's thesis

Supervisor: Deniss Kumlander
Doctor's degree

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Vladimir Kulagin 164225IAPB

VEEBIRAKENDUS TARKVARAPROJEKTIDE ARUANNETE KOOSTAMISEKS JA KLIENTIDE TAGASISIDE KOGUMISEKS

bakalaureusetöö

Juhendaja: Deniss Kumlander
Doktorikraad

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Vladimir Kulagin

21.05.2019

Abstract

The goal of this thesis is to automate processes of creating reports on software development projects, where the projects data should be acquired from Jira. This thesis is focused on the back-end part developing and provides a detailed description of its architecture. The result of this thesis is a web application, which provides all the required functionality to prepare reports on software development projects.

This thesis is written in English and is 47 pages long, including 6 chapters, 14 figures and 19 tables.

Annotatsioon

Veebirakendus tarkvaraprojektide aruannete koostamiseks ja klientide tagasiside kogumiseks

Antud bakalaureusetöö eesmärk on tarkavaraprojektide aruannete koostamise ja klientide tagasiside kogumise protsessi automatiseerimine. Bakalaureusetöö on kontsentreeritud *back-end'i* osa arendamisele ning annab detailne ülevaade selle arhitektuurist. Töö tulemiks on veebirakendus, mis pakub kogu vajalikku funktsionaalsust tarkavaraprojektide aruannete koostamise ja klientide tagasiside kogumise protsessi automatiseerimiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 47 leheküljel, 6 peatükki, 14 joonist, 19 tabelit.

List of abbreviations and terms

API	*Application Programming Interface*
DB	*Database*
DTO	*Data Transfer Object*
IT	*Information Technology*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
LOC	*Lines of Code*
MD	*Man-Day*
NCLOC	*Non-Commented Lines of Code*
REST	*Representational State Transfer*
SCRUM	Agile framework for managing product development.
SDK	*Software Development Kit*
SSL	*Secure Sockets Layer*
UI	*User Interface*
XML	*Extensible Markup Language*

Table of contents

List of figures

List of tables

1 Introduction

1.1 Background

Kuehne + Nagel is a global transport and logistics company. It has more than 1000 of offices among many countries, including IT centers, which belong to company's IT unit. These centers provide IT solutions for the company and one of these centers was the place of this thesis author's internship. The official name of this center is Kühne + Nagel IT Service Centre AS.

1.2 Problem

One of this IT center tasks is software development. The most part of the software projects are developed using SCRUM methodology and at the end of each sprint project team is giving feedback to the client about how much of project budget has been spent and what has been done during the sprint. For the project managing, the Jira is being used, thus all the budget calculations are being done according to the time tracking data from Jira. In addition, at the end of each sprint a questionnaire is being sent to the end users and the project team, including developers, sponsors, stakeholders, etc. In this way, for each sprint the report is being prepared. This report contains overview of sprint goals, sprint tasks, sprint costs calculations and the results of sprint satisfaction survey.

The problem is that all report preparations are being done manually by a responsible person: sprint costs calculations, sprint tasks overview, survey sending, calculation of average satisfaction rating and charts drawing. The disadvantage of this approach is that all these preparations take time and there is a possibility of human mistakes in the calculations.

1.3 Purpose

The given bachelor's thesis purpose is to automate processes of creating software project reports. This would not only save a lot of time on creating these kind of reports, but also

will significantly reduce the possibility of human mistakes during the cost and rating averages calculations. The main goal is to build software, which will provide all needed functionality for automation the reports creating. The expected result is application, which will allow getting overview of software projects, sprints and issues and calculate their costs, send sprint surveys to the groups of users and provide the results.

1.4 Project scope

The project has been done by two-member team. This thesis author's contribution to the project is the whole back-end part development including the database design. The front-end has been developed by another team member. Thus, this thesis is focused on the back-end part developing and provides a detailed description of its architecture. Since the application size regarding the code is not small, in this thesis a description is provided only of those main components, which are needed to understand the project. Unimportant and minor services are not described.

2 Problem analysis

This chapter provides a detailed description of all requirements regarding the application and a list of chosen technologies for application development.

2.1 General requirements

The list of general requirements is represented below.

1. All data – projects data, sprints data and issues data, including time tracking data, has to be taken from Jira environment.

2. Application (for viewing the reports) should be available also for these users, who does not have access to Jira.

3. Only specific users should be able to login to the application and create reports. This means that not all users, who have access to Jira, should have the possibility to login to the application.

4. The data related to the projects, sprints and issues should not be updated in live mode. This means that if, for example, the new issues are added to the certain sprint or the time tracking data is changed in the Jira environment, this changes should not appear in the application itself. All changes done in the Jira should appear in the application only when user decides to update sprint or project report.

2.2 Functional requirements

The list of functional requirements is represented below.

1. It should be possible to get overview of the projects, project sprints and sprint issues without logging into the system.

2. It should be possible to get overview of sprints and issues costs without logging into the system.

3. It should be possible to create project reports which consist of sprint reports.
4. It should be possible to unpublish certain project or sprint report.
5. It should be possible to update sprint or project report. After update, all changes done to the certain project or sprint in Jira have to be applied in application with costs recalculations.
6. It should be possible to create sprint goals, which can be related to sprint issues and which costs should be calculated according to related issues costs. In addition, it should be possible to set status of sprint goal to „achieved" or „not achieved".
7. It should be possible to send sprint satisfaction surveys to the users email.
8. It should be possible to get overview of sprint satisfaction survey results with calculated averages.
9. It should be possible to give access to the application for creating reports to another users.
10. It should be possible to modify project style and save the changes.

2.3 Technological requirements

There is only one technological requirement and it is regarding the database. The database management system, which has to be used is PostgreSQL 10. For the rest there is no any strict requirements and the technology choice for the development is free.

2.4 Application use scope

The use scope of the application will be a private corporate network, which is available only for corporation members. This means that the application will not be accessible from the global network.

2.5 Defining application architecture

According to the general requirements, it is impossible to build this application as Jira plugin or stand-alone application with all business logic running on client side. These

solutions are not suitable, as non-Jira users should also have access to the published reports. Moreover, there is requirement that the changes made to the data in Jira should not appear immediately in report, so there is need to store the projects, sprints and issues data in the database. Due to this reasons it has been decided to build this application as web application with independent UI and server parts.

2.6 Choice of development tools

The list of all chosen technologies is represented below.

Back-end: Spring Boot, Java, Maven (later replaced with Gradle), Liquibase, PostgeSQL
Front-end: React, npm
Deployment and production: Docker

As was mentioned before, there are no strict technological requirements. In this way for the back-end part has been chosen Spring Boot. Firstly, the most part of web applications in the company are also developed using this tool. Secondly, it reduces lots of development time and increases productivity as it avoids writing lots of boilerplate code. Finally, this thesis author has previous experience with using this tool. Thus, a Spring Boot claims to be a good choice for the back-end part development.

The chosen build tool for the back-end is Maven, which has been later replaced with Gradle by a person, who has been deploying the application. The reason of this replacement has left unknown.

The chosen database management system is PostgreSQL 10, as it is technological requirement, as was described above. For managing and applying database schema changes has been chosen Liquibase as it is widely used in the company.

For the front-end part due to the similar reasons has been chosen React. This choice has been made by a person who is responsible for the front-end development.

The deployment has been done using Docker by a responsible person from the company.

3 Database design

This chapter provides a detailed description of database tables and their relations. As the most part of data validation is being done using Spring Boot, the database tables are not overwhelmed with check constraints.

3.1 Project pulse physical model

A project pulse represents a project report. A detailed description of project pulse DB table is provided in Table 1 and the project pulse model is represented in Figure 1**.**

Table 1. Project pulse database table description.

Column name	Description	Example value
project_key	A unique identifier of the project. Is obtained from Jira environment.	TP
name	A name of the project. Is obtained from Jira environment.	Test Project
rapid_view_id	A Jira board or also called rapid view unique identifier, integer value, which is related with given project. It is needed to obtain from Jira all the sprints that are related with the project.	0123456789
is_published	A boolean value, which represents whether the project pulse is published (visible for users).	true
sponsored_budget	A double value, which represents a total budget, which was sponsored for the project development. This value has to be entered manually by a project manager on project pulse publishing.	50000.0
budget_spent	A double value, which represents an amount of spent budget for the project development. This value is calculated automatically by the application.	12500.0
total_max_per_sprint	A double value, which represents a maximum amount of budged which can be spent during one sprint. This value has to be entered manually by a project manager on project pulse publishing.	4000.0
service_rate	A double value, which represents a service rate per man-day. This value has to be entered manually by a project manager on project pulse publishing.	450.0

A description of project pulse DB table relations is provided in Table 2.

Table 2. Project pulse database table relations.

Related table name	Description
project_pulse_chart_color	One-to-many relation. Represents a list of chart colors for the style of given project.
project_pulse_background_color	One-to-many relation. Represents a list of background colors for the style of given project.
project_pulse_text_color	One-to-many relation. Represents a list of text colors for the style of given project.
project_pulse_sprint_pulses	One-to-many relation. Table is needed to relate project pulse with sprint pulses.

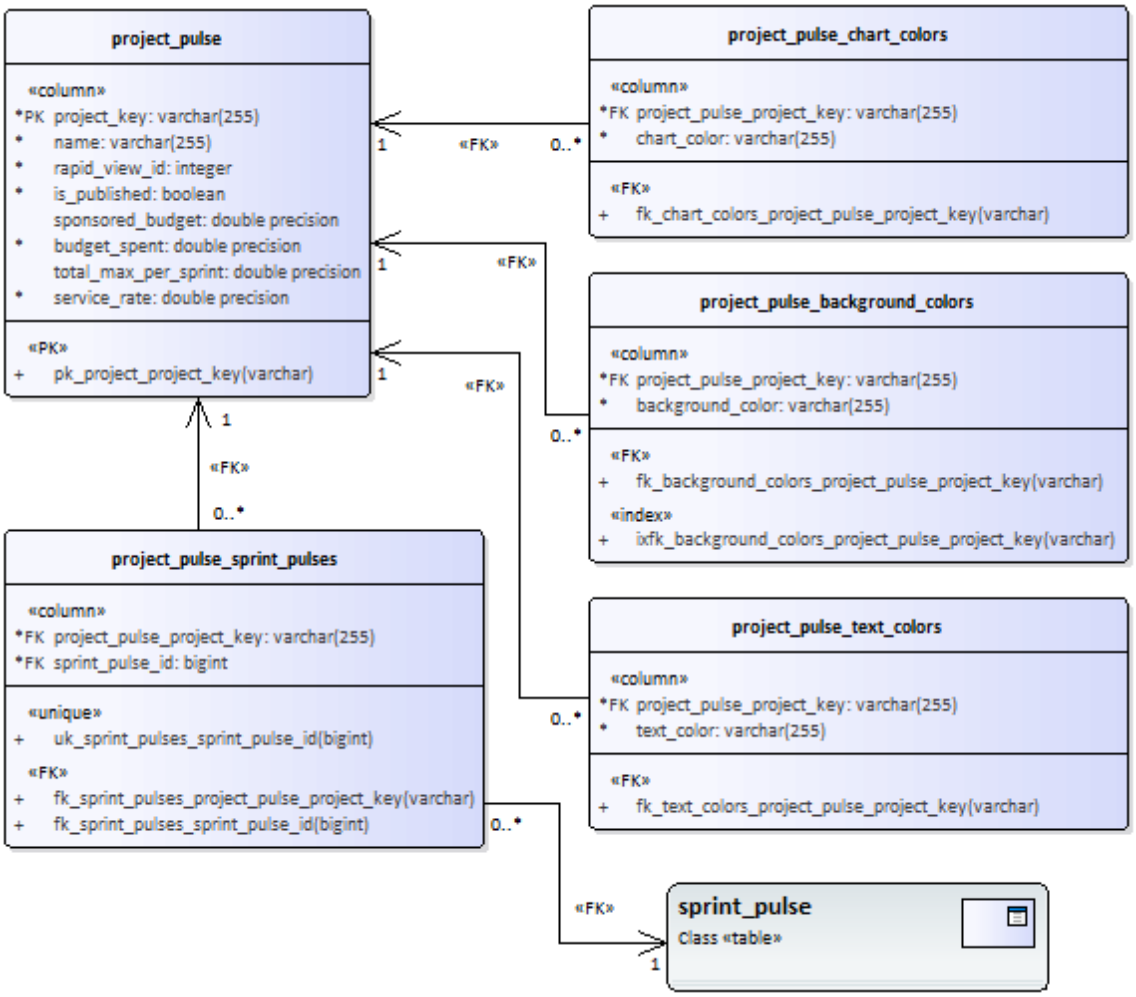

Figure 1. Project pulse physical model diagram.

3.2 Sprint pulse physical model

A sprint pulse represents a sprint report. A detailed description of sprint pulse DB table is provided in Table 3 and the sprint pulse model is represented in Figure 2.

Table 3. Sprint pulse database table description.

Column name	Description	Example value
id	A unique identifier of the sprint pulse. Is generated by the system.	0123456789
sprint_id	A unique identifier of the sprint among the rapid view, but not among the project. Is obtained from Jira environment.	0123456789
name	A name of the sprint. Is obtained from Jira environment.	Sprint 1
is_published	A boolean value, which represent whether the sprint pulse is published (visible for users).	true
start_date	A start date of the sprint. Is obtained from Jira environment.	08.04.2019 00:00:00
end_date	An end date of the sprint. Is obtained from Jira environment.	15.04.2019 00:00:00

A description of sprint pulse DB table relations is provided in Table 4.

Table 4. Sprint pulse database table relations

Related table name	Description
project_pulse_sprint_pulses	One-to-many relation. Table is needed to relate project pulse with sprint pulses.
sprint_pulse_sprint_issues	One-to-many relation. Table is needed to relate sprint pulse with sprint issues.
sprint_pulse_sprint_goals	One-to-many relation. Table is needed to relate sprint pulse with sprint goals.
sprint_pulse_sprint_feedback_list	One-to-many relation. Table is needed to relate sprint pulse with sprint feedbacks.

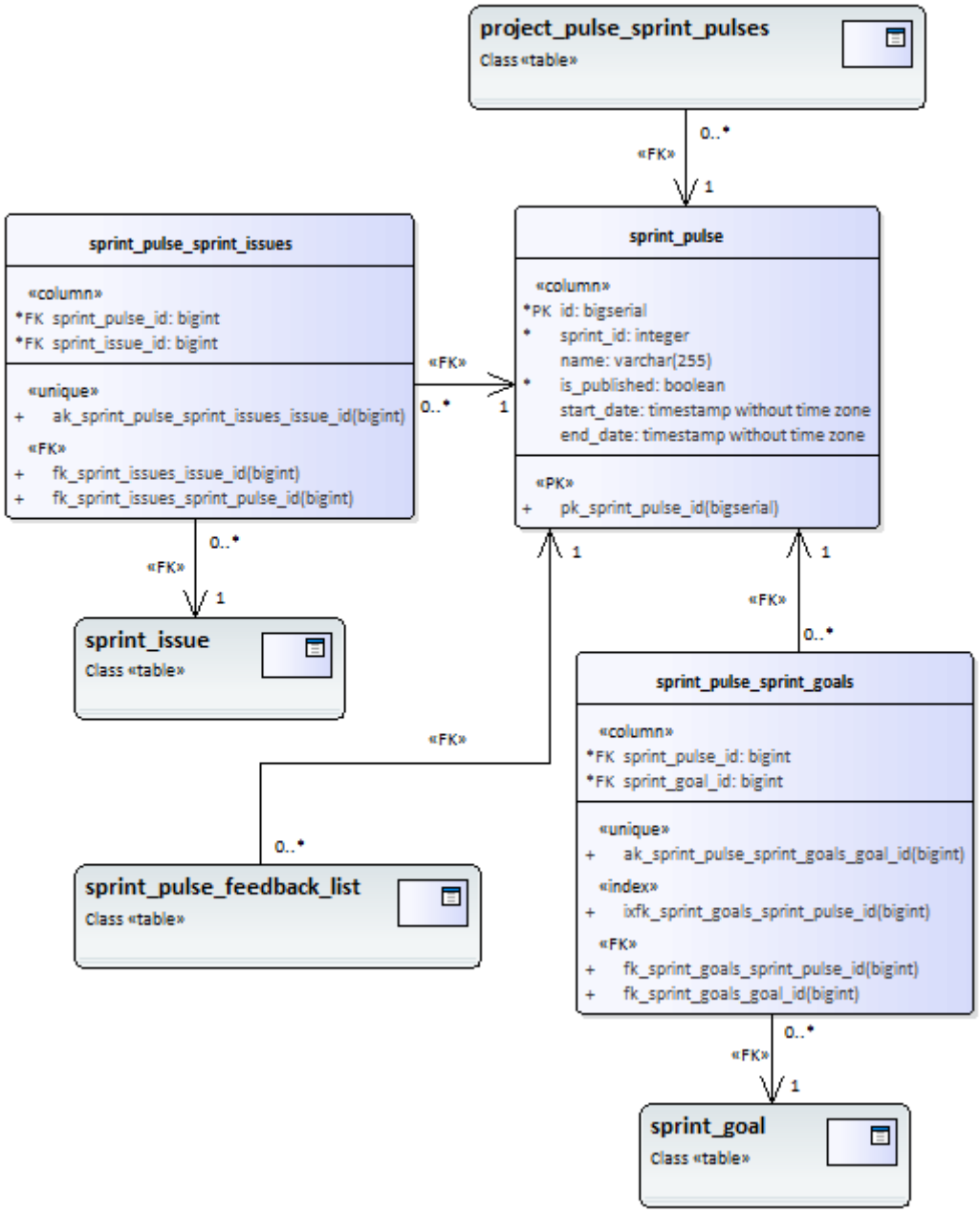

Figure 2. Sprint pulse physical model diagram.

3.3 Sprint issue physical model

A detailed description of sprint issue DB table is provided in Table 5 and the sprint issue model is represented in Figure 3.

Table 5. Sprint issue database table description.

Column name	Description	Example value
id	A unique identifier of the sprint issue. Is generated by the system.	0123456789
issue_key	A unique identifier of the Jira issue. Is obtained from Jira environment.	TP-12
type	A type of the Jira issue. Is obtained from Jira environment.	Task
status	A status of the Jira issue represents a stage the issue is currently at in its lifecycle. Is obtained from Jira environment.	RESOLVED
summary	A summary of the Jira issue. Is obtained from Jira environment.	Implement new feature.
time_spent	A total amount of time in seconds spent for the issue. Is obtained from Jira environment.	18000
cost	A calculated amount of budget in euro spent for the issue. This value is calculated automatically by application according to the spent time.	281.25

A description of sprint issue DB table relations is provided in Table 6.

Table 6. Sprint issue database table relations.

Related table name	Description
sprint_pulse_sprint_issues	One-to-many relation. Table is needed to relate sprint pulse with sprint issues.
sprint_goal_sprint_issues	One-to-many relation. Table is needed to relate sprint goal with sprint issues. Related sprint issues are used to calculate sprint goal cost.

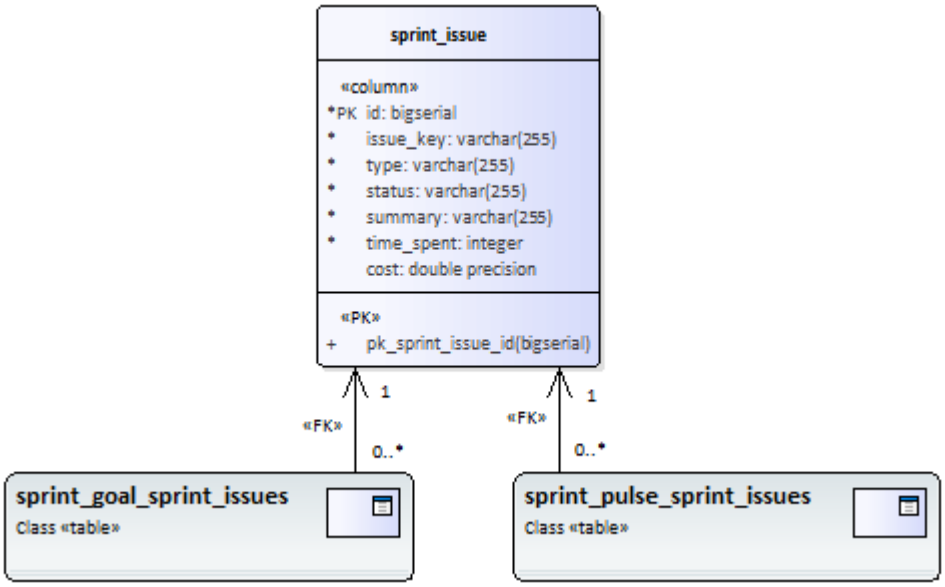

Figure 3. Sprint issue physical model diagram.

3.4 Sprint goal physical model

A sprint goal is not Jira object mapping model. This model is required for creating goals for the sprints and calculating their costs according to the related issues. A detailed description of sprint goal DB table is provided in Table 7 and the sprint goal model is represented in Figure 4.

Table 7. Sprint goal database table description.

Column name	Description	Example value
id	A unique identifier of the sprint goal. Is generated by the system.	0123456789
description	A goal description. This value has to be entered manually by a project manager.	Achieve 1.5 times better performance.
is_achieved	A boolean value, which represents whether is sprint goal achieved during a related sprint.	true

A description of sprint goal DB table relations is provided in Table 8.

Table 8. Sprint goal database table relations.

Related table name	Description
sprint_pulse_sprint_goals	One-to-many relation. Table is needed to relate sprint pulse with sprint goals.
sprint_goal_sprint_issues	One-to-many relation. Table is needed to relate sprint goal with sprint issues. Related sprint issues are used to calculate sprint goal cost.

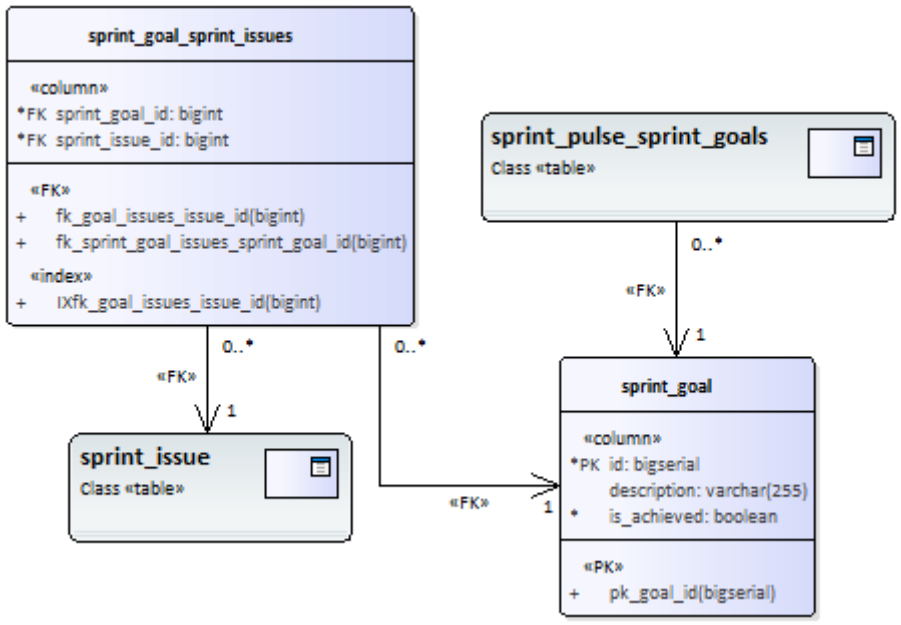

Figure 4. Sprint goal physical model diagram.

3.5 Sprint feedback physical model

A detailed description of sprint feedback list DB table is provided in Table 9 and the sprint feedback model is represented in Figure 5.

Table 9. Sprint feedback list database table description.

Column name	Description	Example value
sprint_pulse_id	A related sprint pulse unique identifier.	0123456789
comment	A given comment for a related sprint.	I am completely satisfied with the sprint results.
rating	A given rating for the related sprint.	5
feedback_user_email	A user email.	user@mail.com
feedback_user_role_name	A role name to which user belongs.	Developers

A description of sprint feedback list DB table relations is provided in Table 10.

Table 10. Sprint feedback list database table relations.

Related table name	Description
sprint_pulse	Many-to-one relation. Table is needed to relate sprint pulse with sprint feedbacks.
feedback_user	One-to-many relation. Table is needed to relate sprint feedback with feedback user.
feedback_user_role_name	One-to-many relation. Table is needed to relate sprint feedback with feedback user role.

A feedback user DB table description is represented in Table 11 and feedback user role DB table description is represented in Table 12.

Table 11. Sprint feedback user table description.

Column name	Description	Example value
email	A unique identifier of the feedback user.	user@mail.com
sprint_pulse_id	A related sprint pulse unique identifier.	0123456789

Table 12. Sprint feedback user role table description.

Column name	Description	Example value
name	A unique identifier of the feedback user role. Represents a role name.	Developers
type	An integer value that represents a role type, which can be „End users” or „Project team”.	0

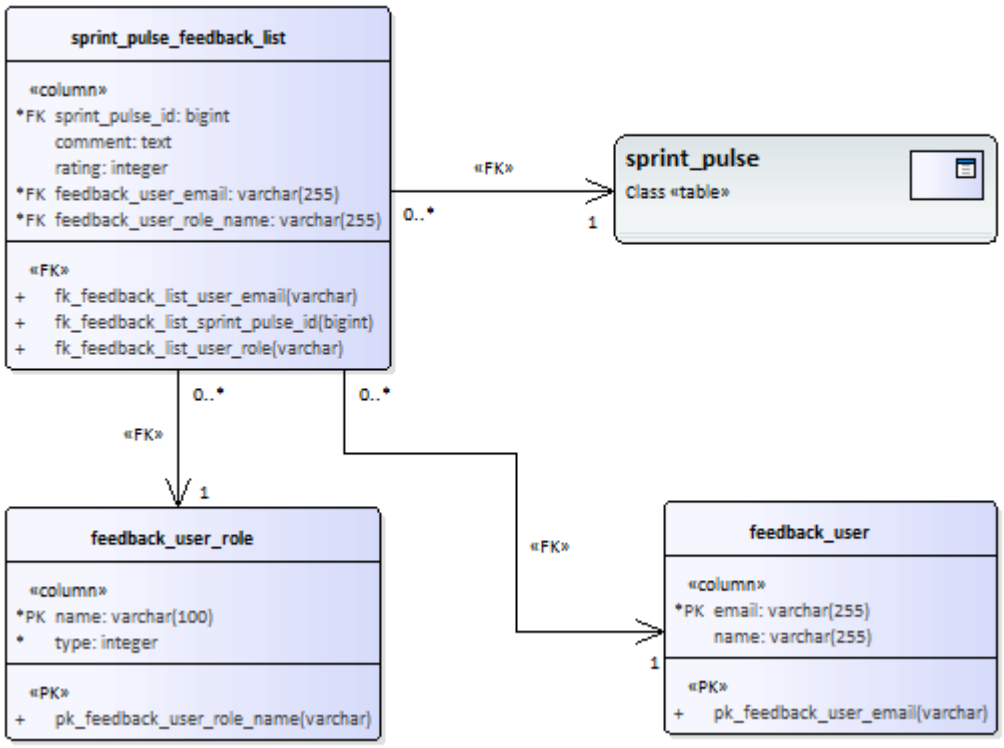

Figure 5. Sprint feedback physical model diagram

3.6 Pulse user physical model

A pulse user represents a user of the application, who is responsible for creating project and sprint reports. A detailed description of pulse user DB table is provided in Table 13 and the sprint goal model is represented in Figure 6.

Table 13. Pulse user table description.

Column name	Description	Example value
email	A unique identifier of the pulse user.	user@mail.com
is_root	A boolean value, which represent whether the user is a root user.	true

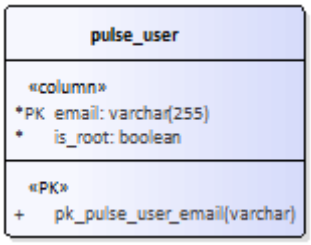

Figure 6. Pulse user physical model diagram.

4 Implementation on Spring

This chapter provides a detailed description of back-end part implementation on Spring. The description is provided only for those main components, which are needed to understand the project. Unimportant and minor services and classes are not described. During the implementation the best practices have been used, which are described in Baeldung [1] and Mkyong [2] articles.

4.1 Spring class model

An entity class model diagram is provided in Figure 7. The diagram shows only entity classes, no service or controller classes are listed here.

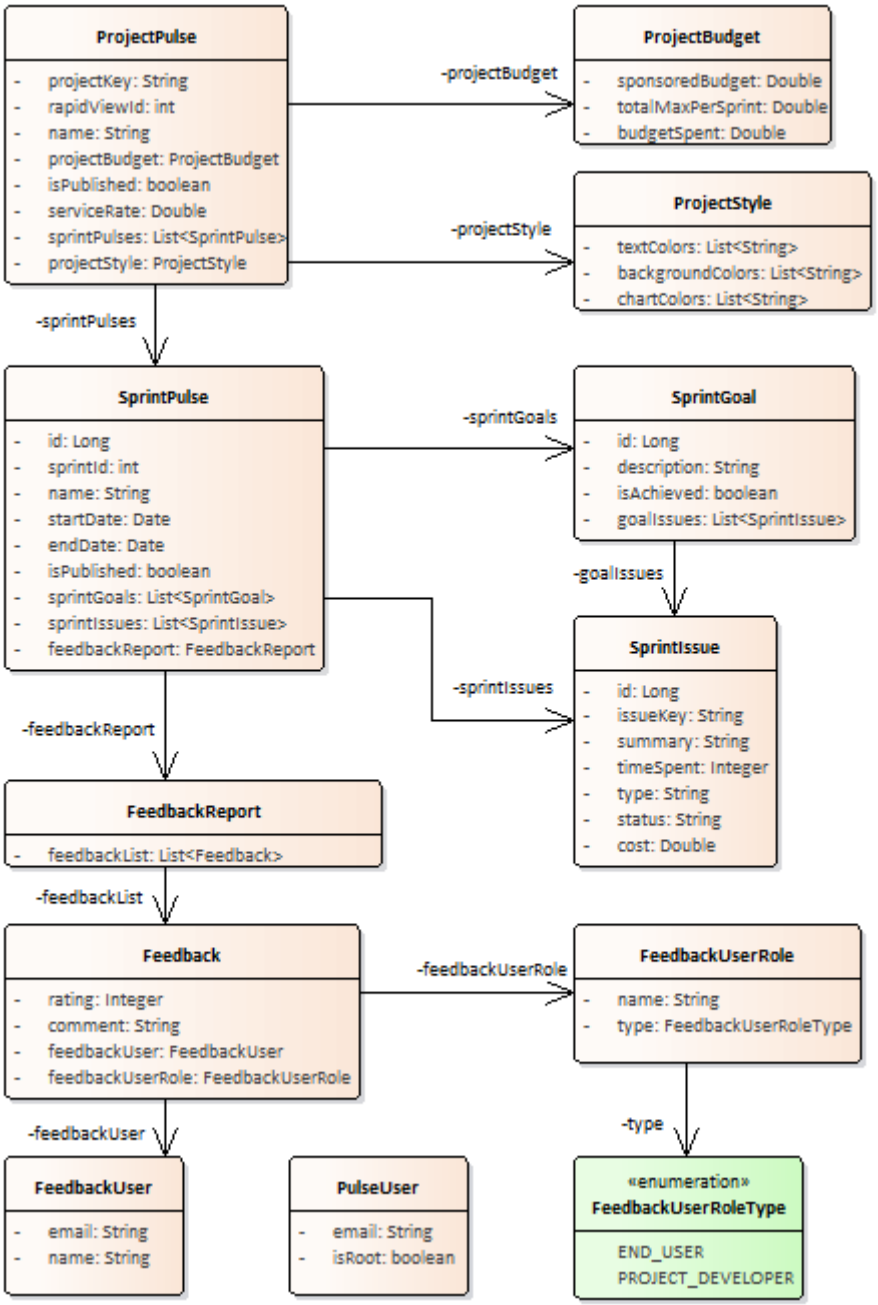

Figure 7. Spring entity class model diagram.

4.2 Jira API integration

There are an official Jira REST APIs, which provide methods to interact with the Jira Server applications remotely. The Jira Server platform provides the REST API for common features, like issues and workflows. The Jira Software and Jira Service Desk applications have REST APIs for their application-specific features, like sprints (Jira Software) or customer requests (Jira Service Desk) [3].

In order to save time on development has been used third party client library for Java, which provides ready methods and models to interact with Jira REST API [4]. All requests to the Jira are done using Jira client class objects, which provide both ready methods and a manual way of requesting data. However, some ready methods of this class do not work as expected and causes errors. In this way for the half Jira API request are used Jira client methods and for another part, HTTP GET requests are made manually using the same class object. The disadvantage of manual requests are that in response we get a plain JSON object instead of corresponding Java object.

4.3 Application controllers

This chapter provides a detailed description of application REST controllers and their methods. All controller methods are used by the front-end of this application.

There are three Spring REST controllers: the first one is used for authorization purposes, the second one is used to interact with the Jira API directly and the last one provides methods for creating and handling project and sprint reports.

The description of Authorization Controller, which is used for authorization purposes, is represented in Table 14. The more detailed description of authorization is provided in chapter 4.7.

Table 14. Authorization Spring controller description.

Method name **Method parameter**	**Method description** **Method parameter description**
/login	Log user into Jira. Retrive a logged user email from Jira, if email is not in the application users list, which is represented by *PulseUser* DB table, then throw authorization exception, as not all Jira users can log into the application. If the login to Jira is succcesful and logged user is also a following application user, then return a cookie with JWT in response.
username	A Jira server username.
password	A Jira server user password.
GET /isLoggedIn	Retrieve JWT from the request and get corresponding Jira client. If Jira client for corresponding user exists, the user is logged in, in this case return true, otherwise return false.
/logout	Retrieve JWT from the request and remove corresponding Jira client.
GET /getUsers	Return list of all application users, which are represented by *PulseUser* DB table. This method is available only for the root users of the application.
/register	Register a new application user. This method is available only for the root users of the application.
pendingEmail	A Jira user email.
/unregister	Unregister an application user. The corresponding user is deleted from *PulseUser* DB table and Jira client of this user is also removed. This method is available only for the root users of the application.
pendingEmail	A Jira user email, which is also registered as this application user.
GET /userHasRootAccess	Retrieve JWT from the request and the user email from it and check wheter the corresponding user is a root user. If it is, then return true, otherwise return false.

The description of Jira controller, which is used to interact with the Jira API directly, is represented in Table 15.

Table 15. Jira Spring controller description.

Method name	**Method description**
GET /projects	This method returns list of all Jira projects.
GET /unpublishedProjects	This method reurns list of these Jira projects for which ProjectPulse instance has not been created.

The Pulse controller provides methods for creating and handling project and sprint reports. The description of given controller is represented in Table 16.

Table 16. Pulse Spring controller description.

Method name **Method parameter**	**Method description** **Method parameter description**
GET /project/{projectKey}/createPulse	Create a project pulse and sprint pulses for sprints of this project. Sprint pulses are initially not published. If project pulse for given project already exists and not published, then recalculate project costs and publish it. Method returns created instance of project pulse.
projectKey	A Jira project key.
serviceRate	A service rate per MD for the given project.
sponsoredBudget	Amount of a sponsored budget for the given project.
totalMaxPerSprint	Total maximum amount of budget which can be spent during the single sprint.
GET /project/{projectKey}/update	Acquire new data from Jira for the whole project pulse and every sprint pulse of the given project. This method adds new sprint pulses to existing project pulse and updates already existing sprint pulses. Sprint issues are also updated and the project and sprint costs are recalculated. Method returns updated instance of project pulse.
projectKey	A Jira project key, which refers to existing project pulse.

Method name **Method parameter**	**Method description** **Method parameter description**
GET /project/{projectKey}/setPublished	Set the project pulse published or vice versa. On unpublishing project pulse data is not being deleted and project pulse can be published again using the same method or with the costs recalculation using the *createPulse* method.
projectKey	A Jira project key, which refers to existing project pulse.
isPublished	A boolean which defines whether the project pulse should be published.
GET /project/{projectKey}/setProjectBudget	Set the project budget values. This method is used to change the budget of existing project. Method returns project pulse instance with updated budget values.
projectKey	A Jira project key, which refers to existing project pulse.
sponsoredBudget	Sponsored budget amount.
totalMaxPerSprint	Total maximum amount of budget which can be spent during the single sprint.
GET /project/{projectKey}/setServiceRate	Set new project service rate and recalculate project issues costs. Method returns project pulse instance with updated service rate value.
projectKey	A Jira project key, which refers to existing project pulse.
dailyServiceRate	A new service rate per MD for the given project.
GET /project/{projectKey}/style	Return instance of the project style for the requested project pulse.
projectKey	A Jira project key, which refers to existing project pulse.
POST /project/{projectKey}/setStyle	Replace current project style with new one. Method returns new project style instance.

Method name **Method parameter**	**Method description** **Method parameter description**
projectStyle	A project style request body, see Figure 7 for more details. Project style consists of lists of background, text and chart colours.
GET /project/{projectKey}/pulse	Return instance of project pulse. If project pulse with given project key does not exist, then throw exception.
projectKey	A Jira project key, which refers to existing project pulse.
GET /pulses	Return all instances of published project pulses.
GET /project/{projectKey}/pulse/sprintPulse/{sprintId} /update	Acquire new data from Jira for the given sprint. Sprint name, period and issues are updated. Project costs are recalculated according to the new data. This method is similar to */project/{projectKey}/update* method, the difference is that instead of the whole project only single sprint is updated. Method returns updated sprint pulse instance.
projectKey	A Jira project key, which refers to existing project pulse.
sprintId	Id of the sprint which pulse should be updated.
GET /project/{projectKey}/pulse/sprintPulse/{sprintId} /setPublished	Set the sprint pulse published or vice versa. Sprint pulse data is not being deleted and sprint pulse can be published again using the same method. On publishing, sprint pulse instance is also updated as described in method above. Method returns project pulse instance.
projectKey	A Jira project key, which refers to existing project pulse.
sprintPulse	Id of the sprint which pulse should be published or vice versa.
POST /project/{projectKey}/pulse/sprintPulse/{sprintId} /addGoal	Add new sprint goal to the sprint pulse or update the existing one. Method returns updated sprint pulse instance.

Method name **Method parameter**	**Method description** **Method parameter description**
projectKey	A Jira project key, which refers to existing project pulse.
sprintId	Id of the sprint to which sprint goal should be added or which goal should be updated.
sprintGoal	A sprint goal request body, see Figure 7 for more details. Can represent a new goal or existing one. If goal already exists, then its instance is updated.
issueKeys	Key of the issues passed as URL parameters. Issues with following keys are related with sprint goal in order to calculate the given goal cost.
GET /project/{projectKey}/pulse/sprintPulse/{sprintId} /goal/{goalId}/remove	Delete sprint goal of the sprint pulse. Method returns updated sprint pulse instance.
projectKey	A Jira project key, which refers to existing project pulse.
sprintId	Id of the sprint, which goal should be deleted.
goalId	Id of the goal, which should be deleted.
GET /project/{projectKey}/surveyGroups	Return list of all survey groups of the given project pulse. Survey group is a DTO, which consists of feedback user role and list of feedback users.
projectKey	A Jira project key, which refers to existing project pulse.
POST /project/{projectKey}/pulse/sprintPulse/{sprintId} /sendSurvey	Send sprint survey to the users of survey groups and save the empty feedback instances to the database. Method returns updated sprint pulse instance.
projectKey	A Jira project key, which refers to existing project pulse.
sprintId	Id of the sprint, which survey should be sent.
surveyGroups	Request body, list of all survey groups of the given project pulse. See previous method.

Method name **Method parameter**	**Method description** **Method parameter description**
GET /project/{projectKey}/pulse/sprintPulse/{sprintId}/addFeedback	Update empty feedback instance created on sprint survey sending.
projectKey	A Jira project key, which refers to existing project pulse.
sprintId	Id of the sprint, which feedback instance should be updated.
encodedEmail	An encoded email string. Email is related to feedback user.
comment	A comment for the sprint. This parameter is optional.
rating	A rating for the sprint, an integer between 1 and 5.

4.4 Project and sprint pulse creation and update flow

The project and sprint pulse creation and update are most important and complicated processes of the given application, as during these processes is done synchronization with Jira environment.

A project pulse instance can be:

1. Created
2. Unpublished
3. Republished
4. Updated

The project pulse instance is created only once per one project. During the project pulse creation, the sprint pulse instances are also created, but only with name and period data, the issues data is acquired during the sprint update. Project pulse can be unpublished, this means only that it becomes invisible for the application users. If try to create project pulse for the project that already has a project pulse, then the budget details of the existing instance will be updated and the pulse will be published. The project pulse creation and republishing flow is represented in Figure 8. The state transitions of the project pulse are represented in Figure 9.

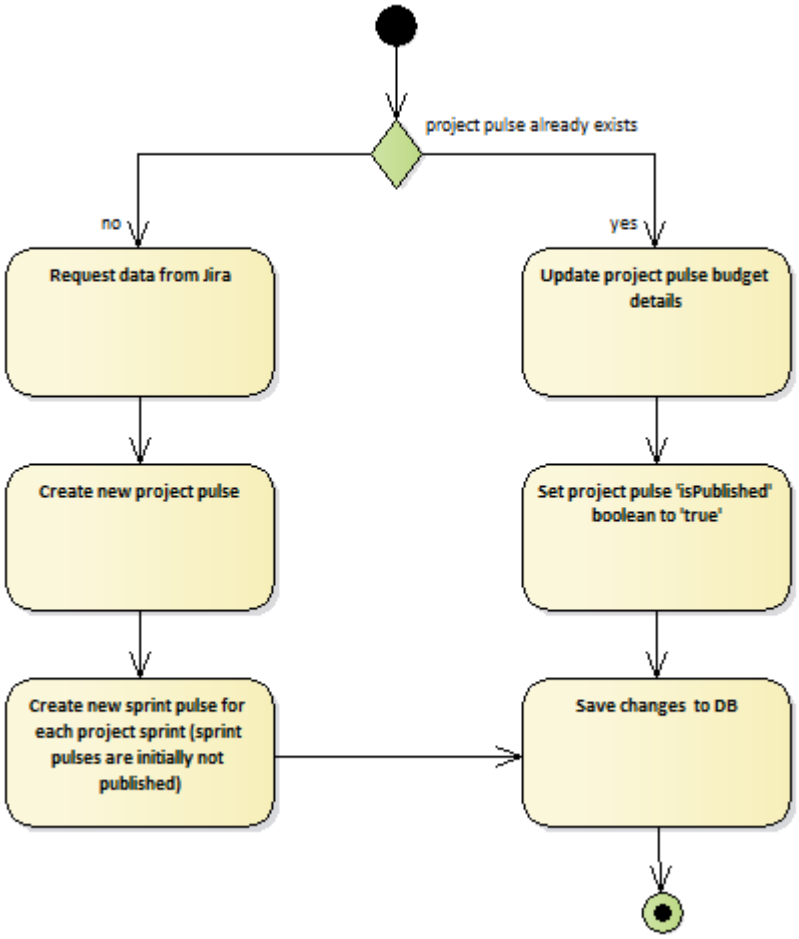

Figure 8. Project pulse creation activity diagram.

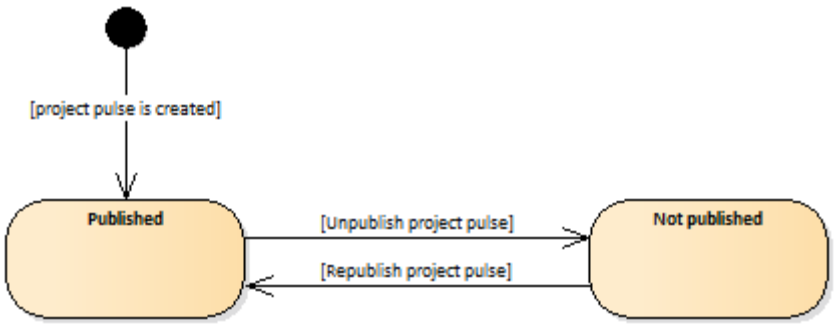

Figure 9. Project pulse state diagram.

A project pulse instance can be also updated. This means that the new data is acquired from Jira and the corresponding changes are done to the project pulse instance. During the update, the sprint pulse instances, which are related to the updatable project pulse, are also updated in the similar way. The update flow of sprint pulse is described in the next paragraph. The update flow of project pulse is represented below in Figure 10.

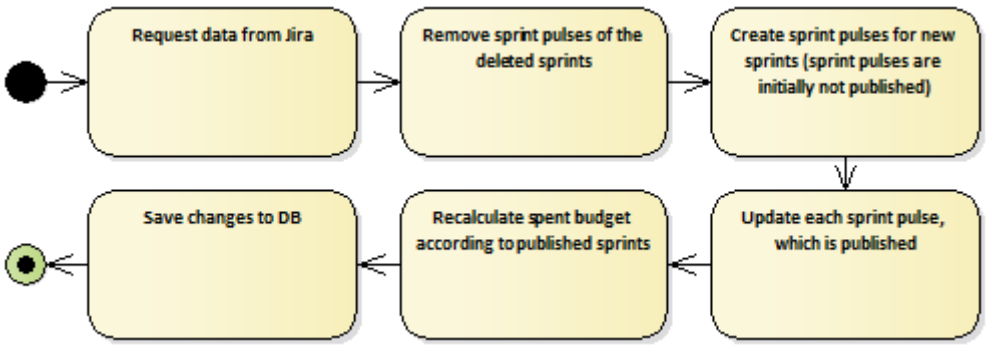

Figure 10. Project pulse update activity diagram.

The case with sprint pulse is very similar; the main difference is that the sprint pulse cannot be created manually, as the sprint pulse instance is created automatically on project pulse creation or project pulse update as was mentioned above. Thus a sprint pulse instance can be:

1. Published
2. Updated
3. Unpublished

After the project pulse with sprint pulse instances is created, the sprint pulse instance can be published. When sprint pulse is being published, it is also being updated automatically. However, a published sprint pulse can be updated manually too. All published sprint

pulses are updated automatically also on project pulse update. The update flow of sprint pulse is represented in Figure 11. The state transitions of the sprint pulse are represented in Figure 12.

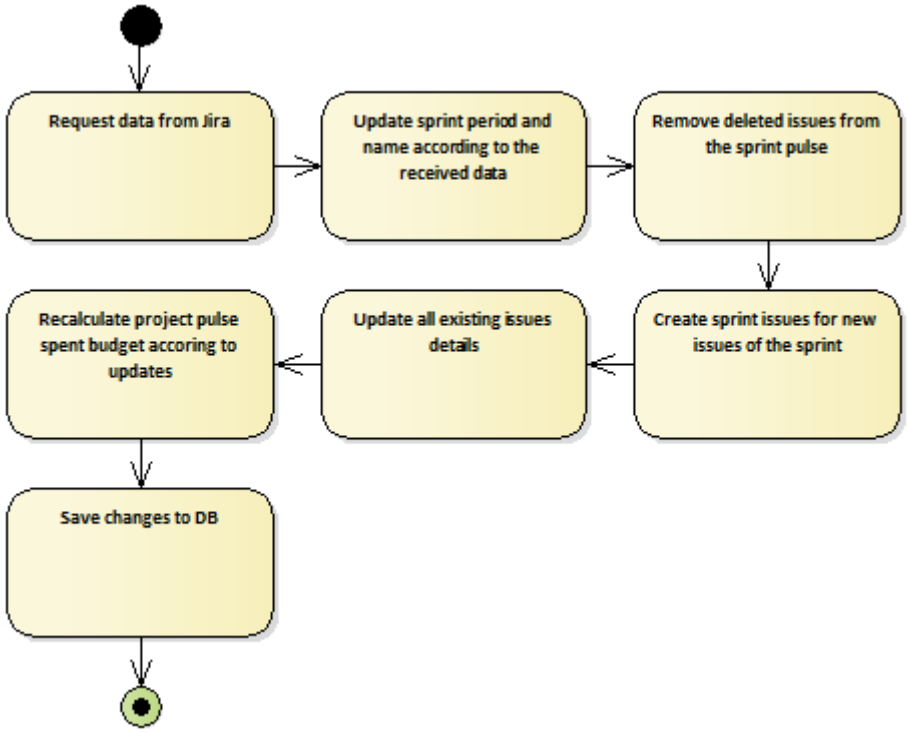

Figure 11. Sprint pulse update activity diagram.

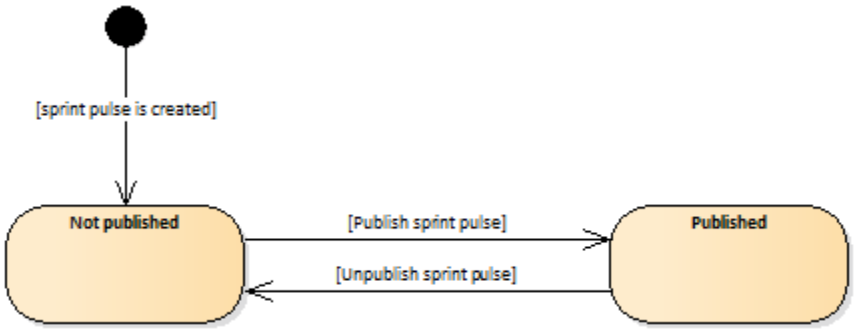

Figure 12. Sprint pulse state diagram.

4.5 Survey sending and feedback results fetching

For the each sprint, it is possible to send a satisfaction survey. When the survey is sent, the empty feedback instances are created. This means that they consist only of a user giving a feedback. See feedback classes in Figure 7. The rating and comment remain unfilled.

To receive feedback and save it to the system it is required a user email. Moreover, the feedback is accepted only from those emails, to which the survey has been sent and for which empty feedback instances have been created, as described above. For more details, see corresponding API method in Table 16.

The rating average values, such as group average and group type average are calculated also on the back-end side. The calculations are done in feedback report class on sprint pulse request with the help of JSON property annotations.

4.6 Application data flow

The application data flow is represented in Figure 13. As it is shown in the diagram, the data requests to the Jira server are made only on project or sprint pulse creation and update. The data received from Jira is being saved to the application database and remains immutable until the next project or sprint pulse update is called. The saved data is further requested by a front-end. It is possible to modify non-Jira related data, such as project style, project budget, sprint goals, sprint feedback.

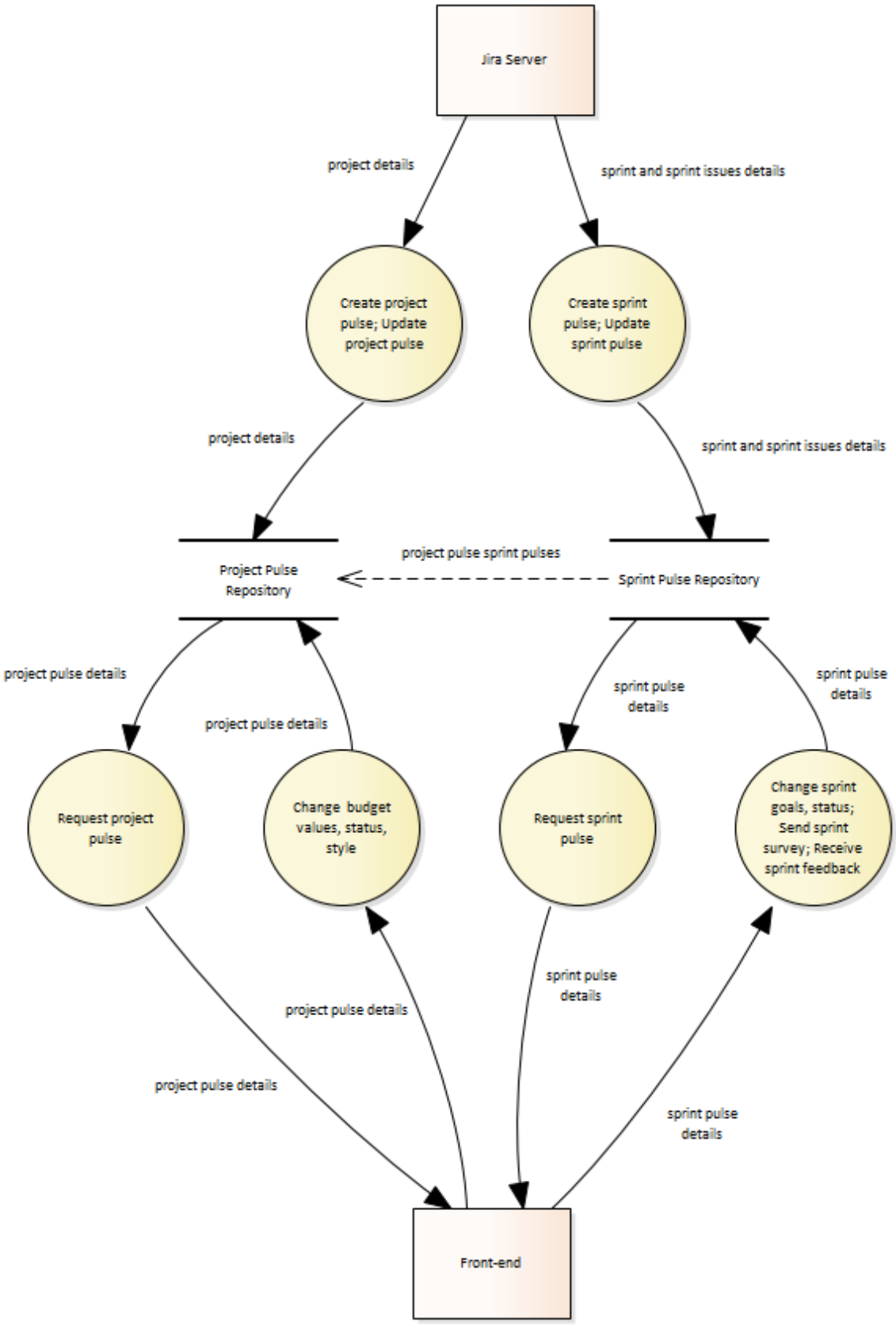

Figure 13. Application data flow diagram

4.7 Authorization

The authorization to the application is done using the Jira credentials, as they are used also to login to the Jira and acquire corresponding Jira client. However, according to the general requirement number 3, which is described in chapter 2.1, not all users, who have access to Jira, should have the possibility to login to the application.

Thus, there are three types of users in this application. The not logged in users, who can only view the project and sprint pulses and have no other permissions. The second type of users are users who can create project and sprint pulses and do all actions related to the project and sprint pulses. This kind of users are also Jira users and they emails are stored in the *PulseUser* DB table. Each user can create project pulses only for these projects to which the user has access in Jira. This means that if user has no acces to some project in Jira environment, he will also unable to create a project pulse for the corrresponding project. The last kind of users are called root users. These users have all permissions as the previous type users. The difference is that root users can register and unregister a second type users and get an overview of all application users. The root users are defined by the application properties.

The Jira API provides three methods of authentication:

1. Basic authentication with a username and password [5],
2. Cookie-based authentication [6],
3. OAuth [7].

As the Jira documentation says, the preferred authentication methods for the Jira REST APIs are OAuth and HTTP basic authentication (when using SSL) [6]. Unfortunately, the OAuth is not supported by the third-party library which is used to interact with Jira and which is described in chapter 4.2. In this way, the basic authentication has been chosen as a way of authentication to the Jira API.

On login to the application, a Jira client is built from user provided credentials, to validate Jira credentials the request to the Jira server is done to retrieve a user email. If request is successful, then it is checked whether the retrieved email contains in application users email list. If user with corresponding email belongs to the application users, then the built

Jira client is saved to the clients list and the JWT is put to the response cookie. Otherwise, the login is unsuccessful and the exception is thrown. The authorization flow is represented in Figure 14.

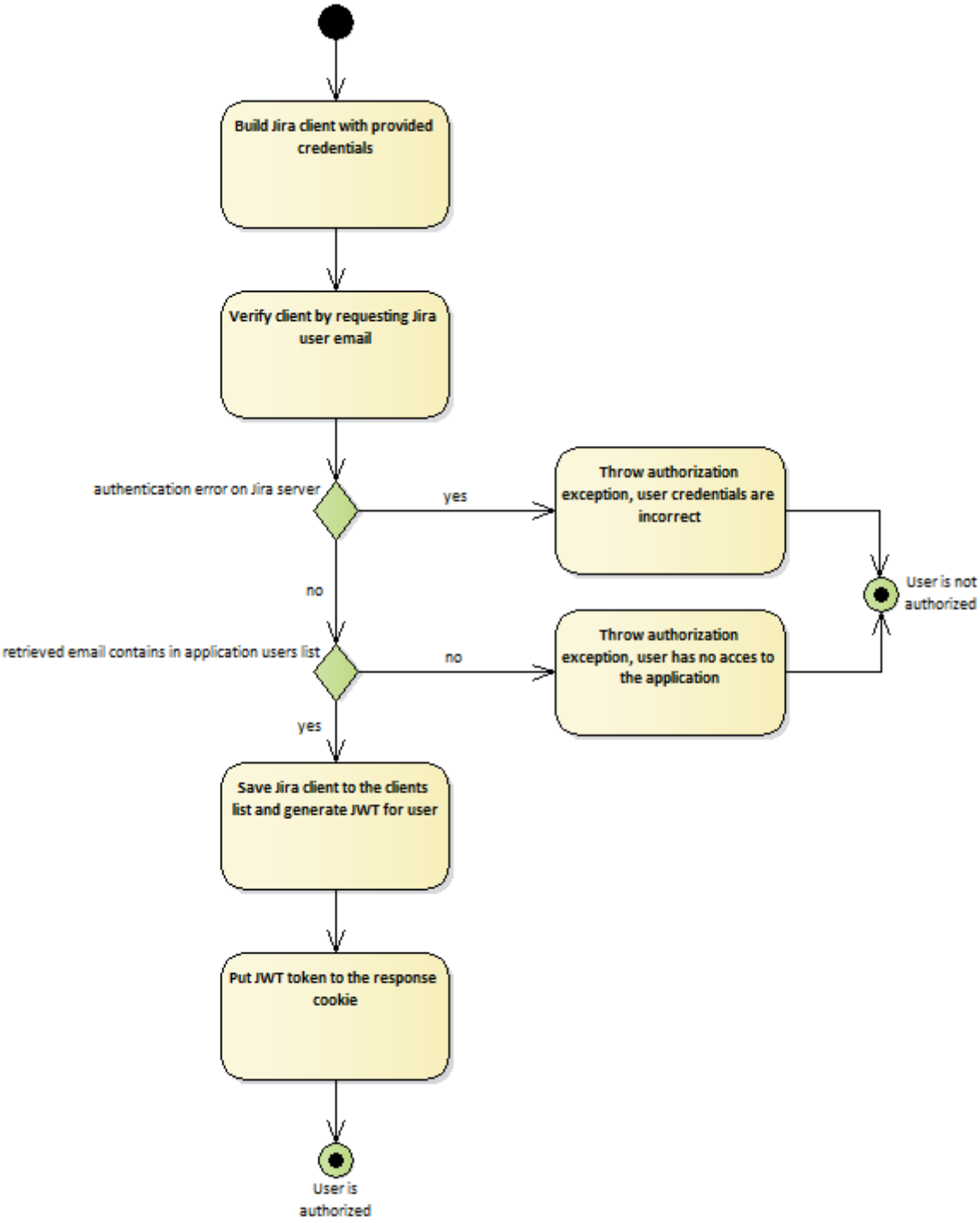

Figure 14. User authorization activity diagram.

Further, before each request, which requires authorization, the JWT is validated. If interaction with Jira is required, then Jira client is taken from the list by retrieved from the JWT user email. JWT does not contain user credentials such as username and

password, as they are stored in the Jira client object. The Jira client is used to make requests to the Jira using the basic authentication as was described above.

On logout, the JWT containing cookie is cleared and the corresponding Jira client is removed from the clients list.

5 Software quality assurance

This chapter provides a testing overview and a code metrics of the application back-end part.

5.1 Testing

To test application, have been created 171 unit tests and 1 integration test. The overall test coverage is 86%. The tests cover validation of entity models, authorization, API controller methods and all application services.

To test the whole application work, has been used a local database and a company Jira server with a real data.

5.2 Code metrics

The LOC metrics of the back-end part, excluding the test sources, are provided in Table 17.

Table 17. LOC metrics of the back-end excluding the test sources.

File type	**LOC**	**NCLOC**
Groovy	41	41
Java	1628	1564
Properties	35	31
Text	252	
XML	293	293
Total	**2249**	**1929**

The LOC metrics of the back-end part test sources are provided in Table 18.

Table 18. LOC metrics of the back-end test sources.

File type	**LOC**	**NCLOC**
Java	2257	2242

Thus, the total LOC of the back-end is 4506 and the NCLOC is 4171.

The complexity metrics such as essential complexity, module design complexity and cyclomatic complexity are provided in Table 19. The total number of back-end methods (test sources are not included) is 192; the table represents only minimum, maximum and average values among all these methods.

Table 19. Complexity metrics of the back-end methods.

	ev(G)	**iv(G)**	**v(G)**
Min	1.00	1.00	1.00
Max	4.00	5.00	6.00
Average	1.07	1.19	1.27

The cyclomatic complexity of each method is in range 1-6 and does not exceed the good complexity maximum value provided in article [8], this means that code is well structured and the code testability is high.

6 Conclusion

The goal of this thesis is to automate processes of creating software project reports, where the projects data should be acquired from Jira. During the application development, all required functionality has been implemented. The built application provides the following functionality:

1. Overview of the projects, sprint and issues;
2. Overview of the project, sprint and issue costs;
3. Project budget values, such as service rate and sponsored amount editing;
4. Overview of the sprint goals, sprint goals creation with possibility to set goal status, possibility to link goal with the issue to calculate goal cost;
5. Sprint goal values, such as status, description and linked issues editing;
6. Possibility to unpublish certain sprint report or the whole project;
7. Possibility to update certain sprint report or the whole project;
8. Survey sending and the survey results fetching;
9. Possibility to create and save custom project style;
10. User management.

According to the application requirements, the goal of this thesis is achieved. The value of the result is high as the application saves a lot of time on creating software development project reports and minimizes the possibility of human mistakes during the costs and rating averages calculation.

References

[1] Baeldung, “Baeldung | Java, Spring and Web Development tutorials,” [Online]. Available: https://www.baeldung.com/. [Accessed 14 5 2019].

[2] Mkyong, “Mkyong.com – Collection of Java web development tutorials, FAQs, and articles,” [Online]. Available: https://www.mkyong.com. [Accessed 15 5 2019].

[3] Atlassian, “REST APIs,” [Online]. Available: https://developer.atlassian.com/server/jira/platform/rest-apis/. [Accessed 2 5 2019].

[4] B. Carroll, “jira-client,” [Online]. Available: https://github.com/rcarz/jira-client. [Accessed 2 5 2019].

[5] Atlassian, “Basic authentication,” [Online]. Available: https://developer.atlassian.com/server/jira/platform/basic-authentication/. [Accessed 5 5 2019].

[6] Atlassian, “Cookie-based authentication,” [Online]. Available: https://developer.atlassian.com/server/jira/platform/cookie-based-authentication/. [Accessed 5 5 2019].

[7] Atlassian, “OAuth,” [Online]. Available: https://developer.atlassian.com/server/jira/platform/oauth/. [Accessed 5 5 2019].

[8] C. Bertrand, “Coding Concepts! Cyclomatic Complexity,” 17 9 2018. [Online]. Available: https://dev.to/designpuddle/coding-concepts---cyclomatic-complexity-3blk. [Accessed 19 5 2019].