Bachelor's thesis

Information and communication technology

NTIVIS14

2018

Tapio Mattila

BUILDING A COMPLETE FULL-STACK SOFTWARE DEVELOPMENT ENVIRONMENT

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology – Embedded software

2018 | 80

Instructor: Principal Lecturer, Jari-Pekka Paalassalo

Tapio Mattila

BUILDING A COMPLETE FULL-STACK SOFTWARE DEVELOPMENT ENVIRONMENT

The purpose of the thesis was to explore and learn the tools and environments used in Java EE full-stack software development. This was achieved by building a simple web-application, which used all of these elements. The main focus was not in the functionality of the application, but creating a suitable architecture and integrating all of the elements to work together. The architecture should be designed so that it would be possible to build large applications using the same platform. For example, databases should be created in a way that they could be scaled to manage millions of documents without noticeable difference in usability.

The elements of the software architecture include: a service-oriented-architecture built web-application, version control tools used to manage source code, an automated testing environment, continuous integration and deployment tools, a distributed build tool and a project management tool that include a ticket system. All the tools used in the thesis are known and largely used in software development field that the software developers use daily in the working field.

As a result of this thesis a simple web-application was built, and a full-stack development environment was built and integrated to work with the application. Creating the environment and connecting the elements to work with each other provides great knowledge and understanding of the whole system. This task is beneficial to any developer who want to enhance their skills. The last chapter consists of a complete walk-through of the building process so building the application and the environment is fully repeatable with the material of this thesis.

KEYWORDS:

Service-oriented-architecture, Java, Spring, Jenkins, JIRA, Vaadin, Git, Maven

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikan koulutusohjelma – Sulautetut ohjelmistot

2018 | 80

Ohjaaja: Yliopettaja, Jari-Pekka Paalassalo

Tapio Mattila

KOKONAISVALTAISEN FULL-STACK-OHJELMISTOTUOTANTOYMPÄRISTÖN RAKENTAMINEN

Opinnäytetyön aiheena oli tutustua ja oppia käyttämään Java EE full-stack-ohjelmistokehityksessä käytettäviä työkaluja ja ympäristöjä. Tämä toteutettiin rakentamalla yksinkertainen verkkosovellus, joka käyttää kaikkia näitä elementtejä. Sovelluksen toiminta oli opinnäytetyössä sivuasia. Pääkeskittyminen tapahtui oikeanlaisen arkkitehtuurin luomiseen ja kaikkien elementtien keskinäiseen integroimiseen. Arkkitehtuuri ja elementtien integrointi piti tehdä niin, että alusta soveltuisi käytettäväksi sellaisenaan suurien projektien alustaksi. Esimerkiksi tietokannat piti luoda niin, että niitä pystyttäisiin skaalaamaan miljooniin dokumentteihin siten, että käytettävyys ei kärsisi.

Arkkitehtuurin elementtejä ovat muun muassa palvelukeskeisellä arkkitehtuurilla luotu verkkosovellus, versionhallintajärjestelmä, automaattinen testausympäristö, automaattinen palvelimelle vienti, käännöspalvelin ja projektihallintaohjelma, joka sisältää tikettijärjestelmän. Kaikki opinnäytetyössä käytetyt työkalut ovat ohjelmistojen kehittämisessä käytettyjä, laajasti tunnettuja työkaluja, joita ohjelmistokehittäjät käyttävät päivittäin.

Opinnäytetyön tuloksena syntyi yksinkertainen verkkosovellus ja sen ympärille integroitu tuotantokehitysympäristö. Ympäristön rakentaminen ja kaikkien elementtien toisiinsa liittäminen siten, että ne toimivat hyvin yhteen antaa erinomaisen tietämyksen ja ymmärryksen koko systeemistä. Tämä rakennusprosessi on hyödyllinen kenelle tahansa ohjelmistokehittäjälle, joka haluaa tehostaa taitojaan. Opinnäytetyön kokeellisessa osassa verkkosovelluksen ja tuotantoympäristön rakennus on kerrottu vaihe vaiheelta ja koko työ on täysin toistettavissa seuraamalla rakennusprosessia.

ASIASANAT:

Palvelukeskeinen arkkitehtuuri, Java, Spring, Jenkins, JIRA, Vaadin, Git, Maven

CONTENT

LIST OF ABBREVIATIONS 7

1 INTRODUCTION 1

2 SOFTWARE DEVELOPMENT LIFECYCLE 3

3 ELEMENTS OF SOFTWARE DEVELOPMENT ENVIRONMENT 6

3.1 Source code 6

3.2 Artifactory repository 6

3.3 Client-server model 6

3.4 Databases 8

3.5 Source code management/version control 8

3.6 Continuous Integration and continuous delivery CI/CD 9

3.7 Project management tool 10

3.8 Other tools 11

4 TOOLS USED IN BUILDING THE ENVIRONMENT 12

4.1 Java 12

4.2 Eclipse 12

4.3 Maven 13

4.4 Java EE 14

4.5 Frameworks 14

4.5.1 Spring 15

4.5.2 Spring Boot 15

4.5.3 Vaadin 16

4.6 GIT 17

4.7 Tomcat 18

4.8 Jenkins 18

4.8.1 Continuous integration 19

4.8.2 Continuous delivery/deployment 19

4.8.3 Distributed builds 19

4.9 JIRA 20

4.9.1 Fisheye 21

4.9.2 Crucible 22

4.10 MongoDB 22
4.10.1 Replication 23
4.10.2 Sharding 24
4.10.3 Authentication 25

5 BUILDING THE ENVIRONMENT 27
5.1 Introduction 27
5.2 Start with planning 28
5.3 Start coding 29
5.3.1 Installing and configuring necessary tools 29
5.3.2 Configuring Eclipse IDE 32
5.3.3 Simple spring boot hello world application 33
5.3.4 Committing work to GIT 39
5.3.5 Creating Vaadin UI 40
5.4 Adding logs 46
5.5 Adding database to the project 49
5.6 Unit testing 53
5.7 Adding replica sets and sharding to databases 56
5.8 Connecting Jenkins 64
5.9 Connecting JIRA 72
5.10 How it all work together? 77

6 CONCLUSION OF THE PROJECT AND FINAL THOUGHTS 79

REFERENCES 80

APPENDICES

Appendix 1. Source code of the date-storage application
Appendix 2. Environmental variable scripts
Appendix 3. MongoDB database scripts
Appendix 4. Configuring Jenkins

FIGURES

Figure 1. One type of software development lifecycle 3

Figure 2. Example Jenkins master/slave architecture 20
Figure 3. Kanban board example 1 21
Figure 4. Kanban board example 2 21
Figure 5. Relational database VS. non-relational database 22
Figure 6. Basic structure of a single replica set with 3 members 23
Figure 7. Basic sharded collection structure with 3 shards and a config server 25
Figure 8. Sequence diagram of the project functionality 27
Figure 9. Testing environment variables for the tools that are used in the project 32
Figure 10. Basic hello world spring boot application pom.xml file 34
Figure 11. Project layout for the basic spring boot hello world application 36
Figure 12. Application class for the basic hello world spring boot application 36
Figure 13. Hello class for controlling the output of the application 37
Figure 14. Result of the basic hello world spring boot application 38
Figure 15. Changing the project to work with Vaadin 8 41
Figure 16. MainUI class Vaadin annotations and layout components 43
Figure 17. Date-project front-end project structure 43
Figure 18. MainUI class init method that contains UI structure of the application 45
Figure 19. Front-end of the date-project 46
Figure 20. Log4j2.xml file, three main parts, properties, appenders and loggers 48
Figure 21. Result of quering data in datedb, dates collection 51
Figure 22. Project structure after the initial test for mongodb 51
Figure 23. Final application layer presentation 53
Figure 24. Adding JUnit dependency to date-project application 54
Figure 25. Initializing DateToRepositoryTest to be JUnit test case 55
Figure 26. Test case for saving dates to repository 55
Figure 27. Final project structure after the test part is included 56
Figure 28. Example of the replica set initialization process with shard a 60
Figure 29. Creating root user for the database 61
Figure 30. Creating cluster admin user for config server 61
Figure 31. Enable sharding for database and collection 62
Figure 32. Adding test users to test sharded collection chunk distribution 63
Figure 33. Complete application.properties file for date-project application 63
Figure 34. MongoDB sharded collection structure for application database 64
Figure 35. Changing port numbers in server.xml file 65
Figure 36. Maven build setup in date-storage-application Jenkins build 67
Figure 37. Users and roles inside tomcat-users.xml 68
Figure 38. Deploy to container configurations 69
Figure 39. Date-storage-application before deploying project to production server 71
Figure 40. Slave executor added in the executor list 72
Figure 41. Git integration for JIRA in the attlassian marketplace 73
Figure 42. Creating user in fisheye 75
Figure 43. JIRA issue status changed with git commit message 77
Figure 44. JIRA smart commit showing the changes that were made to the code 77

LIST OF ABBREVIATIONS

BOM	Bill Of Materials
CI	Continuous Integration
CRON	A software utility CRON is a time-based job scheduler in Unix-like computer operating systems (Wikipedia 2017)
CRUD	Acronym for create, read, update, delete
DAO	Data Access Object
DBMS	Database Management System
GUI	Graphical User Interface
HDD	Hard Disk Drive
IDE	Integrated development environment used to create source code for the application
JAR	Java Archive file, package file format that is used to aggregate Java files into to one file for distribution (Wikipedia 2017)
JDBC	Java Database Connection
JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtual Machine.
OSI	Open Systems Interconnection model
POM	Project Object Model
Replica set	Replicating the initial MongoDB database to primary and secondary members
RAM	Random Access Memory
SDLC	Software Development Lifecycle

Sharding	Horizontally scaling MongoDB database
SQL	Structured Query Language
SSD	Solid State Drive
WAR	Web Application Archive
XML	Extensible Markup Language
UI	User Interface
URL	Uniform Resource Locator

1 INTRODUCTION

When creating web applications in software development there are lot more to be considered than just creating the source code for the program. This is obviously very important part, but for more than one developer being able to work with the application creation in the real-world situation there are several things that needs to be thought of. The environment where developers are working should include some kind of shared repository where the code is pushed and stored. There should also be a ticket system for project management purposes. Automated testing and deployment would also be very good thing to have as a part of the environment. All of these things and more are in use of the todays developers that are building software's and applications for people to enjoy.

Purpose of this thesis was to explore and learn different tools used in software development and to examine how different elements of the environment connect to each other. This was done by first creating a simple web application using Java EE framework solutions and MongoDB database. After the demo application was built, then a right kind of an environment was built around the application that uses most of the elements that are in use in the development field.

The demo application that was built, was a simple button in a browser that saves the date and the time of the button press in the database. The database was created to be scalable and reliable for the demo application to be able to be refined in a big and complex application in the future. The git repository was connected to the demo application and automated testing tool and project management tool were connected to the project. Chapters 4 and 5 dives more in to details of the demo application and environment structure.

Security of a web application is an essential part of any application that is in production and in public use. In the time of growing industry of web-based applications and cloud, protecting the application against malicious users and different kind of attacks should be a top priority. That being said, the security part in building the development environment and demo application was left out from this thesis. In database creation authentication was used, and in creating the environment only trusted tools were used, but further dive into security aspect of these tools was not made. Passwords used in this project with

those tools are not strong enough in production version environment and any user re-creating the environment should use stronger passwords.

The assignment for this thesis came from Trivore Ltd., a company in Turku that specializes in software integrations and application developments. Trivore is using software development environment in their own works to be able to produce applications and software's with its developer team.

Thesis starts with describing the elements in full-stack software development lifecycle. Then the focus moves on telling what are the main parts of the environment and showing different options of tools and what are the characteristic of these tools. In chapter four the focus is on the tools that were used in building the demo application and environment around the application. In this chapter tools that were used were described in more detail. Chapter five is a walk-through on how the environment was built and in the last chapter the process is reviewed.

Thesis was written in the way that building the environment is fully repeatable with the information included in the document and appendices. The operating system that was used to build this environment was Windows 10 64-bit.

2 SOFTWARE DEVELOPMENT LIFECYCLE

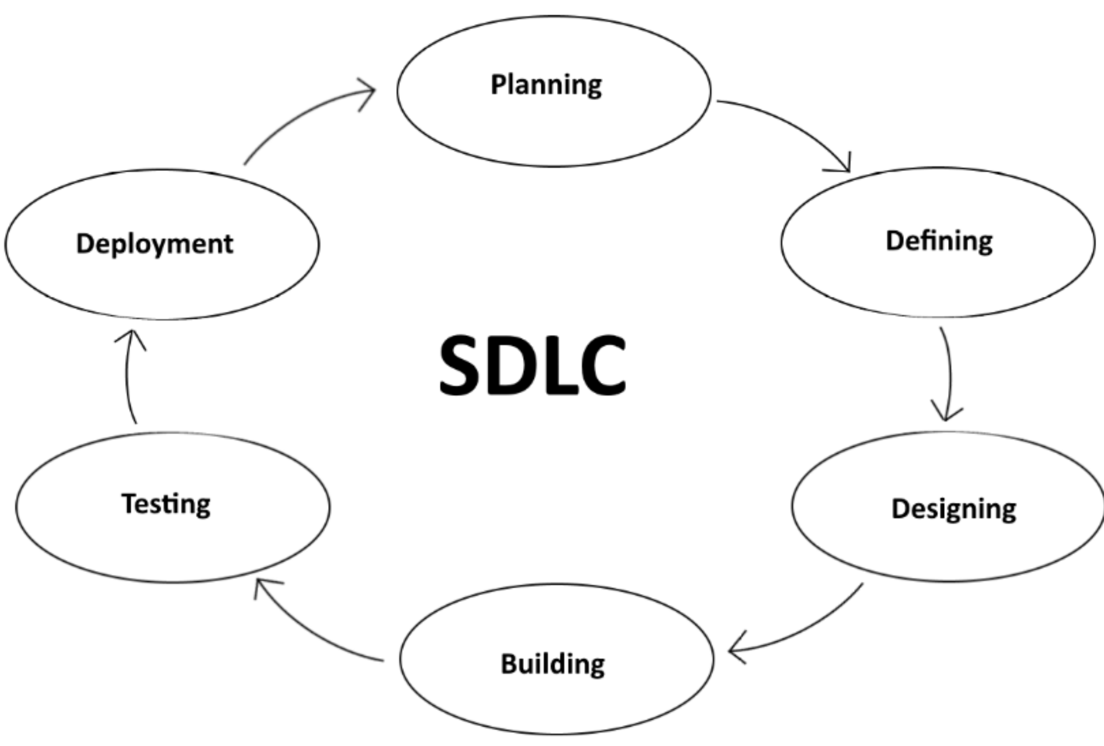

Figure 1. One type of software development lifecycle

Before diving into the tools used to build the full-stack development environment, it is helpful to think about the process of creating a web-based product into production and different phases it includes.

Software development lifecycle is a typical way of presenting the lifecycle of a software from the idea to a complete product. There are many different versions of the software development lifecycle depending on the type of software being made. Fitting example for the full-stack web application build process consists of six main stages; planning, defining, designing, building, testing, deployment. This example seen in figure 1. is mainly used in iterative and agile models. (SDLC overview, tutorialspoint 2017. Systems development lifecycle, Wikipedia 2018)

In the planning stage the idea is refined to more contextual thing. The plan is answering a lot of initial questions about the product that is going to be build. What the product is

going to do, what are the initial requirements for the product, what are the key points of the product.

The defining stage goes more deeply on the product on how the requirements are going to be filled and are some requirements needed to be left out.

In the designing stage the product plan is worked out on the design level and all the elements that are valid in to the project are going to be defined and thought. The architecture of the product is thought and the more final sketch of what the product is going to be looking as a final form is determined. In the design stage all the parts of the product are thought so in the building stage there is little to none things that are not thought of.

The next stage is the building stage where the product is built to its final form by the frames that are defined in the earlier stages.

After the product is build, the product is going through a series of tests to make sure that the product meets the demands of the initial requirements and planning.

The last stage is the deployment stage where the product is deployed in to the production.

This is the rough frame of building a product. In reality this model is refined in to several different models depending on the project in hand. Usually tests are performed in between the building stages and throughout the lifecycle.

It is very important to have some kind of a model, on how the product is going to be built from an idea to a final product. Most important SDLC models are:

- Waterfall model
- Iterative model
- Spiral model
- V – model
- Agile - model

In waterfall model each phase of the SDLC circle needs to be completed, before advancing to the next phase. With waterfall model requirements of the hole project needs to be clear and complete before starting to build the software. The waterfall model phases are conception, initiation, analysis, design, construction, testing, deployment and

maintenance. This model is very strict and is used for example in manufacturing and construction industries where early detection of software bugs or other flaws is exponentially less expensive than finding them in other stages of the project. (SDLC waterfall model, tutorialspoint 2017. Waterfall model, Wikipedia 2017)

Iterative model splits the application creation process to small pieces that are repeated throughout the software creation. These steps can be design, development, testing and implementation. With every repeated cycle, application takes more to its final form until it is finally complete. Iterative model does not need for the product plan to be complete and can be started as soon as small part of the plan is ready. Iterative model is more flexible than waterfall model and is used a lot for example in web sites and web applications where customer feedback can be had easily. (SDLC iterative model, tutorialspoint 2017. Iterative design, Wikipedia 2018)

Spiral model is a combination of waterfall model and iterative model. In spiral model there are four phases that are repeated through the project work. These phases are identification, design, building and evaluation/risk analysis. These phases are repeated until the product is complete. Spiral model is an incremental risk reduction model where each iteration the risks are evaluated. This lowers the risk that the project is going to fail. (SDLC spiral model, tutorialspoint 2017. Spiral model, Wikipedia 2017)

V-model can be considered to be an extension of a waterfall model. After moving down like a waterfall model, the cycle takes an upward motion in the form of validation and testing. V – model phases can also be performed as a parallel with other phases. (V – model (software development), Wikipedia 2018)

Agile-model or better known as agile software development is an approach in software development with a set of values and principles. The manifesto of agile software development is stating these principles as a set of guidelines when working in an agile environment. Customer satisfactory by early and continuous delivery of valuable software, welcoming change and frequent delivery of working software. Close cooperation's between different parties, building project around motivated individuals, preferring face-to-face communication and measuring progress with a working software. Sustainable development, continuous attention to technical excellence and good design, self-organizing team and teams reflecting regularly how to become more effective. (Agile software development, Wikipedia 2018)

3 ELEMENTS OF SOFTWARE DEVELOPMENT ENVIRONMENT

3.1 Source code

Source code of the project is working as a bone structure of the software or an application. Inside the source code all the functionality of the program is defined and made. There are lots of different architectures inside the source code of how the program is made depending on the final target of the application. Also, the programming language is depending on the target platform of the application. Not all programming languages can perform on every situation. In this thesis the programming language is Java and the target application is a web application.

3.2 Artifactory repository

Artifactory repository also called as binary repository is a centralized place where the dependencies that the project needs to be build are stored. The dependencies can be made by the company itself for it to use the same dependencies in another project or the dependencies can be produced by a third party to provide dependencies that are needed for their services to function. The main purpose of the artifactory repository is to make developers life easy with managing different dependencies and packaging the final application in a format that can be deployed to server. (Binary repository manager, Wikipedia 2017)

At this project a software called Maven is used. It is the most popular build automation tool for Java and provides an easy to manage environment for managing the dependencies and configurations of the project.

3.3 Client-server model

What is a server? In computing, a server is a computer program or a device that provides functionality for other programs or devices called “clients” (Server computing, Wikipedia 2017). The physical server is usually a dedicated machine that provides the functionality that it is configured to do for the clients that uses the server, but it does not have to be. Server can also be a laptop or a local computer that is configured to provide services for

other computers. Client/server model is the most frequently used model and is also called a request/response model. (Server computing, Wikipedia 2017)

In a client-server model, server is providing functions or services to one or many clients, which are requesting these services. Servers are named by the services they provide. Different server models are described in the next paragraph. Client do not concern how the server is performing the response as long as it fulfills the request sent by the client. The client then only needs to understand the response sent by the server. All client-server protocols operate in application layer of the OSI model. (Client-server model, Wikipedia 2018)

There are different server models for different purposes of use. For example, a client/server model server can be a web server. Web browser in the local machine that is requesting the information from the internet is the client and web server is the machine that provides the information or files that matches the HTTP request made by the user. (Server definition, Whatis.com 2017)

Some of the server models are:

- Web server
- Application server
- Proxy server
- File server

Web server follows the structure explained above. Application server is a complete platform that consist both the service environment to run the application and also facilities to create the application. Proxy server is a server that is handling client's requests to seek other servers. File server is used to store files and share them with other computers in the network. (Application server, proxy server, file server, Wikipedia 2017)

In this project work no external server was used, and the traffic was passed through local machine with different ports. The name for this is a localhost. Also, all the data was stored locally in the running machine. The server types that would be used for this project are web server and file server. The artifactory repository dependency files and documents that are stored inside a database would be stored in an external server.

3.4 Databases

Databases are the way of storing and organizing data that is produced from the use of the application of software. Databases are essential part of any program since they provide a way to persist the data for later use. This could be done for a number of reasons such as storing user details for an account details, storing player scores for a game, storing logging data from application use and many more. (Database, Wikipedia 2017)

There are two main database management systems (DBMS). Relational databases and non-relational databases. Most notable relational databases are: MySQL, PostgreSQL, SQLite and MariaDB. With non-relational databases: MongoDB and CouchDB. The difference of these two database systems is the way the data is stored.

In relational database the data is stored in a predefined schema that consists of rows and columns. The set of rows and columns are called as tables. All the relational databases use structured query language (SQL) for manipulating the data inside the database. The terminology that is used to manipulate the data is called CRUD. CRUD is an acronym of the words, create, read, update and delete. (Relational database management system, Wikipedia 2017. Definition of relational database, WhatIs.com 2006)

Non-relational databases are schemaless, so they are more flexible in use. They use collections of data that can be stored as JSON (JavaScript Object Notation) format. Non-relational databases are very good at storing data in changing situations, because non-relational databases are easy to update, and the schema of non-relational database is changing as the new data is added without separate instructions for database. Relational databases are great if the database is going through a lot of complicated querying (searches) and if the database schema is not going to change (a lot, often). (Relational vs. non-relational database, Pluralsight 2014)

At this project a non-relational database called MongoDB is used. MongoDB is the next big thing in the database circle and is very flexible and scalable database option.

3.5 Source code management/version control

Source code management is one of the most important thing in the software development. It is used to manage the source code, make branches from the master branch that enables a team to work with a same code, review the changes that have

been made in the code and to keep track who have done these changes. Without version control it would be almost impossible for team of developers to make changes in the same code base in sync without conflicts in the code. With version control it is always possible to revert the code base to an old stage if for example the change that was made corrupted the code in any way and the old stage before the change needed to be redeemed, because the source code is saved in the repository.

There are different version control systems with different architectures. The dividing is made between centralized systems and decentralized systems. In centralized system the version control system, developers use single shared repository. As in decentralized system the entire copy of the history of the repository is kept in each developer's local machine. Examples for centralized version control systems are Subversion and Perforge and examples for decentralized version control systems are Git and Mercurial. (Version control, Wikipedia 2017. Version control systems distributed vs centralized, Oshyn blog 2012)

The most used version control system is Git and it has kept the dominant position for numbers of years. This is also the version control system that is used in this thesis work.

3.6 Continuous Integration and continuous delivery CI/CD

Continuous integration is a practice of developers committing code changes frequently usually ones a day making the changes to the mainline happen several times a day. With every commit to the repository, an automated testing is performed to make sure these changes did not broke anything. With this process the mainline stays clean and is easily released to production version if needed. (Jenkins Master CI for DevOps and Developers, Udemy 2017.)

Continuous delivery and continuous deployment are processes of automatically deploying application to the staging server and production server. This process is done after all the testing phases are passed without any conflicts. The continuous deployment relies heavily on reliable tests that are run with every build when developers are committing code.

Different continuous integration tools include Jenkins, TeamCity, Bamboo, CircleCI and more. Jenkins an open source free continuous integration tools that is offering very wide range of plugins that makes it compatible to lot of different scenarios. TeamCity is a CI

tool from company called JetBrains. The free version of TeamCity is allowing use of all the features, but is limiting the extensibility of the CI for 20 configurations. Bamboo is a continuous integration tool from a big Australian company called Atlassian. Bamboo is free to use for 30 days and after that paid plans are necessary. Bamboo is best used with Atlassian's own repository called Bitbucket although it is also compatible with other repositories like GitHub. (Top 8 continuous integration tools, code-maze.com 2016)

Jenkins is the choice for continuous integration tool for this thesis work because it is free and very popular CI tool that is suits well for this project.

3.7 Project management tool

Project management tool is a tool that is used in the project to organize and handle different changes, bug fixes or any work that needs to be done. Project manager needs to be able to handle who is doing what and see the work flow where things are going in any given time. This has a major role in productivity of the development team. Different issues regarding of the changes that needs to be made and what everyone is doing is also visible to each development team member.

There are different kind of project management tools available and not all of them provide the functionality that is suitable for any project or any environment. The task of finding a project management tool that is suitable depends on the needs of the project.

Some project management tools that can handle a wide variety of projects and requirements are JIRA, Asana, Redmine and Slack. (Project management software, Capterra.com 2018)

The project management tool that is used in this project is called JIRA. JIRA is a project management tool from a company called Atlassian. It provides a simple dashboard that lets the user to track the process of the project with one glance. Issues can be created for different priorities and issues can be targeted to any member of the project. With a Kanban board, users can see the progress of the project with simple interface that lets the user to drag and drop different tasks to different states like not started, in progress and done. JIRA is very suitable for the project management tool to work with this thesis project.

3.8 Other tools

There are also a several other tools that are used in web development environment, but are not covered in this chapter in detail and are not implemented in the thesis project.

Testing is one significant part of development environment and there are wide variety of testing tools available. Web application testing tools falls under several categories. These categories are cross-browser testing, web site security testing, link manager testing, web functional/regression testing and load, stress and performance testing. In addition to these there are couple W3C validators that are used to determine that the website is functioning properly. W3C CSS validator and W3C Link Checker. (Most popular web application testing tools, Softwaretestinghelp.com 2017)

Examples for testing tools in each category:

- Browsera, cross-browser testing
- NTOSpider, website security testing
- LinkTiger, link manager testing
- Webload, load, stress and performance testing
- Selenium, web functional/regression testing

In the thesis project, no external testing platform is used, but the testing is made with unit testing the code base.

Requirement management is also a part of development environment. Requirement management is a process of documenting, analyzing, tracing and prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders. (Requirement management, Wikipedia 2018). JIRA can be used for requirements management purposes in conjunction with Confluence that is content collaboration tool by Atlassian. (Using JIRA for requirements management, Confluence.atlassian.com 2017)

4 TOOLS USED IN BUILDING THE ENVIRONMENT

Building the development environment uses lots of different software development tools that are described in this chapter. This consist of Java, the chosen programming language for building the test application. Helping the application creation frameworks like Spring and Vaadin are used. Source code is created in Eclipse IDE and Maven is used to help importing different dependencies and to organize the project. Git repository is used to hold the source code and Jenkins and JIRA are used to handle the automated testing and ticket system. MongoDB is used to persist the data for later usage. These components are explained and described in this chapter.

4.1 Java

Java is a high-level programming language that was created in 1995 by Sun Microsystems and was later acquired by Oracle Corporation. It is a programming language that is intended to “write once, run anywhere”. Java is doing this with JVM (Java Virtual Machine), that is a virtual machine that runs the Java code inside the given machine. Because Java does not rely on the platform specific things and run its own platform in the form of virtual machine, it is portable to any platform to run. Java is very popular programming language in terms of job need and usability. It is used as a programming language in variety of information technology fields such as the android platform and web applications in enterprise field. (Java programming language, Wikipedia 2017)

4.2 Eclipse

Eclipse IDE (Integrated development environment) is a tool which is used to create, compile and test source code. It is the most popular IDE for creating Java applications. Other notable IDE’s that are used for creating Java applications are IntelliJ and NetBeans. Eclipse IDE is free and open source and there are different developer packages for Java, C++, PHP, JavaScript and more. Updates and new versions are constantly released at regular intervals, the latest version being Oxygen. (Eclipse, Wikipedia 2017)

4.3 Maven

Apache Maven is a software project management and comprehension tool that is used for describing how the software is built and what dependencies it uses. These are described inside the pom.xml file (Project Object Model). Maven makes it easy for creating Java application by simplifying the configuration process. It is mainly used for creating applications in Java programming language, but there are plugins and addons for another programming languages as well, although the support is minimal and in reality, other languages are rarely used. (Description for Maven, Apache Maven main page 2017. Apache Maven, Wikipedia 2017)

In maven pom.xml file the description of the software is made and then dependencies, properties and other software related information's are given to the pom.xml file. Maven then imports those dependencies from the maven remote repository to the local machine. This collection of JAR files that is imported in the machine is called artifactory repository. Before maven all the files that needed to be included in the project for it to be working correctly needed to be imported manually. This was often very tedious process and maven makes it easy. When adding some dependencies like spring boot starter dependency, the one dependency that is added to the project is adding its own dependencies with no additional commands or work. These dependencies are called transitive dependencies.

Maven is using a maven lifecycle steps to build the project. The steps are:

- validate
- compile
- test
- package
- verify
- install
- deploy

Validate, validates the project and checks that all the needed information is available. Compile compiles the source code of the project. Test tests the compiled code with a suitable unit testing framework, usually JUnit framework. Package is packaging the project to distributable format, such as JAR. Verify checks the result of the tests to make sure the criteria's are met. Install is installing the package to the local repository in the

local machine, for it to be used in other projects. And deploy copies the final distributable package to the remote repository where it can be shared with other projects. These commands are the core of maven and are the same kind with any project that uses maven. (Introduction to the build lifecycle, Apache Maven introduction page 2017)

When building the project with maven lifecycle commands, all the lifecycle commands before the one that is given are run and then the command that is given will run. There are also default lifecycle commands that are run without the need of giving the explicit command. These are default, clean and site. Default is the lifecycle command that handles the project deployment. Clean command is cleaning the target folder of the project and deleting old files. Site is handling the site documentation of the project. For example, a command **clean package** is cleaning the project target folder for any old files for the project to start with clean solution and then package the project in a distributable format.

4.4 Java EE

Java Platform Enterprise Edition, is a set of specifications, extending Java SE (standard edition) with specifications for enterprise features such as distributed builds and web services (Java EE, Wikipedia). Java EE provides developers means to deploy web applications and web services with java server pages, web servlets, sockets and applets. There are lot of Java APIs to comply with different scenarios of usage. Java Enterprise Edition is mainly used nowadays with frameworks like Spring to make the Java programming more dynamic and easy. (Java platform enterprise edition, wikipedia 2017.)

4.5 Frameworks

Different frameworks help the developer to achieve the goal with lesser typing and with less work on the configurations. Java EE is fairly complex by itself when creating applications, and frameworks like Spring, Spring Boot and Vaadin are there to simplify the process. With these frameworks, developers can focus more on the business logic and less on the front-end and different configurations. (Application framework, Wikipedia 2017. Software framework, Wikipedia 2017.)

4.5.1 Spring

Spring framework came to existence in 2003 as a response of the complexities of J2EE that was the second release of Java enterprise edition. Creating applications with pure Java EE was a struggle and there was a lot of unnecessary code to clutter the application. Spring was created to resolve this issue with a sole mission of simplifying Java development. At its core spring is a dependency injection framework. This means the spring is handling the dependencies of different classes and creating a loosely coupled instances of the different methods that classes are using. For example, if the connection in the database was needed to be made with old java EE way, the JDBC (Java Database Connection) was instantiated inside the class that was using it. With spring there is no need for direct dependency like this and the database connection can be separated from the class that is using it. (Simplifying Java Development, Spring in Action 4th Edition 2014. Quick introduction to spring framework, Udemy 2017)

The configurations and connections for these components are done with XML. Even though spring is more lightweight compared to the plain Java and using Enterprise JavaBeans, the configurations are fairly complex. The use of XML files is extensive.

4.5.2 Spring Boot

Spring Boot is the latest version of Spring framework. With spring boot, it is very easy to make a web application and test the work in the browser. In spring boot there is need for very little spring configurations and applications can be deployed with no xml files. Spring boot uses “convention over configuration” model that means that developers using the framework are required to make fewer decisions without losing the flexibility of the system. Nowadays applications that are built with spring, are almost always using also the features of spring boot. (Quick introduction to spring framework, Udemy 2017)

There are some key concepts and terminology used in spring boot that are the core of the framework.

- Bean
- Autowiring
- Dependency injection
- Inversion of control
- IOC container

- Application context

Beans are different objects that are managed with spring framework.

Autowiring is the process of spring identifying the dependencies, identifying matches for those dependencies and populating them.

Dependency injection is injecting some class as a dependency for another class.

Inversion of control is a process of which the framework is taking control of the objects that would otherwise be controlled with a custom code made by the developer.

IOC container stands for an inversion of control container that is a generic expression of anything that is using the inversion of control.

Application context is the most important part of spring boot. It is the one where all the beans are created and managed and where all the core logic of spring framework happens.

Spring boot is extremely popular framework currently in Java development and that is based on the few things that spring boot framework is.

- It enables writing of testable code
- When using dependency injection, the testing is very easy
- Very good support for testing frameworks like Mockito and JUnit
- No plumping code (no unnecessary code to clutter the program)
- Flexible architecture
- Stayed very current, spring is updated regularly

4.5.3 Vaadin

Vaadin is a Finnish company that was founded in 2000. It has over 130 employees and its headquarter is located in Turku Finland. It also has offices in California and Germany.

Vaadin framework is a Java web application development framework that is designed to make creation and maintenance of high quality web-based user interfaces easy (Book of Vaadin 2017 chapter 1.1).

Vaadin is divided in server-side programming model and client-side programming model. It is using pure Java to make front-ends and server-side functionalities on the

applications so there is no need for using web languages like HTML and JavaScript, because Vaadin is translating Java code to browser languages as needed. It is highly extensible and works seamlessly with spring framework. The basic idea for Vaadin is to make the front-end coding easy and let developers to focus more on the business logic of the application.

Vaadin framework was an obvious choice for the front-end creation for this application since it was a familiar framework and very suitable for this project.

4.6 GIT

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. (Git main page 2017.)

Git is a version control system that was created by Linus Torvalds in 2005. Linus Torvalds is also the creator of Linux Kernel.

With Git developers can commit their changes on the repository, build a same project simultaneously with branches and have backup for their code in the repository for every commit made. Git is mainly used to track the changes in files inside the git repository. Most popular git repositories are GitHub and Bitbucket. Git repository is a repository that is used for projects that uses Git. (Git, Wikipedia 2017)

The way git works in the simple project example would be as follows. First there is a main project, the root project. Then developer is starting to work on a problem in the project and he/she makes a new branch from the root project. This branch is a clone of the root project at the time the branch is made and now the developer can make changes to the branched project and test those changes without the root project being compromised. When the problem is solved, the branched project is connected to the root project and the root project is getting the changes that were made in the branched project. This is a very simplified example about the workings of git. The main goal of git is to be able to monitor the changes that are made in the source code, have a backup of those source codes (commits) and being able to revert the changes if needed to get back the old working code (if something is broken for example). Also, every commit that is made, is made with a named user. So, any changes that were made can be tracked to the committer of the changes if necessary.

4.7 Tomcat

Apache Tomcat, often called as Tomcat Server is an open source Java Servlet Container developed by Apache Software Foundation. Apache Tomcat is used for a Java application container. Tomcat offers a web server environment and implements several Java EE specifications like Java Servlets and Java Server Pages (JSP).

Tomcat provides flexible, secure and easy to use container environment for Java applications that is very straight forward to configure with xml files. (Apache Tomcat, Wikipedia 2017.)

4.8 Jenkins

Jenkins is an open source automation tool that is used to automate several non-human tasks in the software creation. Continuous integration, continuous delivery and continuous deployment are the main parts of Jenkins. (Jenkins software, Wikipedia 2017.)

Simple example of a real-life use case. Developer push the code changes in to a code repository like GitHub. Build is triggered automatically in Jenkins and all the tests coded in the application are run. If the build fails because some tests were not successful, development team is notified instantly and any flaws in the code can be corrected immediately. If Jenkins is holding a clean build that is ready for use, it is deployed in to staging server automatically often with a time triggered deployment in the night time.

Jenkins is a continuous integration tool that was started as a name Hudson. It was started as a hobby project of Koshuke Kawaguchi, an employee of a Sun Microsystems in 2004 and was first used as a continuous integration tool for Sun Microsystems. It was released to public use at 2005 and other companies started using it as well as their continuous integration tool. It quickly became very popular tool in software development field and Koshuke was asked to be work full time with a Hudson project. By 2010 it became the most used continuous integration tool in the software development field with over 70% market share. In 2011, Sun Microsystems was acquired by Oracle and the Hudson name was changed to Jenkins. (Master Jenkins CI for DevOps and Developers, Udemy 2017.)

Jenkins is providing a big quantity of different plugins that are used for testing, security, code analyze, mailing functionality, visual presentation and more. One of the most used plugin is the pipeline plugin that is used for presenting the build flow as a visual pipeline presentation.

4.8.1 Continuous integration

Continuous integration means that the code is constantly merged with the main branch of the development work and automated testing is performed. Continuous integration is highly a mindset for the development team. Any build failures should be treated as a high-priority and the code commits should happen frequently. Team should also be committed to do high-quality tests, so the product quality be as high as possible. This is speeding up the release frequency for the product to the production when the tests are handled automatically, and any build failures are fixed immediately. Jenkins uses CRON syntax in the continuous integration that is monitoring the repository changes to trigger a build. (Master Jenkins CI for DevOps and Developers, Udemy 2017.)

4.8.2 Continuous delivery/deployment

Continuous delivery means that the code is automatically shipped in to the staging server as soon as the code is tested and analyzed. In the staging server the code is reviewed in the way that unit testing is not able to do. Unit testing is not able to test the design and hands-on workings of the product, just business logic and possible bugs. Continuous deployment means that the deployment on the production server is also automated. (Master Jenkins CI for DevOps and Developers, Udemy 2017.)

4.8.3 Distributed builds

Jenkins supports the “master/slave” mode, where the workload of building projects are delegated to multiple “slave” nodes, allowing a single Jenkins installation to host a large number of projects, or to provide different environments needed for builds/tests. (Distributed builds, Jenkins wiki 2017)

As stated above there are two main reasons for Jenkins master/slave architecture. First one is to distribute the load from one Jenkins server to multiple slave servers. Second one is to use Jenkins slaves to provide a different operating system for tests and builds.

The basic setup for Jenkins distributed builds would be as follows. Jenkins is monitoring git repository for any changes that is made in the source code and committed in to the repository. After the change is detected, Jenkins is building the project and running tests. To be able to run the project in multiple environments, different tests for each operating

system are made. These tests could not be handled with a single Jenkins server, so slave servers are used.

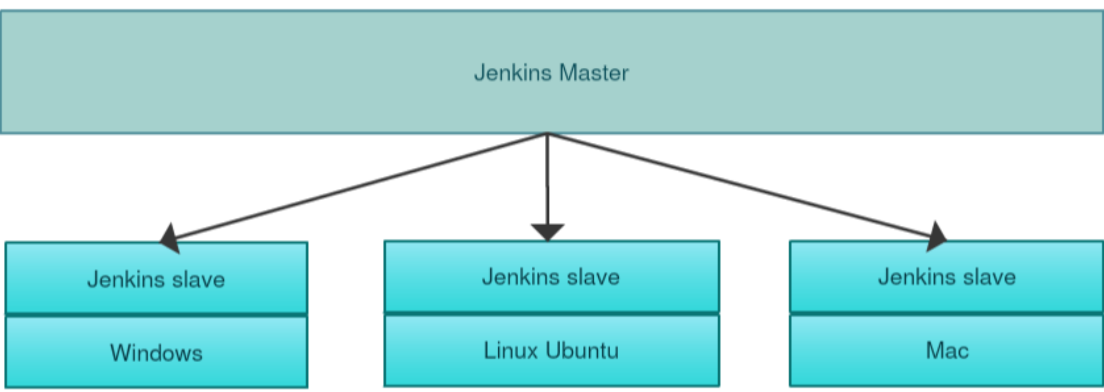

Figure 2. Example Jenkins master/slave architecture

4.9 JIRA

JIRA is a project management tool that helps teams to manage projects more easily. With JIRA users can made issues considering the work they are doing, and issues can be prioritized by the severity of the bug or a problem. The issues can be bugs to correct, simple tasks or new features to add to the application to name a few. The main part of the JIRA project management tool is the dashboard where all the key information about the project that is going can be shown. Dashboard is highly customizable, and it can hold almost anything regarding the project, charts, schedules, milestones, open issues and so on. JIRA is supporting SCRUM and agile development. The Kanban board is the one-stop tool to see what issues are open, under work or done. Drag and drop can be used to change the status of the issue. JIRA is supporting over 100 add-ons that extend the functionality of the initial JIRA. Popular add-on for agile teams is Agile Planning with Greenhopper that lets agile teams to add sprints, track the work that has been done and to report the progress. JIRA is free to use for 30 days and after that the paid plans are used. (Atlassian main page, JIRA 2017. JIRA in a Nutshell demo, by Atlassian 2012)

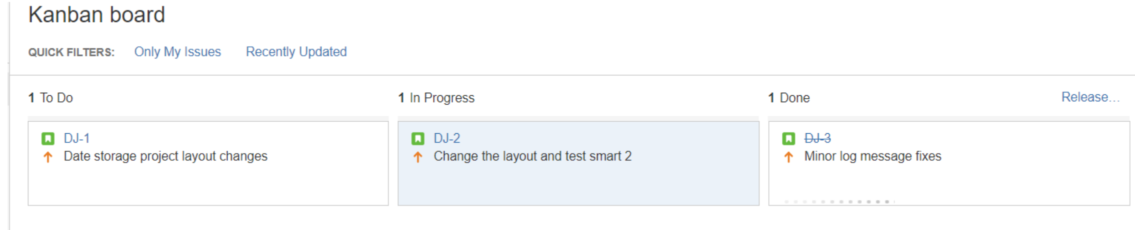

Figure 3. Kanban board example 1

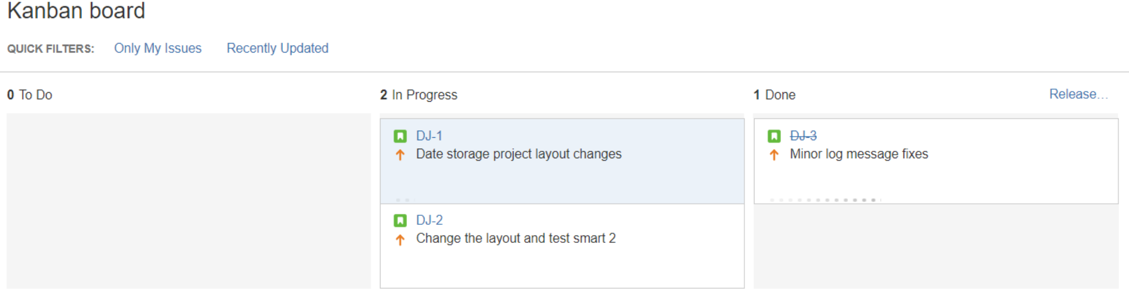

Figure 4. Kanban board example 2

In the figures 3 and 4 the Kanban board process is presented. With Kanban board it is easy to track the progress of different issues in the workflow.

4.9.1 Fisheye

Fisheye is a revision-control tool and works as a plugin for JIRA to extend JIRA functionalities. With fisheye the code repository can be shared with Fisheye and the information about code changes can be brought to JIRA issues through Fisheye revision. Inside Fisheye the code can be reviewed, and Fisheye is providing high-class tools to collaborate with the code including a side-by-side review. Everything in Fisheye works as a URL so code reviews can be linked straight to the selected line of code by sharing the copied URL link for that review. High chart support is also provided to view the changes made on any branch with visual aid that provides an easy to read visible information about the branch. (Fisheye, Atlassian main page 2017. Atlassian Fisheye overview video, by Atlassian 2010)

4.9.2 Crucible

Crucible is a code review tool that is used often in synchronized with Fisheye. It is used to tell peers to review the code that is committed and to give feedback on the code. The reviews can be linked to JIRA issues to help peers know what is the issue that the review is considering about. All of the code inspection is done inside Fisheye and Crucible without the need of other mediums to see the code. Crucible is an essential tool to connect JIRA and Fisheye for creating code reviews, giving a feedback to other people code and also to change the code according to the reviews. (Crucible, Atlassian main page 2017. Intro to Crucible, by Atlassian 2010. Crucible, Wikipedia 2017.)

4.10 MongoDB

MongoDB is a non-relational database that is used more and more on future projects. MongoDB is open source and free as it is published under a combination of GNU Affero General Public License and Apache License. It is written in C++ and mongo shell works on JavaScript. Compared to a relational database terminology, MongoDB terminology differs in a significant way. Terms table, row and column are called collection, document and field in MongoDB. MongoDB schemas are built with documents that are using JSON (JavaScript Object Notation) as the data format and new collections and fields are updated to existing databases with no need of creating the additional information separately. It is a cross-platform, document-oriented, database system. (MongoDB, Wikipedia 2017.)

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents

Figure 5. Relational database VS. non-relational database

As seen in the figure 5. the syntax for conventional relational database (in this case MySQL) and a non-relational database (in this case MongoDB) are significantly different, but perform the same tasks. (MongoDB overview, Tutorialspoint 2017.)

4.10.1 Replication

Replication is a way of making the database reliable and the data highly available. With MongoDB this is achieved with replica sets that consists of a one primary member and multiple secondary members. Primary member is receiving all the write operations and the data is then asynchronously copied to the secondary members.

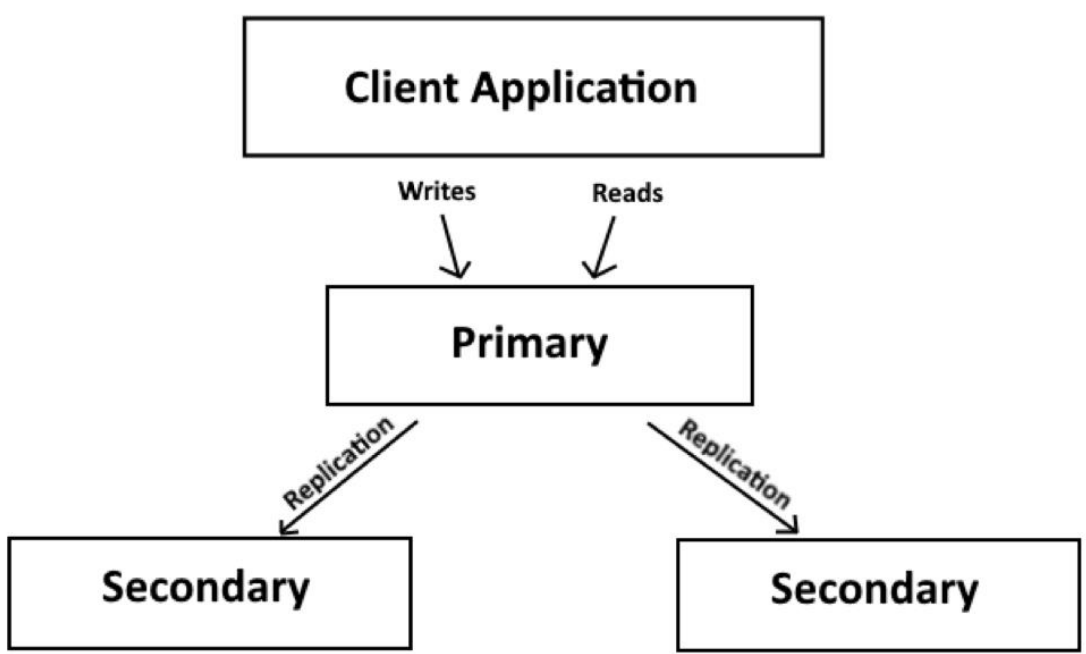

Figure 6. Basic structure of a single replica set with 3 members

There are multiple reasons why replica sets are done. Main points are increasing redundancy, making the data highly available and disaster recovery for the data. Replication of the data is the basis of any production level deployment.

All the changes in the replica set are saved in the oplog. Secondary members are then copying the information from the oplog and replicating the data in secondary members. (Replication in MongoDB, MongoDB docs 2017.)

If for some reason the primary member becomes unavailable, the secondary members hold an election to decide a new primary member among the secondary members. First secondary member that is holding the election and receives the majority of votes within secondaries becomes a new primary member. The usual minimum requirement for the replica set is to have a one primary member and two data holding secondary members. There is also a possibility to have one primary, one secondary and one arbiter, but this is not considered to be as reliable as the first option. Replica set should always be built in the way that there is an odd number of members in the replica set so the majority of votes can be achieved. Replica set can hold up to 50 members and have 7 voting members. (Replica set members, MongoDB docs 2017.)

Arbiter is a replica set member that only purpose is to give majority vote when election is performed. It is not holding any data and do not need any hardware to work. Only one arbiter can be made per replica set. Arbiter cannot become primary member in the same way as primary can become secondary and secondary can became primary member. The election process will start if the primary member is unavailable for 10 seconds. Usually the election process will take approximately one minute from start to finish.

4.10.2 Sharding

MongoDB sharding is a process of partitioning data across multiple servers which is providing several beneficial things. It is the MongoDB's way of horizontally scaling the database. It is used for geo-locality which means that the data is distributed across the world, so the data is accessible from the closest server possible. It is also used to achieve hardware optimization. Hardware optimization is achieved with using different kind of storage devices (RAM, SSD, HDD) to access different kind of data inside the sharded collection. It is possible to use more expensive and fast storage drive to storage the meta data of the program that needs fast reads and writes and use a more low-cost solution for, for example historical data that is not time sensitive. (Everything you need to know about sharding, by MongoDB 2014.)

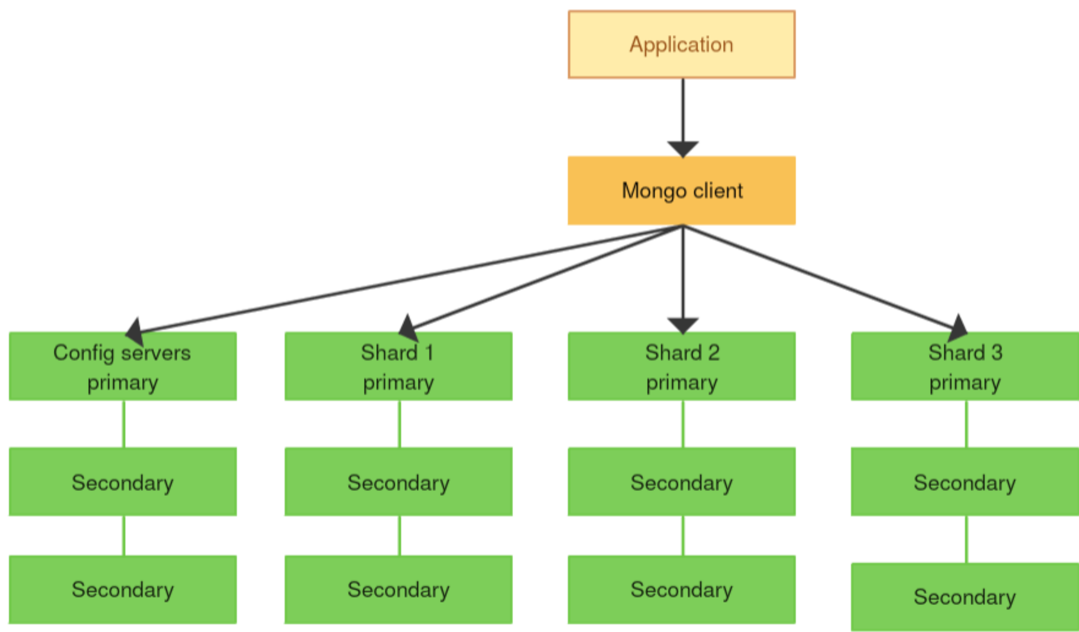

Figure 7. Basic sharded collection structure with 3 shards and a config server

The typical small deployment is a single replica set that is providing a primary member and multiple secondary members. This is providing high availability, but this is not scalable. To scale the database, example that is seen in figure 7. would be as using a client mongo that is connected to the application. The client mongo is connected to a shard clusters that each holds a replica set. The information about the shards and is stored in the config servers that is holding the meta data. Config server is also a replica set and as long as one config server is up the sharded cluster is working. (Sharding in MongoDB, MongoDB docs 2017.)

Scaling with sharding is providing benefits in the hardware costs as the data is distributed across multiple smaller server machines. As for example, most relational databases scale vertically that means that the data is stored in one machine. With MongoDB horizontal scaling if there is need for more storage, the thing to do is add one more machine at the addition of the existing ones. If the database needs to scale down, the thing to do is remove a machine from the cluster.

4.10.3 Authentication

Authentication is not enabled by default in MongoDB and should always be configured to the database before deploying the application to production. MongoDB database supports different roles which can be configured to have separate privileges inside the database. They

can be configured in mongo shell. Authentication and authorization inside the sharded database is done with keyfile authentication. The authentication must be enabled individually within every component of the cluster. The keyfile must be same for every cluster. (MongoDB authentication, MongoDB docs 2017. Manage user and roles, MongoDB docs 2017.)

5 BUILDING THE ENVIRONMENT

5.1 Introduction

This chapter of thesis is a walk-through of building the development environment with the tools that are used in the web development field. The functionality of the complete environment can be seen in a figure 8.

Demo application code is committed in git repository. Git repository works as a centralized heart of the environment and when a commit is made to the repository, Jenkins is noticing the change and pulling the code. Jenkins is performing automatic tests to the committed code and if tests are successful it is going to automatically deploy the demo application to staging server. Jenkins user with administrator rights can then deploy the demo application manually to the production server, after reviewing the application in the staging server. Also, when the commit is made in to the repository with a JIRA smart commit tags, JIRA that is configured to notice changes in the repository, is going to change the given issue status and issue information's according to the commit message.

Source code repository in GitHub.
https://github.com/turc0ss/Date-storage-application.git

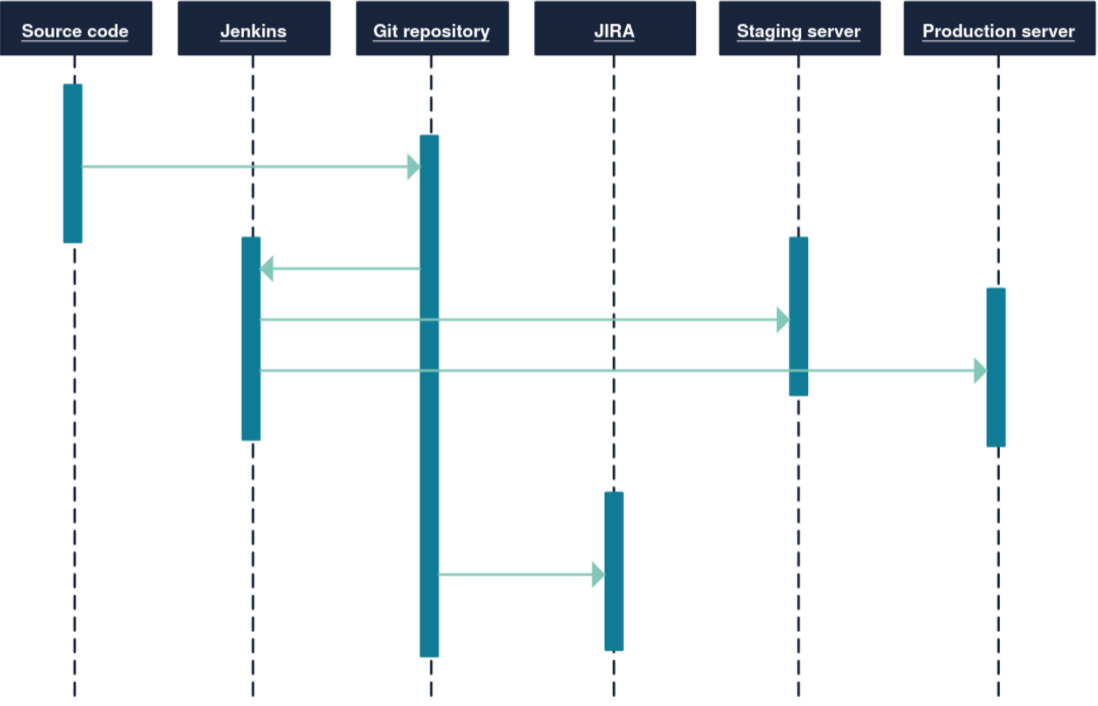

Figure 8. Sequence diagram of the project functionality

Building starts with planning how the complete product (environment) should work and how all the pieces all connected to each other. After a planning process, the next step is to set up the coding environment on the computer. This means the integrated development environment (IDE) that is used to code and build the demo application that is used to demonstrate other parts of a complete development environment. After the initial setup of the IDE, building the demo application can start. Coding is done in little pieces, testing the work after every relevant change. It also consists repeated commits on the version control system to save the progress and to back up the work. The version control system used in this project is GIT and the source code management platform is GitHub.

Basic flow of the coding process is to make the front-end first, test it and then add (connect) the back-end (database) to the front-end. After building the front-end of the project, logging statements are added inside the code, mainly for troubleshooting purposes. The back-end is holding only the basic functionality at first and after testing the back-end with the simple solution, authentication, replication and sharding are added to the MongoDB database. When the functionality of the application is in place and tested the next step is to configure the continuous integration environment with Jenkins and the project management environment with JIRA and connect these tools to the application. In the last stage the project is examined to see how all of these different parts connect to each other and why they are essential tools in any software development project.

5.2 Start with planning

Planning how the complete application should work is a major part of a development lifecycle. Good planning delivers a better application that meets the demands, and work as expected.

Programming language used in this project is Java, database choice is MongoDB. Vaadin framework with spring boot is used for the front-end and Jenkins and JIRA are used for continuous integration tools and project management tools. The operating system used to create this environment is 64-bit Windows 10.

The plan for the working application is as followed. In the front-end there is a simple browser interface which consist header text and a button. When the button is pressed, the time of the button press is printed on the browser. The text printed on the page would be as: **Date and time of a button press:** 19/11/2017 12:07:45. This is the only thing showing in the browser for the user using the application. Additionally, when the button is pressed, the time and date which are showing on the browser are saved on the database. In the database the saved data is replicated in primary and secondary databases in a replica set and sharded to provide

scalability for the system. The system is saving logs on the activities performed in the database and in the source code. The application is connected to the git repository and with every commit there will be automated functionalities for performing following tasks. Automated testing is performed through Jenkins. Jenkins is also performing continuous integration process to automatically deploy the tested code to the staging server where it can be examined by project manager before deploying it to the production server for product buyer or customer to see and use. Jenkins is also performing distributed build to the application to spread the load from the master node to the slave nodes. JIRA is working as project management tool that is handling issues and is also performing smart commits and collaborating with GitHub to get issue updates straight from commits made to the repository.

5.3 Start coding

When the planning process is done, and the end result is clear, the actual building of the application and configuring the environment starts. In this section the UI of the application is built to its final form.

5.3.1 Installing and configuring necessary tools

The first thing to do before any coding is to setup the basic elements on the computer that are needed to build this application. These elements are. Java JDK, Eclipse IDE, Maven, Tomcat, GIT and MongoDB. These needs to be installed to the local machine and then the environment variables for user and system are made and the paths for the variables are set. To me able to do any Java coding the Java Development kit (JDK) needs to be installed. This is done by going to URL http://www.oracle.com/technetwork/java/javase/downloads/ and selecting the latest JDK download link. In the time of building this project the latest version of java is java 8 so that is used. Accepting Oracle Binary Code License Agreement for Java SE and selecting right operating system in the list is finalizing the installation. In this project Windows 64-bit system is selected.

Next the IDE used to code Java is installed. In this project Eclipse IDE is used. Navigating to an URL https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2 and pressing the appropriate download link for the working computer operating system is downloading the zip file on the computer. Then the downloaded zip file can be extracted to any location on the computer. Eclipse can then be started with the executable file inside the extracted folder.

Maven is the next management tool to install in the computer. Navigating to URL https://maven.apache.org/download.cgi and downloading the binary zip file in the link section will download the maven on the computer. After downloading the zip file, the file can be extracted to any location on the computer. ***C:\Maven\apache-maven-3.5.0*** path is used in this project.

To setup tomcat server in the local computer the first thing to do is download a zip file from tomcat archives. Tomcat 9 is the latest version of tomcat, but tomcat 8 is used in this project to provide a stable server container to the project. In the URL https://archive.apache.org/dist/tomcat/tomcat-8/ there is a list of tomcat 8 releases. The 8.0.45 release is used in this project. To download this release first the 8.0.45 release is selected from the list, then the bin is opened and after that the correct version of the release that matches the operating system that is used is downloaded in to the local computer. Apache-tomcat-8.0.45-windows-x64.zip is the zip file for 64-bit windows. When that is downloaded and extracted to the computer tomcat is ready for use. Tomcat can be extracted and used from any location in the computer, but it is recommended to avoid extracting to program files folder because there can be permission problems later. The recommended path in the local computer would be C:\Apache.

GIT is the next thing to install and configure on the system. First thing is to navigate to URL https://git-scm.com/downloads and click the main button to download the latest the version of GIT release. Downloading the installer and running it is installing git in to the local machine.

MongoDB is the last thing to install and configure to have the basic setup running for building the application. First thing is to navigate to URL https://www.mongodb.com/download-center and press the **community server** tab. In that tab **Windows Server 2008 R2 64-bit and later, with SSL support x64** is selected from the version dropdown and then **Download** button is pressed. After the installer is downloaded to the local computer it can be run to finalize the installation.

After the systems are installed to the local machine the next thing is to set the environment variables for each part. In this project this is done with scripts that are doing all the heavy lifting. Power shell is used to run the scripts. Power shell is an advanced command line that is part of a Windows. First the PowerShell is started with administrator rights. Then the power shell is navigated to the path where the scripts are located. Inside this folder set_env_variables.ps1 is run with `.\set_env_variables.ps1` command. With this script the environment variables for these installations are set to the local machine.

Code 1. Content of the set_env_variables.ps1 script

```
setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_144"
setx M3 "C:\Maven\apache-maven-3.5.0"
setx M3_HOME "C:\Maven\apache-maven-3.5.0\bin"
setx GIT_HOME "C:\Program Files\Git"
setx /S tapio-dell-xps JAVA_HOME "C:\Program
Files\Java\jdk1.8.0_144" /M
setx /S tapio-dell-xps M3_HOME "C:\Maven\apache-maven-3.5.0" /M
setx /S tapio-dell-xps GIT_HOME "C:\Program Files\Git" /M
setx /S tapio-dell-xps MONGO_HOME "C:\Program
Files\MongoDB\Server\3.4" /M

Start-Job -FilePath "C:\date-project-scripts\java_path.ps1"
Start-Job -FilePath "C:\date-project-scripts\maven_path.ps1"
Start-Job -FilePath "C:\date-project-scripts\git_path.ps1"
Start-Job -FilePath "C:\date-project-scripts\mongodb_path.ps1"
```

These scripts are specific to the local machine that is used to create this project. Certain folder paths need to be changed to be able to use them on the different machine. Java, maven, git and maven path scripts need to be at the date-project-scripts folder that is created for running these scripts. To test that all the variables and paths are set correctly, commands showed in the figure 9. are used. All the version commands are showing information about the given tool that was installed.

```
PS C:\Users\tapsa> java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
PS C:\Users\tapsa> mvn --version
Apache Maven 3.5.0 (ff8f5e7444045639af65f6095c62210b5713f426; 2017-04-03T22:39:06+03:00)
Maven home: C:\Maven\apache-maven-3.5.0\bin\..
Java version: 1.8.0_144, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_144\jre
Default locale: fi_FI, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
PS C:\Users\tapsa> git --version
git version 2.14.1.windows.1
PS C:\Users\tapsa> mongo --version
MongoDB shell version v3.4.9
git version: 876ebee8c7dd0e2d992f36a848ff4dc50ee6603e
OpenSSL version: OpenSSL 1.0.1u-fips  22 Sep 2016
allocator: tcmalloc
modules: none
build environment:
    distmod: 2008plus-ssl
    distarch: x86_64
    target_arch: x86_64
```

Figure 9. Testing environment variables for the tools that are used in the project

Now the basic elements that needs to be installed and configured to be able to start coding and building the environment are set.

5.3.2 Configuring Eclipse IDE

The first thing to do when starting to build this environment is starting a new project in Eclipse IDE. The default path can be modified and in this project the path is **C:\Eclipse_Mars_works\ date-project.** When eclipse starts a new project, there is some configurations that needs to be done in the IDE. In the preferences the first thing to be changed is change the project to use JDK instead of JRE. The way to change that is going to **window -> preferences** and selecting **Java** from the list on the left. From the dropdown of the Java selection, **installed JREs** needs to be selected and then from the panel showing the installed JREs, the only result which by default is JRE path need to be changed to JDK path. This is done by first removing the JRE and then selecting **add**, selecting **Standard VM** from the next step and then searching the installation directory by pressing **directory** on the next window and searching the path of the Java JDK installation path on the computer. The JDK path in this project is **C:\Program Files\Java\jdk1.8.0_144.** Next thing to configure from the preferences before starting a new project in eclipse is to change some Maven functionalities to search the repositories, to add the dependencies and finding correct class paths. This is done by selecting maven on the preferences list. When maven is selected the

main panel change to give some options for maven configurations. There are couple check marks that needs to be added for maven to work properly in this project. From the list following points need to be checked in addition for the default selection. **Download Artifact Sources, Download Artifact Javadoc, Download repository index updates on startup** and **Update Maven projects on startup.** After checking these pressing **Apply** and then **Ok** will apply the changes. Last thing to configure in eclipse to make maven work properly is to add the index search for the repositories. **Window -> Show View -> Other -> Maven -> Maven repositories -> Ok.** This will add the Maven Repositories tab on the information toolbar. Pressing the tab will show all the repositories. From this view the **Global Repositories** dropdown triangle need to be pressed to show the central maven repository and after that right clicking the central maven repository will show setup list. From this list the selecting **Full Index Enabled** will result the maven to start searching the needed JAR files from the central repository. After that eclipse is configured and building the environment can begin.

5.3.3 Simple spring boot hello world application

There is a several ways of starting a new project in eclipse. Selecting file from top main menu and going from there is the choice used in this project. Selecting **File -> New -> Maven Project** starts a new maven project. In the New Maven Project window **create a simple project (skip archetype selection)** is checked as well as the **Use default Workspace location**. On the next window eclipse wants to know the groupId and artifactId for the project. GroupId is the id for the project's group (pom) which is like a package name in this case and the artifactId is the id of the artifact (pom) which represent the name of the project. In this project following names are used. **GroupId: com.tmattila, artifactId: date-project.** The version and packaging are left as default in this point. First thing to do with a new maven project is going to the pom.xml file. Inside the pom.xml everything in the project can be changed to match the wanted result. Maven will take care of importing correct JAR files to the project, to make desired functionalities work. Inside the pom.xml file the main information's of the project can be seen quickly. It is also possible to change the information's right there on the overview screen, but changing them on the pom.xml tab is giving more control over the configurations. Selecting the pom.xml tab will show the XML layout for the project configurations. Initial setup contains some basic information about the project that was just made. The model version, groupId, artifactId and version, which in the stage of building the initial product is 0.0.1.SNAPSHOT. That is saying that the current project is a snapshot of the work at that time. The first step is to setup basic web server and launch application to browser with a classic hello world text showing on the browser. In this first step

the goal is to show hello world in the browser using spring boot. Several things need to be imported for this to work.

```
<packaging>war</packaging>

<properties>
    <java.version>1.8</java.version>
</properties>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.8.RELEASE</version>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <executable>true</executable>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Figure 10. Basic hello world spring boot application pom.xml file

Inspecting the changes in the pom.xml one by one is showing that packaging is changed to WAR because this application is a web application and as the name WAR, Web Application Archive states packaging need to be war to be able to launch the application in the browser. The java version is changed to Java 8 from the default Java 5 for all of the Java 8 functionalities to work correctly. The parent section spring-boot-starter-parent is stated as a

parent of the project and the version for the spring boot is set to be 1.5.8.RELEASE as it is the latest release of the spring boot at the time of building this project. In this parent section the purposes of groupId and artifactId are shown clearly. Org.springframework.boot is the path where maven is searching for the dependency which name is in this case spring-boot-starter-parent. Then the correct JAR file is imported to the project to fulfill the desired functionality. In this case the parent functionality means that all the spring boot starter imports would be version 1.5.8.RELEASE without being separately stated so in the given dependency imports. Then the dependencies section is for the imported dependencies. The spring-boot-starter-web is needed to be able to launch the application to web browser. Spring-boot-starter-web is importing all of the needed dependency JAR files as a bundle without the need of stating all of the dependencies separately. Spring-boot-starter-tomcat is stating that when provided use a tomcat server instance. And finally, in the build section different plugins are set for the project.

Spring-boot-maven-plugin provides spring boot support for Maven, allowing you to package executable JAR or WAR archives and run the application “in-place”. (Spring Boot Maven Plugin, Spring Boot docs 2017.)

As the above statement from spring boot docs about spring-boot-maven-plugin states, this plugin is collecting all the JAR files and package them into the WAR file that is deployed in the browser. Those are all of the changes pom.xml needs in order to deploy this basic application in to the web. As for universal rule, every time the pom.xml is changed it is necessary to update the maven project. To do this right click the project name and then **maven -> update maven** is selected from the presented list. By doing this the project is always up to date and if there is conflicts that do not get solved by updating the project they will be presented as early as possible.

Project Explorer
- date-project [boot]
  - Deployment Descriptor: date-project
  - Spring Elements
  - JAX-WS Web Services
  - Java Resources
    - src/main/java
      - com.tmattila
        - App.java
        - Hello.java
    - src/main/resources
    - src/test/java
    - src/test/resources
    - Libraries
  - JavaScript Resources
  - Deployed Resources
  - src
  - target
  - pom.xml
- Servers

Figure 11. Project layout for the basic spring boot hello world application

```
package com.tmattila;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class App extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(App.class);
    }
}
```

Figure 12. Application class for the basic hello world spring boot application

```
package com.tmattila;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Hello {

    @RequestMapping("/hello")
    public String index() {
        return "Hello World!";
    }
}
```

Figure 13. Hello class for controlling the output of the application

The hello world application is build with two classes in this initial test application example. First class is the App class that is the main class of this application and is containing the main method.

The SpringBootApplication annotation is adding lots of different annotations and configurations to the class. The main ones are the Configuration annotation that “tags the classes source of bean definitions for the application context” and EnableAutoConfiguration that “tells Spring Boot to start adding beans based on class path settings, other beans and various property settings”. Also, the ComponentScan annotation is very important annotation that the SpringBootApplication annotation adds to the class and this annotation “tells Spring to look for other components, configuration and services in the com.tmattila package, allowing it to find controllers”. The main method uses Spring Boot’s SpringApplication.run() method to launch an application. (Building an Application with Spring Boot, Spring.io Getting Started 2017.)

To be able to launch the application in the external server, the App class needs to be extended with SpringBootServletInitializer and then the configure method that is type SpringApplicationBuilder needs to be overridden. This method is given parameter SpringApplicationBuilder builder and the method is returning the builder.sources() method that takes the App class as a parameter. (Java EE with Vaadin Spring Boot and Maven, Udemy 2017.)

The Hello class is flagged with RestController annotation that itself holds also a couple of different annotations. The main annotation that it holds are Controller annotation and ResponseBody annotation. The Controller annotation indicates that the annotated class is a controller, in other words a web controller and that it is also a component that is found in the

component scan of the main app class. ResponseBody indicates that the method return value should be bound to the web response body.

And finally, the RequestMapping annotation that is setting the URL path for the given annotated method by mapping the web request to specific handler. In this example the index method is simply returning a hello world string that is printed to the browser in http://localhost:8080/date-project/hello/.

For being able to run this on the browser, it is also possible to use the build in functionalities of maven for having build in tomcat server and just run the application as a java application from the main class. In this project more control is wanted so external tomcat server is setup. To setup the server right clicking on any class and selecting **run as…** will open the list of options. From there selecting **Run on Server** will start the configuration process for own server if it is not configured yet or it is necessary for some reason to change the server or the configurations (needs to be WAR packaging for the project to be able to use this option). On the top of the list **Apache** is selected and from there **tomcat v8.0 server** Pressing **next** will open the next panel where the server directory is set. Browsing the installation directory and selecting the **directory** which contains the bin folder will configure the path for the selected server to be the server eclipse will use. In this project following path is used: **C:\apache\apache-tomcat-8.0.45-windows-x64\apache-tomcat-8.0.45.** The server do not have to be tomcat server, but in this case tomcat is used. After setting the path and pressing next, the next panel will show all the available works able to be deployed to the server. Selecting **date-project** and pressing **finish** is launching tomcat and deploying the work to the server. Navigating to the **localhost:8080/date-project/ui** in the browser is showing the result, that is the application deployed to the browser, in this case a hello world message.

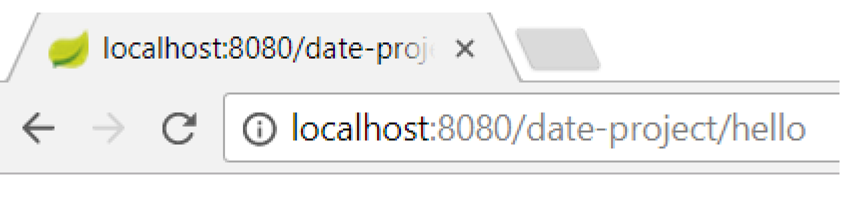

Figure 14. Result of the basic hello world spring boot application

5.3.4 Committing work to GIT

It is always good to push the changes on the source control repository as early as possible and as frequently as possible. That is the next thing to do in the project. To be able to push changes in to the repository, first a repository has to be made. GitHub is a repository used in this project. To create a repository first thing to do is create a GitHub account. After that is done, pressing new repository from the main page will open a repository creation window. From there the first thing to give is the repository name. The name for the repository in this project is called **Date-storage-application**. The description is voluntary and can also be added later if needed. Checking public and Initialize this repository with a README and then pressing create repository will add a new repository in the GitHub account. Now the first step is done.

The next step is to install a plugin for eclipse called egit that gives GIT command GUI inside eclipse. To do this going in the toolbar **help -> eclipse marketplace** and searching egit will give a result of **Egit – Git Integration for Eclipse 4.6.0** (eclipse version will vary depending on the installation time). Pressing install will start an installation process that will install Egit for eclipse to be able to use it. Pressing ok and selecting default options is a good basic installation. Eclipse needs to be restarted when the installation process prompt that and after that the installation is complete and Egit is ready for use.

Next thing to do is connecting the repository to eclipse. That is done by navigating **Window -> Show View -> Other –> Git -> Git Repositories** and pressing ok. This will add the git repositories on the bottom toolbar similar way as the Maven Repositories were added before. The GitHub repository needs to be cloned by going to the **Date-storage-application** repository in GitHub and pressing green **clone or download** button and copying the URL from the URL field. To be noted that this is done with Clone with HTTPS selected. Inside eclipse pressing **Clone a Git Repository** from the options that are presented when the Git tab is pressed at the bottom of the eclipse window is opening a window where the repository URL is pasted. When the git URL is pasted in to the URL field, the host and repository path are filled automatically. Using authentication is recommended for any software so the authentication information's should be given in this stage. The user and password given in this stage will be asked every time a code is pushed on to this repository through eclipse. In this project GitHub account credentials are used. After pressing next, eclipse is searching for the repository and when it finds it, the next window is showing the master branch on branch selection. The next stage is to configure a local storage location for the repository. Usually the default destination is good. In this project **C:\users\tapsa\git\Date-storage-application** destination is used. When pressing finish the window will close and eclipse is in normal state

again. The connected repository will now show in the Git Repositories tab. Next thing to do is to configure the Git Repository. This is done by clicking date-project and selecting **Team -> Share Project.** Selecting the connected repository from the repository dropdown and clicking finish is configuring the repository inside eclipse. When all of this is done, project is now ready to be pushed in to the repository. Clicking **date-project** and **Team -> Commit…** is opening another window where it asks for identifications. Here any name and preferably a working email address is provided for identification purposes. When there are more people working the same project, it is necessary to have some kind of identification for who is committing what. Closing to the last stages there is another window where eclipse is asking for commit message and selecting the files that are going to be pushed to the repository. For commit message Initial commit for basic spring boot hello world application is chosen and the files that are going to be pushed are the two files that are starting with **date-project/src**, and the **pom.xml** file. After selecting **App.java**, **Hello.java** and **pom.xml** file, **commit and push** is pressed. This will combine the two commands into one and push the committed files immediately. When Commit and Push is pressed, and the credentials are given the first commit is now made into the GitHub repository.

Next step is to refine this basic hello world application to use the Vaadin framework and make the front-end of the date-project application. First pom.xml need to be changed to include all the necessary dependencies to be able to use Vaadin and then the project structure need to be changed and relevant classes added to the project.

To prevent cluttering the document excessively with the code, not all the code is showing in this document. To see the full source code, scripts and configuration files, GitHub repository is available for observation. The project code is examined and explained, but in some cases, all the relevant code was not able to fit in this document. In those cases, it is found inside the git repository.

https://github.com/turc0ss/Date-storage-application.git

5.3.5 Creating Vaadin UI

First thing that is changed inside the pom.xml file is **prerequisites maven 3** (figure 16). That basically tells the project to work only with maven 3.0.0 or higher version. That is important for the next step which is including Vaadin 8. Vaadin 8 requires Maven 3 or higher to work so including prerequisites inside pom.xml, that is guaranteed. Next Vaadin is added and this is done with three steps. First inside properties section **vaadin.version** and **vaadin.plugin.version** tags are added in the pom.xml to setup the version that is

desired for Vaadin. Then it is necessary to add a new section inside pom.xml that is **depencymanagement**. Inside these new tags **vaadin-bom** is setup to use the vaadin version that is set inside properties section. Vaadin-bom stands for Vaadin Bill of Materials (BOM). Final thing to add in the pom.xml are dependencies for Vaadin. **Vaadin-spring-boot-starter** and **vaadin-spring** are added to give the project Vaadin provided functionalities. And **spring-context** gives the project spring related functionalities to make sure that spring is working correctly.

```
<prerequisites>
        <maven>3</maven>
</prerequisites>

<properties>
        <java.version>1.8</java.version>
        <vaadin.version>8.1.6</vaadin.version>
        <vaadin.plugin.version>8.1.6</vaadin.plugin.version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencyManagement>
        <dependencies>
                <dependency>
                        <groupId>com.vaadin</groupId>
                        <artifactId>vaadin-bom</artifactId>
                        <version>${vaadin.version}</version>
                        <type>pom</type>
                        <scope>import</scope>
                </dependency>
```

Figure 15. Changing the project to work with Vaadin 8

Coding the front-end of the date-project start with changing the project package structure as seen in the figure 18. Removing the hello class is the first thing as that is no longer needed in this project. The package com.tmattila contains the application class, that is the main class of the project. The annotation **SpringBootApplication** tells the project that it is a component, it provides component scan to search other components inside

the project and it also provides configurations for the project. Then a new package **com.tmattila.ui** is added to project to hold the **MainUI** class. This class is the sole class to hold all the front-end design and layout information for the date-project. In the more complex project there would be numerous other classes to hold UI information's for all the relevant fields of the program. Inside this class the first thing to do is to extend the base class with a Vaadin UI class. This is a Vaadin class that holds in all Vaadin functionalities. Next thing to do is to set some annotations to change the theme of the project and setting a custom title that is the tab title inside the browser. **@Title("Date Project")** gives the tab inside the browser the relevant custom name that in this case is Date Project. It could be anything, there is no limitations naming this custom tab. **@Theme("valo")** annotation makes the project use Vaadin theme valo for design and layout. **@SpringUI(path="/ui")** adds the given class to uiscope and this class is automatically picked up with spring. The path="/ui" defines the URL for the project so in this case the URL to use for launching the project in browser at this time is **localhost:8080/date-project/ui**. The MainUI extends the UI class so the class can use Vaadin related classes and methods. The UI class is the main class for creating UI with Vaadin. When the class is extended with Vaadin UI there is a init() method that needs to be implemented and this works the key method for launching Vaadin components. To get the front-end working, needed Vaadin components are first declared and then initialized inside the init() method. VerticalLayout as rootLayout, HorizontalLayout as headerLayout, buttonLayout and dateMarkings. Java Date class is declared (date.utils) on the top of the class to have a date variable in use. And lastly Java DateFormat and SimpleDateFormat classes are used to modify the default date format to a custom date format. Inside init() method these components are initialized. The headerLabel is getting its content from the other class called DateStringUtils that is a enum class and holds all the string representations of the project. This is convenient way of making the project international if necessary. If change of language needs to be made or if there is a same string that is used in lots of places in the project, then there is only one place that control the output of the strings. All that needs to be done if some string value needs to be changed is to change it inside the enum class.

```
@Theme("valo")
@Title("Date Project")
@SpringUI(path="/ui")
public class MainUI extends UI {

        private VerticalLayout rootLayout;

        private HorizontalLayout headerLayout;

        private HorizontalLayout buttonLayout;

        private HorizontalLayout dateMarkings;
```

Figure 16. MainUI class Vaadin annotations and layout components

```
date-project [boot]
    Deployment Descriptor: date-project
    Spring Elements
    JAX-WS Web Services
    Java Resources
        src/main/java
            com.tmattila.launcher
                App.java
            com.tmattila.ui
                MainUI.java
            com.tmattila.utils
                DateStringUtils.java
        src/main/resources
        src/test/java
        src/test/resources
        Libraries
    JavaScript Resources
    Deployed Resources
    src
    target
    pom.xml
Servers
```

Figure 17. Date-project front-end project structure

In Vaadin there is a basic structure that repeats itself very often. First a layout is created. Then there is some component that is created and then the component is added to the

layout. Finally all the layouts that holds a lot of components and functionalities are added into the main layout (called rootLayout in this project) and the rootLayout is set to the content of the page. In this page there are three parts. A header, button and the date markings when the button is pressed. This page is constructed in the similar way that was told above. First the main rootLayout is initialized, headerLayout is initialized and header label is added to the header layout. Then a buttonLayout which contains the button is created and the button itself is created. For the date markings another layout is created. When these layouts and components are created the next thing is to create the functionality for the button. Button is using Vaadin button class addClickListener that is waiting for a button to be pressed for some action to be performed. The method is made by lambda expression -> and the action the button will performs is the buildLayout() method that is a custom method that is created for this project. Inside that method a new date instance is created for the time that the method is called, in other words, button is pressed. And then the date is formatted to be in custom format. To show the formatted date, two labels are made. First label is creating a helper text, ***Date and time of button press:*** and another label is combining the helper text and the formatted date. Then the new combination label is added to the dateMarkings layout and the dateMarkings layout is added to the main rootLayout. After the button method has done its work the init() method continues its work of making these layouts. First the buttonLayout adds the dateButton component to its layout. Then the main roorLayout gets the headerLayout and the buttonLayout and lastly the rootLayout is set to the content of the page. It is noticed that the dateLabel is added to the rootLayout before the other layouts, but because the buttonLayout is owning the dateMarkings layout, the order is correct. The final design of the front-end is presented in the figure 20.

```
@Override
protected void init(VaadinRequest request) {

        rootLayout = new VerticalLayout();
        headerLayout = new HorizontalLayout();

        headerLabel = new Label("<h2><b>" + DateStringUtils.HEADER_TEXT.getString() + "</b></h2>", ContentMode.HTML);
        headerLayout.addComponent(headerLabel);

        buttonLayout = new HorizontalLayout();
        buttonLayout.setMargin(false);
        buttonLayout.setSpacing(false);

        dateButton = new Button(DateStringUtils.BUTTON_TEXT.getString());
        dateMarkings = new HorizontalLayout();

        dateButton.addClickListener(e -> {
                buildLayout();
        });

        buttonLayout.addComponent(dateButton);

        rootLayout.addComponent(headerLabel);
        rootLayout.addComponent(buttonLayout);
        setContent(rootLayout);
}
```

Figure 18. MainUI class init method that contains UI structure of the application

After the front-end is made with a functionality of the application printing dates to the page when the button is pressed, it is again time to push the code in to the repository. This is done by right clicking the project name and then navigating **team -> commit**. The commit message should give an explanation about the content of the commit, for what has been done and/or what is the purpose of the commit. In this project the commit message for this second commit is *Initial commit for date-project with Vaadin front-end*. The files to select for this commit are all starting with **date-project/src** and in this case, it is noted that the Hello.java is also included for this commit, because it is erased from this project and is no longer needed. It is still showing in the past commit, but not in this one and not in the ones after this if it not added in this project again.

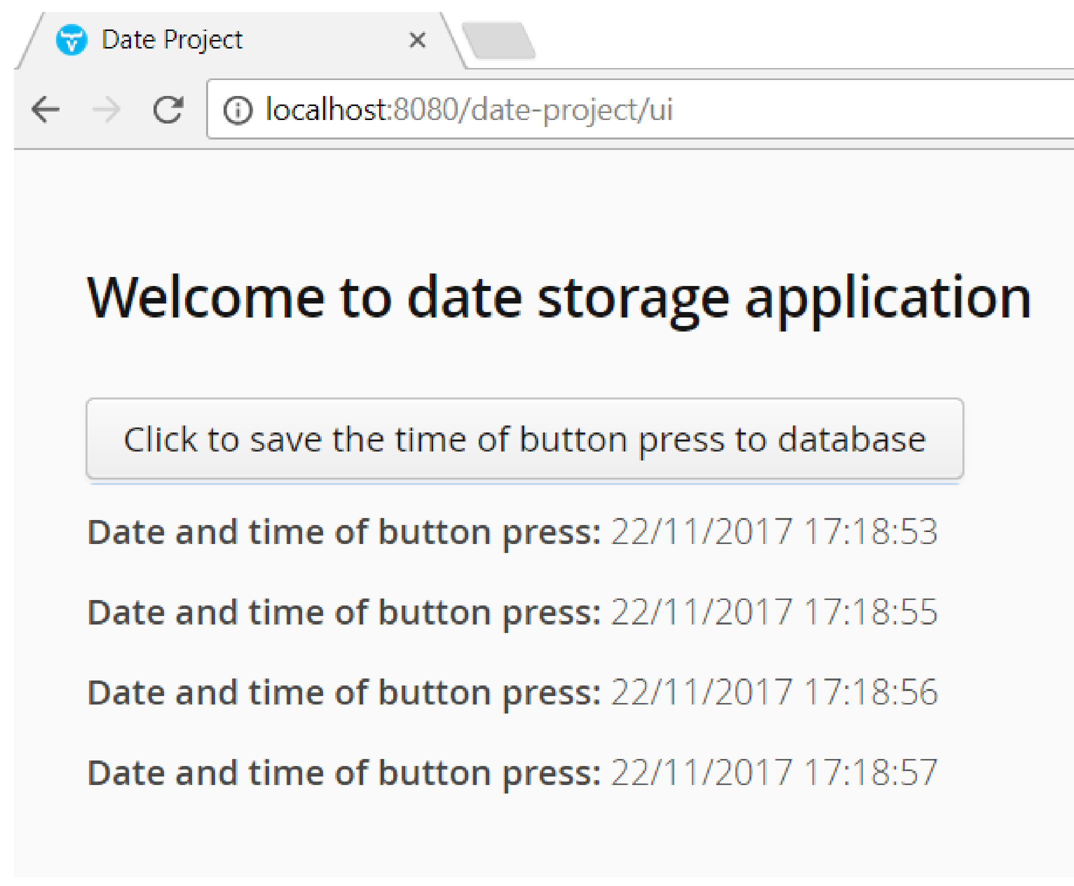

Figure 19. Front-end of the date-project

5.4 Adding logs

“Logging is very important part of software development lifecycle. The importance of logging is apparent when the product is launched in production and errors and/or bugs are starting to emerge. When there are errors that needs to be taken care of, it is essential that there is a way of investigating the code and try to replay what has happened. The only way to do this is having enough information to tell what has happened and logging is the way of getting that information.” (Logging and log4j.properties explained, by Brandan Jones 2015 (YouTube).)

Logging in Java applications is done with logging framework called log4j. There is little to none competition for this framework and it is considered of being the industry standard, when it comes of logging. It is free, reliable, very frequently used and easy to configure with log4j properties file or with log4j xml file. Creator of Log4j is Apache Software Foundation and it is created as part of an Apache Logging Services. Log4j2 is a version 2 of the log4j logging framework. It is highly recommended to use log4j2 for any future project because Apache has announced that it is no longer supporting log4j version 1.x. Log4j2 is not backwards compatible with version 1.x, but there is an adapter available. There are two main parts of

logging with log4j. Appender is something that publishes the logging information and logger is what collects the logs from the source code and provides them to appender. Multiple appenders can be assigned to a same logger. Root logger is considered as default logger. Logs can be saved to a file, sent to an email, printed in the console or saved on a server, to name a few options. Same log can be published in different places.

There are also different severities on the levels of logging messages. The different logging message severities from highest priority to lowest priority are fatal, error, info, warning and debug. With fatal level message the severity of the error is so big that the program is going to crash. Error level message is that something that would be in the catch block to signal that the given functionality didn't work as expected, but would still allow the program to be running. Info is designed to be information level that highlight the progress of the application. Warning is signaling potential harmful situations. And lastly the debug level is used to signal things that are most helpful for debugging the program like entering a method and exiting a method. (Logging and log4j.properties explained, by Brandan Jones 2015 (YouTube). Log4j Logging Levels, Apache Logging Docs 2012.)

The first thing to do to create logging statements in this project is to add dependencies in the pom.xml file for log4j. The dependencies to add are **log4j-api** and **log4j-core**. The artifactId's for these dependencies are **org.apache.logging.log4j**. In addition, second thing to add in the pom.xml file is dependency in the dependencymanagement section that is going to set the version of log4j to 2.10.0. ArtifactId is **log4j-bom**, version **2.10.0**, scope **import** and type **pom**.

After the dependencies are set, the next thing is configuring the Log4j. In this project, this is done with xml file with adding **log4j2.xml** file to the **src/main/resources** path. The logging messages are printed on a file on the computer and on the console inside eclipse.

The xml configuration file is very clear and descriptive. There are three parts in this configuration file. In the top of the file there is a properties section that is setting a file path for property named 'filename'. Then the appenders are set with a name and a target output. The first appender file is named file because that is also the target form and then the target is set to be the filename path. The path could be given right in this spot, but setting it in the in the properties section makes it for more readable. To call this path **${filename}** is used where the filename is the set as property name in properties section. In the patternlayout section the form of the message is set. In this project the form is set as **2017-11-28 07:21:30 DEBUG MainUI:96 [http-nio-8080-exec-7] - Exiting buildLayout()** for delivering as much information as possible. First it shows the date and time of the message and the severity level of the logging message. The next part shows the class where the logging message was

used with the line number inside the source code. Then the machine where the code is run is showing and in this case, it is localhost:8080. And lastly the logging message which in this case is giving information about the program exiting the buildLayout method. The next appender is for printing the log messages on the console. This is configured with adding console appender with a name console and target SYSTEM_OUT. The patternlayout is identical to the file appender because the message which is printed in the console is showing the exact same form. Then adding a logger section with a root level of debug and appenderRefs to the appenders will conclude the configuration file.

```
<Properties>
        <Property name="filename">C:/date-project-logging/logging.log</Property>
</Properties>

<Appenders>
        <File name="File" fileName="${filename}">
                <PatternLayout>
                        <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %C{1}:%L [%t] - %m%n</pattern>
                </PatternLayout>
        </File>

        <Console name="Console" target="SYSTEM_OUT">
                <PatternLayout
                        pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} %highlight{%-5level} %C{1}:%L [%t] - %msg%n" />
        </Console>
</Appenders>


<Loggers>
        <Root level="debug">
                <AppenderRef ref="Console" />
                <AppenderRef ref="File" />
        </Root>
</Loggers>
```

Figure 20. Log4j2.xml file, three main parts, properties, appenders and loggers

To add logs in the source code is straightforward. It is done by adding Logger declaration in the top of the class with a LoggerManager that have a parameter for the class that is using the logger. In the MainUI class this would mean. **Private static final Logger logger = LogManager.getLogger(MainUI.class);**. After the initialization of logger is done, using the logger is as simple as adding statement like **logger.debug("Exiting BuildLayout()");** in the part of the code that the logging message is needed. In this project a LoggingMessages enum class is used to hold all of the logging message strings so the language of the application with the message contents are easy to change.

5.5 Adding database to the project

Adding a database to the project starts with making a very basic save to database functionality through the mongoRepository interface without any service layer. The reason for this first part is to test the connection and save some data to database. The connection is made straight to the mainUI class, so it is directly connected to front-end. This is not the recommended way and the next thing to do after this initial test is to add a service layer to handle the connection to the database.

First thing to do is to create an entity that is going to be saved in the database. In this initial test there is only a frame that is filled with the data from front-end elements. Dates model start with declaring that the Dates class is a document for MongoDB by using **@Document** annotation above class declaration. Same time the collection name is declared to be name dates with **collection = “dates”.** The default collection name if the collection name is not defined is the class name that is using the document annotation. The fields that are declared are id, title and formattedDate. Id is a autogenerated string that MongoDB is generating. Title and formattedDate are strings and they are used inside the Dates constructor to fill the frame which is used inside MainUI class. In later stages the constructor is left empty and the entity frame is populated through getters and setters with service layer. To use MongoDB functionalities DateRepository interface is created and that interface extends MongoRepository. Parameters used for MongoRepository are **Dates** model class and **Date** class that is coming from java.date.utils. The interface is annotated with **@Repository** annotation to indicate spring that this is a repository and that this is a component. Last thing to do before saving data inside MainUI is configurating eclipse with MongoDB. This is done with **application.properties** file that is created inside **src/main/resources** folder. In this file the basic information’s about the database connection is established. This include the application name that is date storage application, a host name that is tapio-dell-xps after the assigned name for the local computer, a port number that the database is using that is in this project **20120** and a database name inside the MongoDB that in this project is **datedb**. After creating a Dates model, creating an interface that extends MongoRepository and configuring MongoDB and eclipse with application.properties file, the MongoDB functionalities can be used inside the project. Inside MainUI the DateRepository class is autowired in this class and after this the MongoRepository save method can be used to save data to database. In this initial test the way to save data to database is to save a new Dates object with parameters of title and the formatted date strings according to the Dates constructor.

In the local computer C drive new folders are made with a structure of **C:\date-project-initial-db\data\db** and **C:\date-project-initial-db\data\logs\logs.log**. The db folder is holding the

documents that are saved, and logs folder holds the information about the MongoDB loggings. For the project to be able to create and use a database, a socket needs to be open to the port that is using the database. Also, the database path and logging paths needs to be configured. This is done inside the command line. To use UNIX-type commands on the command line, git bash shell is used. It is a command line that is provided by git that use UNIX-type commands much like in Linux. After making those new folders typing

Code 2. Command to start a database in MongoDB

```
mongod –-port 20120 –-dbpath /c/date-project-initial-db/data/db --
logpath /c/date-project-initial-db/data/logs/logs.log &
```

is starting the MongoDB in port 20120. It is noted that in front of port, dbpath and logpath commands double dash is used for the commands to be valid. Double dash markings in front of a command is very regular thing in UNIX-type commands. That is it, the MongoDB can now be used with the project. To test this the project is run with a server in the usual way and couple of button presses is made from the front-end UI. The first time the button is pressed eclipse is printing in the console that MongoDB driver connection is made and opened. That means that the data is saved in the MongoDB database. To confirm this typing `mongo --port 20120` in git bash is opening a mongo shell that is used to interact with the data. After typing the `mongo --port 20120` command git bash is showing information about the MongoDB shell version, MongoDB server version and the port that the connection is was made. Command `show dbs` is showing all the databases. After using the demo application in the browser and configuring MongoDB there is a new database datedb in addition to admin and local databases. Command `use datedb` is telling the shell to use this database so that the following commands are affecting this database. Show collections is showing the collections inside this database and there is only one for now named dates. To see the content of this collection, `db.dates.find().pretty();` is typed in the shell. This is showing the content in a JSON format. `Db.dates.find();` is also working, but the data is showing in more packaged format and this can be more difficult to read when there is a lot of documents. The initial test is done, and MongoDB connection is verified. Next step is to make a service layer that is handling the data traffic.

```
{
        "_id" : ObjectId("5a1e475769d8103bcc198229"),
        "_class" : "com.tmattila.model.Dates",
        "Title:" : "date:",
        "Date:" : "29/11/2017 07:36:23"
}
```


Figure 21. Result of querying data in datedb, dates collection

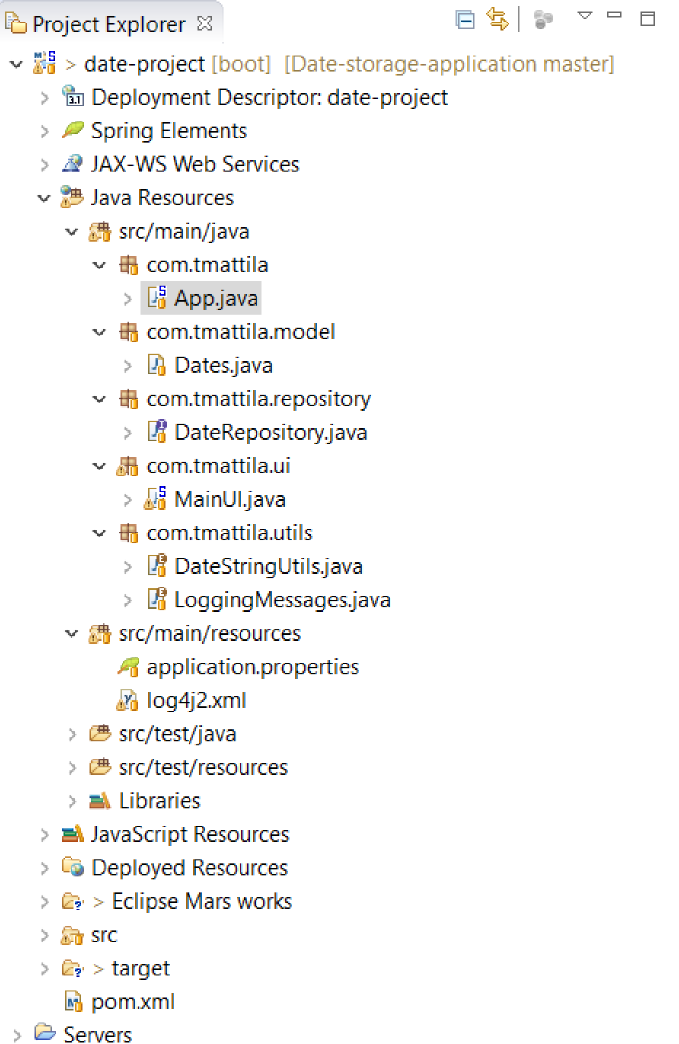

Figure 22. Project structure after the initial test for MongoDB

Refining the date-project to use service layer for managing database connection starts with clearing the Dates constructor from its content and adding getters and setters to title and formatted date. Then Date class from java.utils.date needs to be declared because the Dates is working with its own date to save that in to database, not the one used on the front-end. Also, getters and setters need to be created for Date. New date is initialized inside

setFormattedDate() and the date is formatted into dd/mm/yyyy hh:mm:ss form and set to be the formattedDate. ToString() method is also added to show the string representation of title and formatted date. Then com.tmattila.service package is created and it is populated with the DateService interface. This interface is holding one method that is saveDateToRepository with a parameter of Dates entitys DAO, data access object. Then a class named DateServiceImpl is created that uses the method created inside the DateService interface. This is done by implementing the DateService interface with implement keyword and implementing the interface methods. In DateServiceImpl class DateRepository is autowired so that the repository functionality can be used inside this class. Inside the implemented saveDateToRepository method a new Dates instance is created, and the title and formatted date is set with Dates class setters. The correct date is set by using the formatted date setter method and providing the date dao formatted date getter as a parameter. In this way the date that is saved on the database is the one that is created at the time of calling the saveDateToRepository() method. Then the newly created dates object is saved on to the database with dateRepository.save() method and providing the dates as a parameter for this save method.

Inside MainUI the Dates class is declared and then initialized inside the init() method. DateService is autowired in the root of MainUI class (outside of init medhod) so the saveDateToRepository is available to use inside MainUI. The functionality of saving data in to the database is moved in an own method called saveDate(). Inside this method a try catch block is used to hold the functionality of saving data to database so that if there is an error in saving the data it will be caught, and the error message is printed to the console with stack trace. Both MainUI saveDate() method and DateServiceImpl stages in the code are heavily logged with enter and exit logs and information about the stage of the program. And those logs are created in the LoggingMessages enum to store all the string messages in the same place.

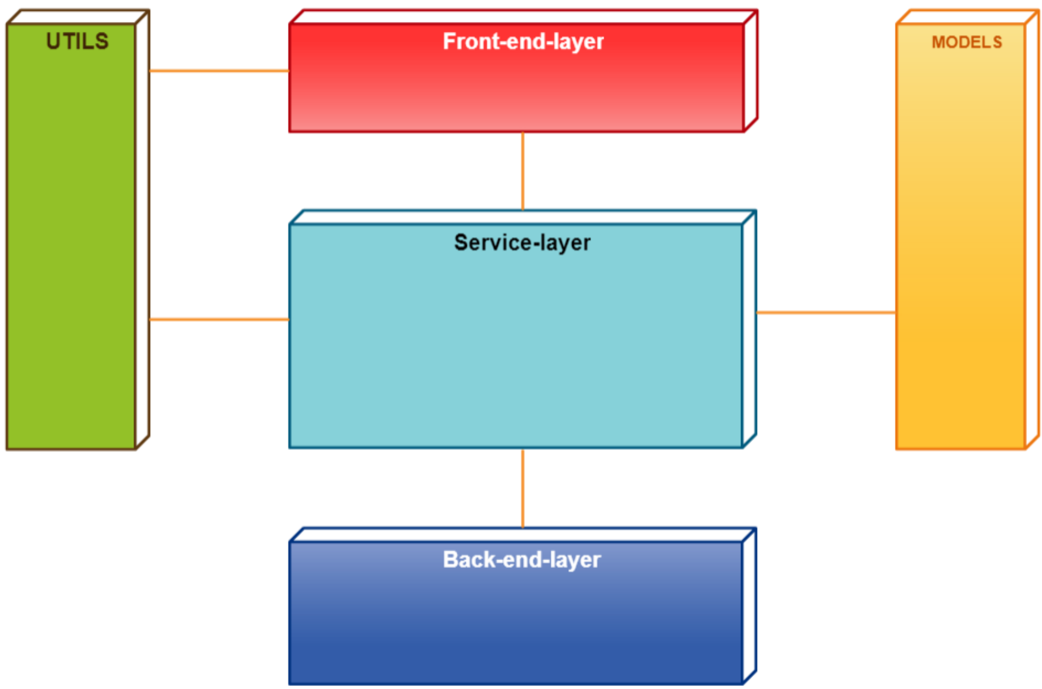

Figure 23. Final application layer presentation

5.6 Unit testing

To start the unit testing with JUnit, first thing is to include the dependencies to pom.xml file. The dependencies that needs to be included are **junit** and **spring-boot-starter-test**. To include junit to project dependencies add a dependency tag that holds a junit artifactId and junit groupId. Adding spring-boot-starter-test is very similar. First dependency tag is included and inside those, groupId is set to org.springframework.boot and artifactId is set to spring-boot-starter-test. Additional thing to add with this is the scope tag and inside those tags a test parameter is included. This tag is telling maven to use this dependency only when testing.

```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
```

Figure 24. Adding JUnit dependency to date-project application

Next thing is to add the **application.properties** file inside **src/test/resources** folder. This is almost identical to the application.properties file configured in database creation. The only difference is the database name because the tests are saved on the different database to keep the databases clean. Third thing is to add the test class inside **src/test/java** folder. This is done by adding a new junit test case with right clicking on the **src/test/java** and selecting **other**. From there typing junit in the typing field is showing the junit options suite and case. Selecting **junit test case** is working in this project. Next window is the configuration window for junit and here the package name and class name are set to the junit test case. The package name is **test.com.tmattila.datetorepositorytest** and class name is **DateToRepositoryTest**. It is very important that the class name is descriptive and ends with test word. This is the common practice on making test cases. SetUp() and testDown() stubs can be checked even they are not used in this test case. It is just good to have them there if needed. Eclipse is creating a skeleton of the junit test case with three methods, setup(), tearDown() and test(). In this test case all the test code is going on the test() method. First thing to do is tell the spring explicitly that this is test class by annotating the class definition with annotation **@RunWith(SpringRunner.class).** Another thing to do is tell the test case what is the class that needs to be tested. In this test case it includes the hole project functionality so the class that needs to be tested is App.class. This is done with annotation **@SpringBootTest(classes = App.class)**. With this the hole project is tested because App.class is the main class of the project. In this context the test is more like integration test than unit test (because the application is so small that there is only one functionality to test). In the test method a new Date object is initialized and DateFormat and SimpleDateFormat are used to format the date for right form. Then a new Dates object is initialized and with that the title and formattedDate are set. The test part is the Assert.assertNotNull() that is checking that there is no null content when saving data in the database. DateRepository.save(dates) is gived to a parameter for save method for the dates to be saved in the database. In this test case as in other code parts logging is also used. Logging messages for entering and exiting the test are made. Logs are also added to represent the formatted date and to tell that the test was completed successfully. These log messages are type info.

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
public class DateToRepositoryTest {
```

Figure 25. Initializing DateToRepositoryTest to be JUnit test case

```
@Test
public void test() {
        logger.debug("Enter repository test");
        Date date = new Date();
        DateFormat dateF = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
        String formattedDate = dateF.format(date);
        Dates dates = new Dates();
        dates.setTitle(DateStringUtils.DATE_TEST_TITLE.getString());
        dates.setFormattedDate(formattedDate);
        Assert.assertNotNull(dateRepository.save(dates));
        System.out.println(formattedDate);
        logger.info("Test complete");
        logger.debug("Exit test");
}
```

Figure 26. Test case for saving dates to repository

To be able to test the testing, project is run with maven build in **run configurations**. New maven build is made with the commands **clean package**. This is running the clean command for the project and cleaning the project from all the WAR files that have been built in previous runs and setting the project to be at default starting point. Then the package step is packaging the project that has been cleaned to be a separate war file that is distributed to the browser. With this the test classes are run on the package stage before the project is packaged to a war file. After this the project is separately pushed to the server and run on the server.

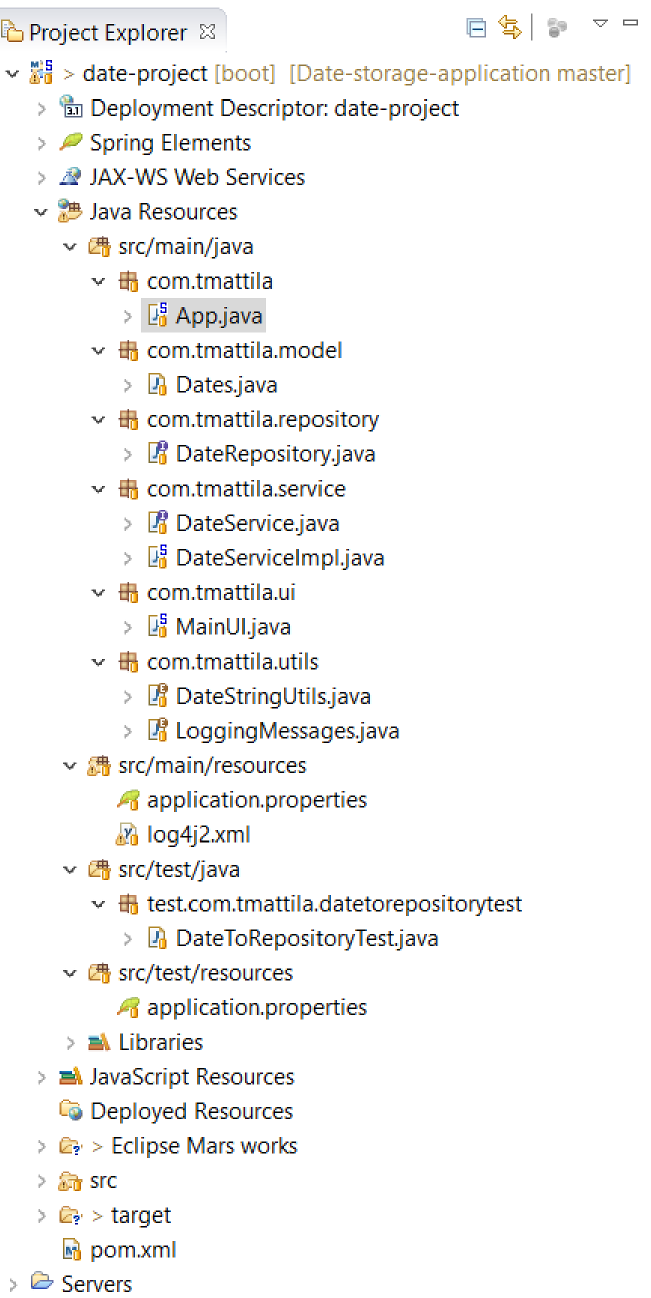

Figure 27. Final project structure after the test part is included

5.7 Adding replica sets and sharding to databases

At this stage of the project MongoDB database is a single folder in a local computer. To change this to be scalable and reliable database cluster, a replication and sharding are performed. With these actions database is going to consist of a sharded collection, a cluster and every cluster in the sharded collection, a replica set is made. Replica sets includes a primary database and secondary databases. In this project the database structure is as follows. Database is going to consist four shards and for each shard there are a primary

database for that shard and two secondary databases. Because the single folder database was only a step to test the database, it is not used in this project and the database is made from zero to be a sharded collection. Also at this stage the database is made secure with a keyfile authentication, so it is really usable in production environment. In the production version application database clusters would be on separate servers, not in a single computer, but this is a starting point from where it can be changed to be located on servers easily.

First thing to do is to create a new folder to hold all the shards in the computer. Date-project-initial-db is forgotten and a **date-project-db** folder is made in the c drive root. The architecture of the project database is consisting of a config-server replica set that is handling all the meta data of this database cluster. Meta data is an information about the characteristic of the database, not the actual data stored by the user. Then there are four shards that are replica sets and a one client mongo that is going to be the connection point for the user. All the actions for saving data, querying data, etc. are performed through a client end point, a mongo. This is also the end point for the application to be connected on the database cluster. To make a life little easier, shell scripts for creating the folder structure and shards are created. These scripts are included on the appendixes and explained at this stage. In this project the scripts are put in the date-project-db folder for simplicity. Git bash is used to be the command line in this database creating and configuration for its UNIX-type typing. In windows command line the commands used would vary a little. For any database configuration with MongoDB, a command line is used. There is no graphical user interface available to configure MongoDB databases. When the script files are in the date-project-db folder, the first step is to create the folder structure of the cluster. First the executable rights are added to the script, so it can be run. This is done by typing `chmod +x create_folders.sh` in the command line. This is adding an executable right to that script and is done for all of the scripts before running those at the first time. The *create_folders.sh* script is creating a keyfile and log folders in the root date-project-db folder and a parent folder for config-server and shards. The parent folder of the config-server and shards are filled with child folders that are the replica sets of that shard. So, in every shard parent folder, there are three child folders to contain the replica set primary and secondary members of that database. In the script this is simply made with mkdir commands. In these scripts the same commands could be typed directly on command line and the same result would happen. When the commands are long and repetitive, it is a good practice to use scripts to perform the actions. This is also reducing the change of typing errors. The script is run with command *`./create_folders.sh`*. After the script is run, the folders are made to hold all the information about the database and the data itself. The next thing is to create the keyfile that is used for authentication in the database. Every shard and replica set are going to use this

keyfile to perform authentication in the member. The `create_keyfile.sh` script is containing following script.

Code 3. Keyfile creation script

```
openssl rand -base64 741 > /c/date-project-db/keyfile/keyfile
```

This is creating a random 1003-character long string that is stored in the keyfile folder and the file is named keyfile.

When the base line of the database creation is made, the next thing is to start the shards. The first shard that is going to be created is the config-server. To start this, the executable mode is added to the script file and `./start_conf-serv.sh` is typed in the command line.

Code 4. Script for setting first replica set member for config server shard

```
mongod --replSet conf-serv --logpath C:/date-project-db/log/conf-serv0.log --dbpath C:/date-project-db/conf-serv/conf-0 --port 26050 --configsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

What this is doing is declaring that the mongod instance is a replica set named conf-serv. This is creating the first replica set member of the config server shard cluster. Then it is declaring that it is a config server. The smallfiles addition is there to reduce the default file size of MongoDB. This is helpful in the local computer setup, but should not be used in the production version. Then the script is setting the paths of the logs, keyfile and the database. The port number for this member is 26050 as declared in the script. The ampersand symbol is the fork symbol that is used to “Enable a daemon mode that runs the mongos or mongod process in the the background”. (Process management option, fork, MongoDB 2017.) This is the first member. The same procedure is done for the second and third member of the replica set. These are the port numbers 26051 and 26052, and the file path and member names are conf-1 and conf-2. Log files are named with same design. With the start_conf-serv.sh script all of the three members are started. **Netstat -a** is a useful command that shows all of the ports that are in use and can be

very helpful when confirming that the servers have been started. Next thing is to start shard servers a, b, c and d. This is also done with a script.

Code 5. Script to make a first member of shard a

```
mongod --replSet a --logpath C:/date-project-db/log/shard-a0.log --
dbpath C:/date-project-db/shard-a/shard0 --port 27001 --shardsvr --
smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

In the code 5. present a part of the start_shard-a.sh script and it is starting the first replica set member of a shard a in a port 27001. Only major difference to the config server script is the `--shardsvr` that is declaring this member as a member of a shard server. All the other parts are very much like config server script with the path and port declarations. The next thing is to initiate the replica sets that have been started as seen in a figure 29. This is done with first using `mongo --port 27001` to enter inside the replica set on a mongo shell, then setting the config variable to hold information about the name of the replica set and the information about the replica set members. After that `rs.initiate(config)` is used to initialize the replica set. This is starting the evaluating process to elect a primary member and secondary members. After this the command `rs.status()` is used to see the information about all of the replica set members. `Rs.isMaster()` command is used to see the specific information about the current member. This is also a good way to see if the current member is the primary node of the replica set for what the command is mainly used.

```
$ mongo --port 27001
MongoDB shell version v3.4.9
connecting to: mongodb://127.0.0.1:27001/
MongoDB server version: 3.4.9
config = {
"_id" : "a", "members" : [
{ "_id" : 0, "host" : "tapio-dell-xps:27001" },
{ "_id" : 1, "host" : "tapio-dell-xps:27002" },
{ "_id" : 2, "host" : "tapio-dell-xps:27003" }
]
};
{
        "_id" : "a",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "tapio-dell-xps:27001"
                },
                {
                        "_id" : 1,
                        "host" : "tapio-dell-xps:27002"
                },
                {
                        "_id" : 2,
                        "host" : "tapio-dell-xps:27003"
                }
        ]
}
rs.initiate(config);
```

Figure 28. Example of the replica set initialization process with shard a

After this the mongos client is going to be started. This is also done with a script in this project.

Code 6. Script start_mongos.sh to start mongo client in port 20120

```
mongos --logpath C:/date-project-db/log/mongos.log --configdb conf-
serv/tapio-dell-xps:26050,tapio-dell-xps:26051,tapio-dell-xps:26052
--port 20120 --keyFile C:/date-project-db/keyfile/keyfile &
```

This single line of code that is presented in code 6., is the only part of the script. It is declaring that the mongos is fetching the information about the sharded collection through config server that included three members and to listen a port 20120. Then there is the similar declaration of the log path and keyfile path.

The next thing is to connect to the mongo shell with the client instance `mongo --port 20120`. In a mongo shell the first thing is to create a super user that also the application is using. This is done by first using the admin database with `use admin` command and then using a command seen in figure 30.

```
use admin
switched to db admin
db.createUser({
        "user" : "RootUser",
        "pwd" : "mongo123",
        "roles" : [ "root" ]
});
Successfully added user: { "user" : "RootUser", "roles" : [ "root" ] }
```

Figure 29. Creating root user for the database

This command is creating a new user with a name **RootUser**, password **mongo123** and role **root** that is effectively giving the user a super user capability. Super user is a user that has all around permission in the program to do anything. Then in the mongo shell to use the root user it is necessary to authenticate the current user as the root user. This is done by using the admin database and using the command `db.auth("RootUser", "mongo123");`. The second user that needs to be created is the user that handles the config servers. The command used to create config server user can be seen in a figure 31. This command is much like the root user creation, but the name in this case is **ConfigUser** and the role is **clusterAdmin** for the admin database.

```
use admin
switched to db admin
db.createUser({
        "user" : "ConfigUser",
        "pwd" : "mongo123",
        "roles" : [ { "role" : "clusterAdmin", "db" : "admin" } ]
});
Successfully added user: {
        "user" : "ConfigUser",
        "roles" : [
                {
                        "role" : "clusterAdmin",
                        "db" : "admin"
                }
        ]
}
```

Figure 30. Creating cluster admin user for config server

If the `sh.status()` command is used on the mongo shell when authorized as root user, the information that comes up is telling that the mongo client is a sharded cluster but there are no shards added in the cluster. To add the a, b, c and d shards in the cluster the

`sh.addShard()` command is used. The hole command for adding a shard to the cluster is `sh.addShard("a/tapio-dell-xps:27001").` The parameter inside the addShard method are the shard name, server name that is in this case the local computer (computer name is used, not localhost) and the replica set primary member. The other shards are added in the same way. Now the sharded cluster is made. The next thing is to add authentication configurations inside the application code in application.properties files. Enable sharding for the databases that are created when the button is pressed, and the date is saved on the database that eclipse is creating and then the collection is sharded inside the mongo shell. Sharded collection is partitioned to chunks when the shard size is big enough. To be able to first test the sharded collection and be sure that the chunks in the sharded collection are made and that they are distributed across the shards equally, a test case is made. This is because the shard needs to be fairly large, for the shard to start making chunks and the date storage application date storing is going to be too tedious process to save hundreds-of-thousands records in to the database to show the sharded collection in action. To test this a new database named test-user is made with the command `use test-users`. After this a single document is inserted inside the database to activate the database. This is done with command `db.info.insert({ "user" : "test-user" });`. Info is the collection name in this example test. Now when the database is active it needs to be sharded. Using `sh.enableSharding("test-users");` is enabling sharding in this database and `sh.shardCollection("test-users.info", {"_id":1}, true);` shards the collection.

```
sh.enableSharding("test-users");
{ "ok" : 1 }
sh.shardCollection("test-users.info", {"_id":1}, true);
{ "collectionsharded" : "test-users.info", "ok" : 1 }
```

Figure 31. Enable sharding for database and collection

`Sh.status();` is now showing the database name test-user that is located in c shard in primary member of the replica set and that it is partitioned. It is showing the number of chunks in each of the shards and in this case c shard is holding 1 chunk. To see the sharding and balancing of the chunks in action, half a million new documents are inserted into the database with a for loop. The command that is used to do this is seen in a figure 32.

```
for(var i = 0; i < 500000; i++) {
db.info.insert({
"user" : "test-user: " + i,
});
}
```

Figure 32. Adding test users to test sharded collection chunk distribution

After MongoDB has inserted all the documents in to the test-user database (this takes a while), `sh.status()` command can be used to see the change that is made at this point. There are three new chunks that are distributed equally across the shards, so in total there are four chunks, one for each shard.

Inside eclipse in the application.properties file a couple of new properties are added. The first one is **spring.data.mongodb.authentcation-database=admin** that is setting the database which is used for authentication. Second is **spring.data.mongodb.password=mongo123** which sets the password and the third is **spring.data.mongodb.username=RootUser** which sets the user that is used for authentication. These same properties are also added in to the application.properties in the test. The last part is to add the same sharding commands in to the datedb and datedb-tests databases. `Sh.enableSharding("datedb");` and `sh.shardCollection("datedb.dates", {"_id":1}, true);`. Same things are also added in the datedb-tests after using maven build clean package to run the tests. That is it, now the date-project application is connected to the sharded cluster database.

```
spring.application.name=date storage application

#Mongo config
spring.data.mongodb.host=tapio-dell-xps
spring.data.mongodb.port=20120
spring.data.mongodb.database=datedb

#Server configuration
spring.data.mongodb.authentication-database=admin

#User configuration
spring.data.mongodb.password=mongo123
spring.data.mongodb.username=RootUser
```

Figure 33. Complete application.properties file for date-project application

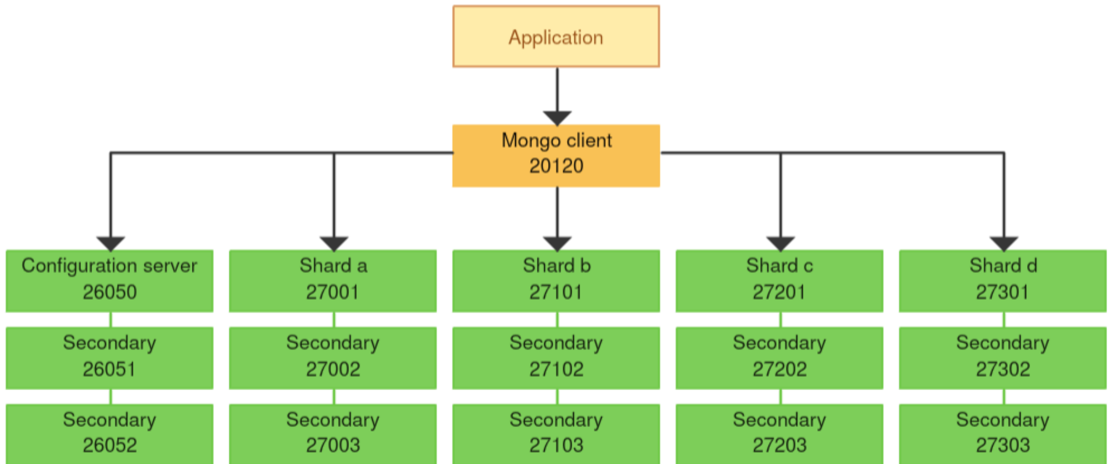

Figure 34. MongoDB sharded collection structure for application database

5.8 Connecting Jenkins

At this stage the project is working with a secure sharded database and a connection to version control system has been made and is working. The next thing is to connect the project to Jenkins which is handling the automated testing, automated deployment and distributed builds for the environment. First it is necessary to download a new tomcat server that is handling Jenkins. In this project tomcat-8.0.43 is used for this purpose and it can be downloaded in the same way as the tomcat-8.0.45 in the set up the environment section. After tomcat is downloaded and extracted to the computer the next step is to download a latest version of the Jenkins with the WAR file. This can be downloaded in https://updates.jenkins-ci.org/download/war/. When the downloaded WAR file is in the computer it is copied inside the tomcat-8.0.43 in **C:\apache\Tomcat-8.0.43\apache-tomcat-8.0.43\webapps** folder. When the **jenkins.war** file is in this folder, tomcat recognizes Jenkins when the tomcat server is run. Before running the Jenkins for a first time the localhost port number for Jenkins is changed to 8484 to prevent collision with the default port 8080. This is done in **server.xml** which can be found in the **conf** folder inside the tomcat. Inside server.xml, server port is changed **from 8005 to 8045**, http connector port is changed **from 8080 to 8484** and ajp connector port is changed **from 8009 to 8049.**

```
<!-- A "Connector" represents an endpoint by which requests are received
     and responses are returned. Documentation at :
     Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
     Java AJP  Connector: /docs/config/ajp.html
     APR (HTTP/AJP) Connector: /docs/apr.html
     Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
-->
<Connector port="8484" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
<!--
<Connector executor="tomcatThreadPool"
           port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
-->
<!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443
     This connector uses the NIO implementation that requires the JSSE
     style configuration. When using the APR/native implementation, the
     OpenSSL style configuration is required as described in the APR/native
     documentation -->
<!--
```

Figure 35. Changing port numbers in server.xml file

To run the tomcat server, one way is to navigate to **C:\apache\apache-tomcat-8.0.43-windows-x64\apache-tomcat-8.0.43\bin** with git bash and run the startup.sh script. To run the script, `./startup.sh` command is used inside git bash. Going to the URL **http:localhost:8484/jenkins** is launching Jenkins in the browser. When the Jenkins is started for the first time it is asking for administrator password. The password is found in the initialPassword file that Jenkins is creating, and it is located in **C:\Jenkins_data\secrets\initialPassword** like shown in the browser. Jenkins asks if any additional features are wanted to be installed with the Jenkins first use installer. The option to skip this is used by pressing the x on the upper corner of the window. This is because any additional features that is needed in the project are installed manually in this project. After the installation process is done Jenkins is opening in the Jenkins main page. To be able to use Jenkins an environmental variable needs to be set.

Code 7. Script jenkins_env.ps1 to set Jenkins environmental variable

```
setx JENKINS_HOME "C:\Jenkins_data"
```

To set the Jenkins environmental variable the above script is run in the power shell. First start power shell with an administrator rights and then type the script to the command line. This will set a user environmental variable name JENKINS_HOME and set the path to Jenkins_data.

Inside Jenkins the first thing to do is set the login password. This is done with pressing the admin button in the right upper corner of the main page and then pressing the configuration button in the left side list. From there a username and password can be set. Username admin and password admin are used in this project for simplicity. Next thing to do is to install a new plugin called **github plugin**. This is done with navigating to control Jenkins and control plugins and typing github plugin in the search field from the available tab. After checking github plugin and pressing install without restart the plugin is installed to Jenkins and can be used. With this plugin a git connection can be made with GitHub and Jenkins. The next thing to do is set the paths for Java JDK, Git and Maven inside Jenkins. This is done in the global configuration tools. From there in JDK section **localJDK** is set for the name of the path and the path in this project is **C:\Program Files\Java\jdk1.8.0_144**. The Git and Maven are set with the same process. Names are set to **localGit** and **localMaven** and maven path is **C:\Maven\apache-maven-3.5.0** and git path is **git.exe**. With git there is no need for setting the hole path, but simple the executable file inside the default path is enough. The default path for git executable is **C:\Program Files\Git\bin**. That is the last of the initial configurations in Jenkins.

To start connecting Jenkins with date-project the first thing is to start a new freestyle project with the name **date-storage-application**. The first thing to do in the configuration window is to set some description to the build. In this project the description for this build is “Jenkins build for date-storage-application”. In the source code management section git configurations are made. From the radio buttons **git** is selected and in the repositories section the project git URL is copied to the repository field. This is done by logging in the GitHub account and selecting date-storage-application from the repositories and then hitting the “green clone or download” button that shows the URL for the repository. In the build section **add build step** is pressed and the **invoke top-level Maven targets** is selected from the list of options. For the name of the maven version the Jenkins configurated **localMaven** is set and **clean package** is typed in the goal field. This is doing the same thing as the clean package when built inside eclipse. Last thing to do in the build section is to press **advanced** button and add the **pom** path to the pom field. In this project it is **date-project/pom.xml**. This depends on the project folder structure that is committed in to GitHub. To test this setup MongoDB needs to be up. Pressing the “**schedule a build for date-storage-application**” button in the right upper corner of the page is building the project inside Jenkins. After this build is done pressing the date-storage-application and then the build in the build history is showing the list of options. The console output is showing the same result as the clean package build inside eclipse and the tests are run. When the tests are run, also the date and time of the test are saved on the datedb-tests database. With this test the connection of Jenkins and the

date-project is confirmed. After this test the date-storage-application is configured again. In the build-triggers section the **poll scm** option is checked and in the schedule five stars separated by spaces are added like * * * * *. This is using the CRON rules and is checking the git repository for every minute for changes in the repository. This is changing the Jenkins to work with GitHub and run tests automatically when changes in the repository are occurring.

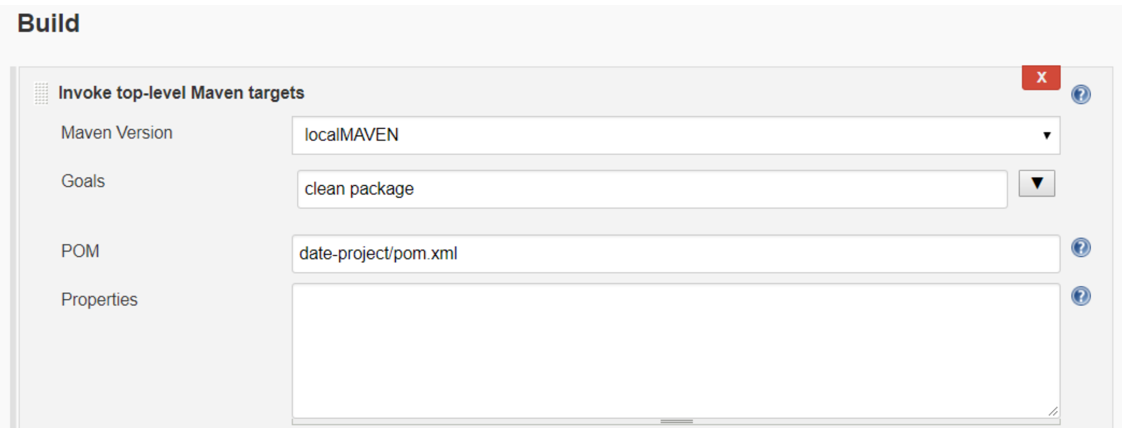

Figure 36. Maven build setup in date-storage-application Jenkins build

Next step is adding a plugin called checkstyle to check for any design flaws inside the date-project code. This is done by navigating plugins and installing a plugin called **checkstyle-plugin**. After this plugin is installed, the date-storage-application in Jenkins is again configured and in the build section a new goal is added in the goal field called **checkstyle:checkstyle**. After installing and configuring the checkstyle plugin date-storage-application is build using the build button. After the build is complete there is a new button in the list when pressing date-storage-application, the **checkstyle warnings** button. When using this new checkstyle button the checkstyle is showing all the design flaws that it is finding inside the code that is in the git repository. Major part of these warnings are Javadoc comment warnings that have been left out in the project for the checkstyle to find them or the lack of them. In this stage the warnings are solved inside the code. The design flaw free source code with Javadoc's added can be found in the git repository. The Javadoc comment message is showing in the method description when hovering over the method name. Some of the things that Javadoc comment can consists of are the description, parameter list, return value and author. When the code is now committed in to the repository it is noted that Jenkins build is run, and tests are performed to the date-storage-application.

The next step is to setup Jenkins to deploy the application to the server automatically when changes have been made in the code, the code is committed in the repository and all the

tests are run successfully without errors. The first thing to do is to add a new tomcat server that is the target server for the Jenkins build project. In this project tomcat-8.0.24 is used. There are several configurations that needs to be made to the tomcat server. In the **server.xml** the server port is changed **from 8005 to 8025**, the http connector port is changed **from 8080 to 8090** and ajp connector port is changed **from 8009 to 8029**. There is a bug that prevents second and other times of tomcat deployments to work and needs to be corrected by entering **context.xml** and adding **antiResourceLocking=”true”** to the context tag so the final thing would be <Context antiResourceLocking=”true”>.

And the last thing to configure is the roles, usernames and passwords for the users in this tomcat installation. **Tomcat-users.xml** is used for this configuration. There is a commented part in the end of the file right above tomcat-users end tag. This needs to be uncommented and changed in the following way. Two new roles are added, role **rolename=”manager-script”** and role **rolename=”admin-gui”**. Then a user is added with parameters, user **username=”tomcat” password=”tomcat” roles=”manager-script, admin-gui”**. Then the file is saved and closed.

```
  <role rolename="manager-script"/>
  <role rolename="admin-gui"/>
  <user username="tomcat" password="tomcat" roles="manager-script, admin-gui"/>
</tomcat-users>
```

Figure 37. Users and roles inside tomcat-users.xml

Inside Jenkins two plugins needs to be installed. **Copy artifact plugin** and **deploy to container plugin**. After installing these plugins additional configurations are made in the date-storage-application build. In the post-build section, a new post-build action named **archive the artifacts** is added to the list. In the files to archive field ****/*.war** is typed. This is for the Jenkins to collect the WAR file that is deployed in the browser. After this, another build is created in the Jenkins main page called **date-storage-application_deploy-to-staging**. In the build section of this new freestyle project a new build step is added that is called **copy artifacts from another project**. In the project name field a **date-storage-application** is added and in the artifacts to copy field ****/*.war** is typed. In the post-build actions, a new action is added with a name **Deploy war/ear to a container** and in the WAR/EAR files field ****/*.war** is typed. Context path date-storage is added in the context path field to change the application URL in a browser as localhost:8090/date-storage/ui. For the container a new container is added with **tomcat 7.x** (that also works with tomcat 8) and then the username and password are set to be tomcat. Then the tomcat 7 is selected from the list and to the URL ***http://localhost:8090*** is added.

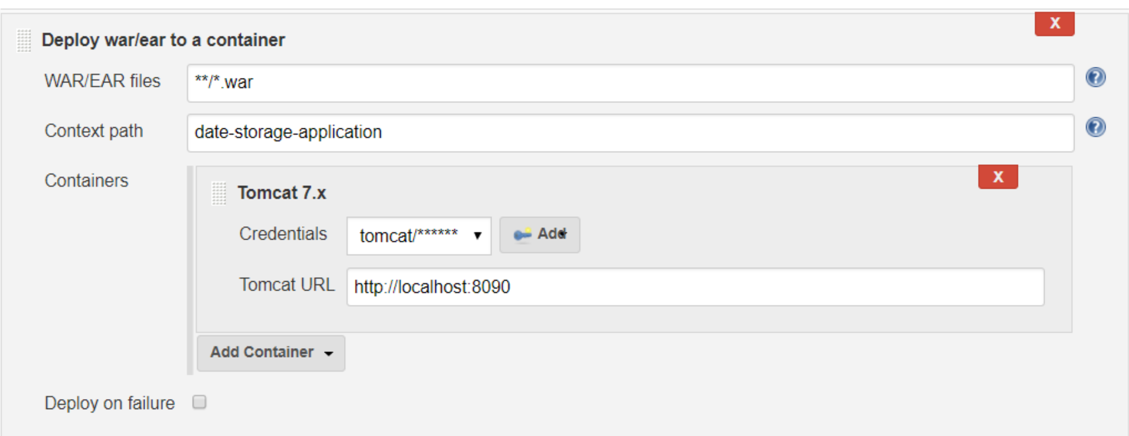

Figure 38. Deploy to container configurations

After this the main build, date-storage-application is configured with a new post-build action. Build other projects is selected from the list and **date-storage-application_deploy-to-staging** is added to the projects to build field. The **trigger only if build is stable** needs to be checked. After all this the tomcat-8.0.24 server is started, and a new build is scheduled in date-storage-application with the build button. After this successful build the date-project is build and launched in the browser in URL http://localhost:8090/date-storage/ui**.** This is the staging server that holds the application for further examination.

After the application is successfully deployed to the staging server a visual modification is added to Jenkins to make the build pipeline easier to follow. First a new plugin is installed called build **pipeline plugin**. After installing this plugin, a plus icon is added in to the main page of Jenkins at the top of the list of projects. Pressing this icon is starting a configuration window for the build pipeline. The name of the pipeline is **date-storage-application-pipeline** and the build pipeline view is checked. In the next page on the **select initial job** field **date-storage-application** is typed and other options are left as default. Now the date-storage-application pipeline is showing. To test this, run icon is pressed and the build process start as normal. The pipeline is a convenient way of following the progress of the build pipeline. Next thing is to change the checkstyle as a separate build that runs in parallel with date-storage-application_deploy-to-staging build. This is done by going to the date-storage-application build and removing checkstyle:checkstyle goal in the build section and removing the post-build action publish checkstyle analysis results. After this a new freestyle project is made with a **name date-storage-application_static-analysis**. In the source code management section, the date-storage-application git repository URL is added. Then a new build step is added in the build section named **invoke top-level Maven targets** and the goals is set to **checkstyle:checkstyle**. In the advanced section the **date-project/pom.xml** is added to the pom field. Then in the post-build section **publish checkstyle analysis report**

post-build is selected. Then in the date-storage-application post-build actions section in the projects to build field **date-storage-application_static-analysis** is added to addition of the date-storage-application_deploy-to-staging build. The process is then tested with pressing the run button in the pipeline view window.

To be able to deploy the application in to the production server, first a new tomcat server needs to be installed and configured. This is done a little different than before. The existing tomcat-8.0.24 server is copied to the same path of the parent folder of the conf, bin and lib folders. To be able to copy the tomcat folder the server needs to be closed first with a `./shutdown.sh` script that is run in the bin folder. Then the copied new folder is named **apache-tomcat-8.0.24-production** and a couple of configurations are made. First in the **server.xml** file the server port 8025 is changed to 8095, then the http connector port 8090 is changed to 9090 and then the ajp connector port 8029 is changed to 8099. All other configurations can be same as in the staging server. After this, both servers are started. In the Jenkins a new freestyle project is made with the name **date-storage-application_deploy-to-production**. In the configurations at the build section a new build step is added named **copy artifacts from another project**. The project name is **date-storage-application** and ****/*.war** is typed in the artifacts to copy field. Next a new post-build action is added in to the next section with name **deploy war/ear to a container**. In the WAR/EAR files ****/*.war** is typed, the context path is **date-storage** and a tomcat 7.x container is selected with same credentials. The URL for the production server is **http:localhost:9090/**. Lastly in the date-storage-application_deploy-to-staging build a new post-build action is added in the post-build actions section with name **build other projects (manual step)** and the downstream project name is **date-storage-application_deploy-to-production**. This is configurating the production server to be manually deployed to the server after all other build have been successfully run. To test this, run button in the pipeline view window is pressed. After pressing run button all other builds are run successfully, and the pipeline is stopping to wait for manual build for date-storage-application_deploy-to-production deployment. When this build button is pressed, the build is run, and the application is deployed to the production server.

Figure 39. Date-storage-application before deploying project to production server

To be able to spread the workload of a Jenkins master node to slave nodes, a distributed build solution is used. First a new slave node need to be made. This is done with navigating to manage **Jenkins -> Manage Nodes,** naming the slave node and selecting permanent agent from the checkbox options. The next window is the configuration window for the slave (also called as agent). The number of executors is defining how many executors there are to use for spreading the workload from the master Jenkins node. In this project the number of executors is defined as 2. Then the remote root directory is defined to where the slave node is distributing the load. In a windows machine it is usually selected to be something like C:\var\Jenkins. In this project it is defined to be **C:\var\Jenkins-date-dist**. In the usage options a use this node as much as possible is selected and the launch method is selected to be a as Java Web Start. Default options for Java Web Start are used. Availability option is also left as default option of keep this agent online as much as possible. The tool options are left as default, because Maven, Git and Java were configured in the Jenkins options and if not changed, Jenkins is using these options also in the slaves.

After the configurations are done the main page of Jenkins is showing a new executor in the executor list. The next step is to click the newly created executor and download agent.jar file in the local machine. Then the agent.jar is launched with the help of the command line script that is attached on the executor page. Agent.jar file needs to be in the same directory where the script is run to be able to connect to the slave node. After the script is run, the connection is successfully made.

Figure 40. Slave executor added in the executor list

The last thing is to change the configurations of the project that is using the distribution. In this project the test builds are distributed to the slave nodes. This is done by going to the configurations of the date-storage-application and marking the restrict where this project can be run checkbox option and adding the label name of the slave node in the text field. Other configurations can be left as they are.

After this is done the project is configurated for the slave node taking some of the workload from the master node. (The Complete Jenkins Course for Developers and DevOps, Udemy 2017. Jenkins Distributed Builds, Jenkins Wiki 2017.)

5.9 Connecting JIRA

Last thing to add in the development environment is the project management tool JIRA. This starts with downloading the JIRA software for a server in the working computer. JIRA is free for 30-day trial period. After the installer is downloaded in the local machine the next thing is to start the installation. In installation options a custom installation is selected, and the path is left as default. In the TCP port configuration, a custom value option is selected, and the port numbers are set to ***8383*** and ***8035***. JIRA is installed as a service, so it is starting every time the system starts. After the installation process is complete navigating to ***localhost:8383/*** is starting the configuration process. **"I'll set it up myself"** option is selected and in the database setup the **built-in** option is selected. For the application title **Date storage application** is used and the **private** option is selected from the mode radio buttons for only the administrators to be able to create new users. In the **"specify your license key"** part it is necessary to **create a new myattlassian account** to generate a JIRA trial license key. After creating a new account and logging in, the "new evaluation license" window opens, and the license key is generated. In these configurations a server license type is selected, the organization name is set to **Date storage application** and the **"up and**

running" button is selected in the "your instance is running" radio buttons. After these changes a generate license key is pressed and it is generated for this project. In the next page an administrator account is setup. After setting the administrator information's the **configure email notifications** option is set to **later** and language is set to **English** and avatar is chosen. A new scrum project is created with a name **date-project JIRA** and the key is automatically set to DJ. The next thing is to install a git integration for JIRA plugin in JIRA from the Atlassian marketplace.

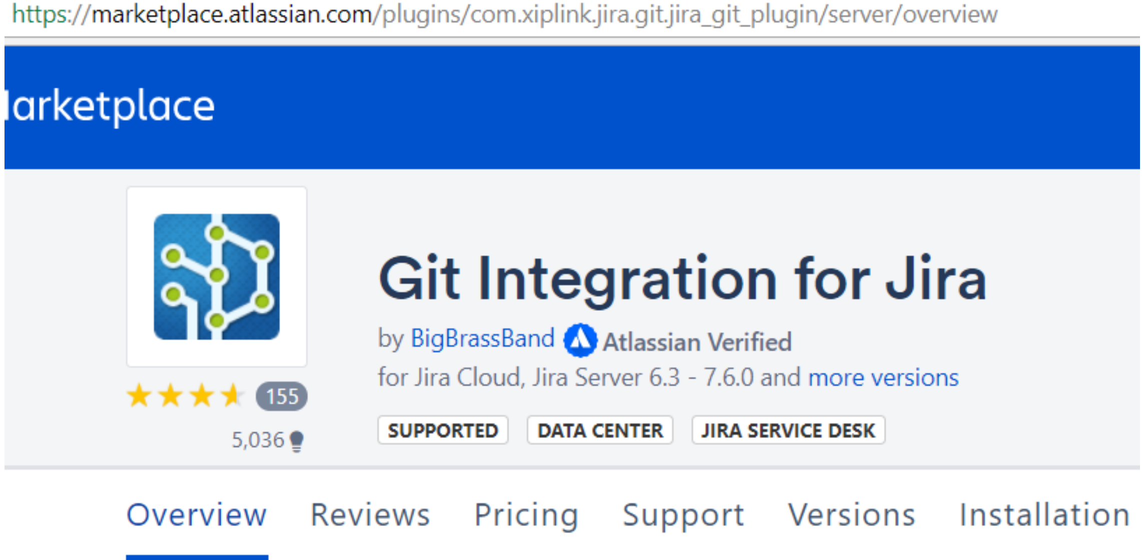

Figure 41. Git integration for JIRA in the Atlassian marketplace

The server is selected in the dropdown of choices of cloud and server and the **try it free** button is pressed. The next page is the first page of the new evaluation license creation phase. In this page the organization name is set and in this project the name is **Date storage application**. Then after accepting the terms of use for Atlassian marketplace the generate license button is pressed. The generated license key is copied to the clipboard and the download add-on button is pressed for the **jira_git_plugin** JAR file to be downloaded in to the local machine. As Jira administrator in the settings the **add-on** tab is pressed and then **manage-add-ons**. From this page the **upload add-on** is pressed, and the downloaded JAR file is searched from the computer. This is bringing the add-on in JIRA and installing it. The next thing is to use the license key. This is done in the manage- add-ons page and the **git integration for Jira** is selected from the list of add-ons. Then the license key is pasted in the license key field and the add-on is updated. After the installation is complete, a git repository that is used, is connected to JIRA with git integration plugin. The smart commit option and repository browser options are left as enabled. Smart commits are the solution for inputting

JIRA issue updates directly from the git commit messages. For this project a simple issue is made, and the smart commit is configured so the projects git repository and project management tool are in sync and working together. The next thing is to create new issue in JIRA. This is done by pressing **issue** button on top of the main page. The summary for the new issue is **Minor log message fixes**. The summary should be descriptive, so the purpose of the issue is clear by looking the issue name. Second plugin that needs to be installed for the smart commits to work is **Jira Fisheye/Bitbucket Server plugin**. This is also downloaded from Atlassian marketplace. This is done by selecting the server option and clicking the **get it now button**. This is downloading the JAR file to the local machine. Then in the **manage add-ons** section in settings add-on tab the upload add-on button is pressed and the **jira_fisheye_plugin** JAR file is searched from computer and the installation is complete. After installing the fisheye plugin, fisheye needs to be downloaded to the local machine. After the basic installer is complete with default installation paths, navigating to URL ***localhost:8060*** is starting fisheye. When starting fisheye by navigating to that URL, the first thing to do is to generate new license. This is done by pressing **Obtain evaluation license** from the first page after launching fisheye and generating the license from there. In the next part the option to **include crucible as a part of the evaluation** is checked. Connecting JIRA at this point is skipped and then administrator password for fisheye is set.

Fisheye is now setup. The next step is to add the date-storage-application repository to fisheye. By pressing the add repository button in fisheye, the next page is a repository configuration page. The repository type is git, display name is set to **date-storage-application** and description is set for the repository. Then the repository location is copied from the GitHub account and for authentication **password for http(s)** is used with GitHub account password. In the final page the default options are left in place. It is very important that in the smart commits section in fisheye there is JIRA issue transitions and review creations checkboxes checked. Without those, smart commit is not working. Setting up the application link between JIRA and fisheye is the next step. To do this a new application link is created in applications tabs applications links URL field. In this field the fisheye URL is given as http://localhost:8060/ From the link applications page the “The servers have the same set of users and usernames” is left unchecked and the **“I am administrator on both instances”** is checked. After pressing continue the page is redirected to fisheye where the new user is prompted to be created for fisheye.

Add New User

Information about the new user

Username Fish1

Display name Fish1_fisu

Email jiratest87@gmail.com

Password ••••••••••

Confirm password ••••••••••

Add Cancel

Figure 42. Creating user in fisheye

After creating the first user as admin in fisheye the application link is set up in fisheye also. Adding the JIRA URL http://localhost:8383/ and marking all the same check boxes in the link applications page is setting up the application link. The application link is creating a config error and to solve this, the pen symbol in the actions is used to configure the connection. Changing the local authentication for outgoing and incoming **directions to OAuth from OAuth (impersonation)** is solving the problem. After this, a user mapping for the repository and the users needs to be made and this is done in fisheye user mappings section as administrator. To log in as administrator, at the bottom of the fisheye page there is an administrator link that redirect the page to fisheye administrator login page. Selecting the repository and pressing add in the **user mappings section** is giving the options to choose the committer and the user for fisheye that are used in providing the link for that user. In this project committer name is turc0ss as the GitHub account owner and the user is the user that is made in the fisheye. Also, the same this is also made with a committer tapsa that is the eclipse user for git plugin.

After this in the repositories section a new application link needs to be made between the repository added in the fisheye and the project in JIRA. To do this, the fisheye user that was used in the user mappings needs to be logged in as a user for fisheye. In the repositories section clicking the **gear symbol** at the end of the repository is opening repository links page. **Add** link on the top corner is pressed and the JIRA project which is wanted to be connected to the repository is clicked. In this project that is **date-storage-application**. Next prompt is asking for authentication and the **authorize** button is pressed. In the next page fisheye is

requesting read and write access to the data on localhost. By pressing **allow** is prompting the last window that is showing the projects that are available, and the **date-storage-application** is selected and **yes** option is selected and **create** button is pressed. After this the smart commit link has been made. To test this a correction for the log message string is made so that the log message in test, is not hard coded, but inside the enum class, and then the change is committed to the repository. The way of the commit message is different is **DJ-3 #comment Fixed logging messages #time 10m #done**. This message is using the smart commit keywords for JIRA to be able to pick up the commit from the repository with fisheye and change the issue that is referenced in the commit message accordingly. The test is a success and the smart commit changed the issue status according to the commit message.

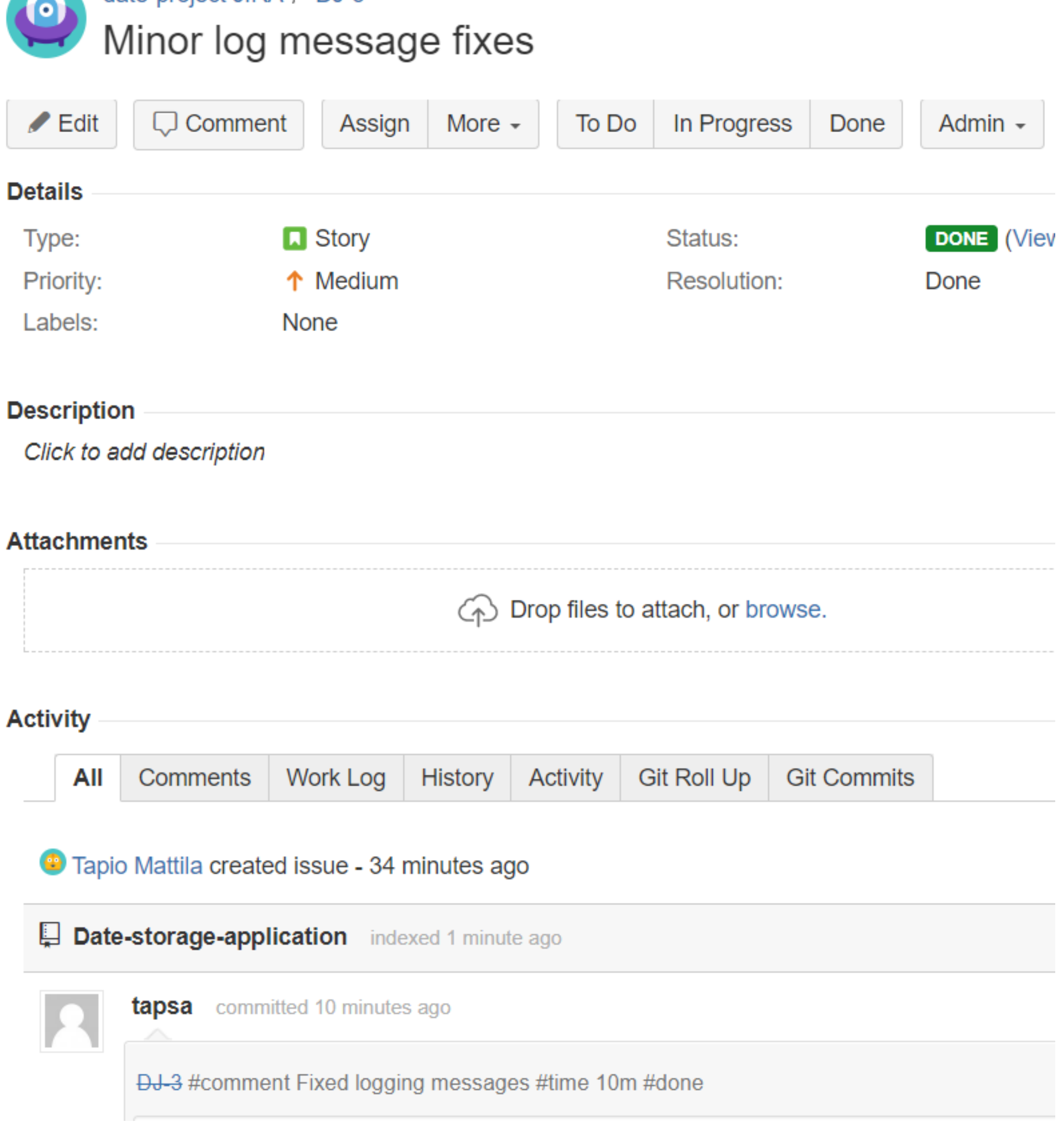

Figure 43. JIRA issue status changed with git commit message

~~DJ-3~~ #comment Fixed logging messages #time 10m #done

Date-storage-application master 9597ca447310dabf53bb801bb4b901056313d3fc

+3 ±1 - date-project/src/main/java/com/tmattila/utils/LoggingMessages.java

+1 ±3 - date-project/src/test/java/test/com/tmattila/datetorepositorytest/DateToRepositoryTest.java

+4 ±4 -

late-project/src/main/java/com/tmattila/utils/LoggingMessages.java

```
      15 lines hidden
16 16   DATE_SAVE_ERROR("Error saving date to database"),
17 17   DATES_OBJECT_CREATED("New Dates object created"),
18 18   SAVEDATETOREPOSITORY_ENTER("Enter saveDateToRepository()"),
19    -  SAVEDATETOREPOSITORY_EXIT("Exit saveDateToRepository()");
   19 +  SAVEDATETOREPOSITORY_EXIT("Exit saveDateToRepository()"),
   20 +  ENTER_REPOSITORY_TEST("Enter repository test"),
   21 +  REPOSITORY_TEST_COMPLETE("Test complete"),
   22 +  EXIT_REPOSITORY_TEST("Exit test");
20 23
21 24   /***
22 25    * Set the string variable to use.
      19 lines hidden
```

late-project/src/test/java/test/com/tmattila/datetorepositorytest/DateToRepositoryTe

```
      18 lines hidden
19 19  import com.tmattila.model.Dates;
20 20  import com.tmattila.repository.DateRepository;
21 21  import com.tmattila.utils.DateStringUtils;
   22 +import com.tmattila.utils.LoggingMessages;
22 23
23 24  @RunWith(SpringRunner.class)
24 25  @SpringBootTest(classes = App.class)
      14 lines hidden
39 40
40 41   @Test
41 42   public void test() {
42    -   logger.debug("Enter repository test");
```

Figure 44. JIRA smart commit showing the changes that were made to the code

5.10 How it all work together?

Now the development environment is made, and all of its parts are connected to each other. There are several different parts that have been configured. At this stage it is good to re-examine what are those parts and how they all fit together.

First the application was created on the local machine. With this simple application many of the features that would be used in the much larger scale project were introduced and configured, such as Spring and Vaadin frameworks, JUnit testing and logging with log4j2. The application was connected to git repository that holds all of the source code and is performing version control for the application. The database was created and configured in such way that it is reliable and scalable and was connected to the application. When all of these were done, the application was connected with Jenkins that holds all the functionality for automated testing and deployment on the staging and production servers and also handling distributed builds for the application, so it can be deployed on different platforms and spread the load of a master node to slave nodes. Lastly JIRA project management tool was connected to the project and smart commit was configured so that git repository commit messages could change the issue states automatically without the need of the developer going in to JIRA and separately changing them.

The key point of the environment is the git repository that is connecting the application, Jenkins and JIRA together. From the IDE, the source code is committed to git repository. The commit message is interpreted by JIRA that is connected to the git repository and if it detects JIRA keywords that are used for smart commit, it is changing the given issue accordingly. When the source code is pushed in to the git repository, Jenkins is detecting that change in the repository and is performing appropriate tests that were configured in the application. Jenkins is also performing code style check for the code that was pushed for any improvements that it could find. When all of the tests are passed, Jenkins is deploying the application in the staging server where it can be manually deployed to the production server.

The complete production development environment that was made is portable to any other application. Git, Jenkins, MongoDB and JIRA are programming language independent and do not rely on the application infrastructure.

6 CONCLUSION OF THE PROJECT AND FINAL THOUGHTS

The goal of this thesis work was to build a full-stack software development environment and by doing so, learn main elements and tools used in a development field. The environment was built with known building blocks that developers use daily in their projects.

By successfully creating the environment and connecting the elements to each other, authors expertise about the system was elevated in another level. By achieving this goal, the project can be considered to be a success.

Project work was also made to be completely repeatable with the step by step instructions of building the environment chapter. This is going to help any future developer that would like to increase the expertise and know how about these subjects to achieve his or her end goal faster.

Built development environment can be used to create applications on wide variety of different use cases. The actual deployment into a development process of the project work remains to be seen in the future.

REFERENCES

Agile software development, Wikipedia 2018. Referred 20.1.2018 https://en.wikipedia.org/wiki/Agile_software_development

Apache Maven, Wikipedia 2017. Referred 8.12.2017 https://en.wikipedia.org/wiki/Apache_Maven

Apache Tomcat, Wikipedia 2017. Referred 16.12.2017 https://en.wikipedia.org/wiki/Apache_Tomcat

Application framework, Wikipedia 2017. Referred 9.12.2017 https://en.wikipedia.org/wiki/Application_framework

Application server, Wikipedia 2017. Referred 13.12.2017 https://en.wikipedia.org/wiki/Application_server

Atlassian Fisheye overview video (YouTube), by Atlassian 2010. Referred 21.12.2017 https://www.youtube.com/watch?v=lH5BTEa9Nws

Atlassian main page, Crucible 2017. Referred 21.12.2017 https://www.atlassian.com/software/crucible

Atlassian main page, Fisheye 2017. Referred 21.12.2017 https://www.atlassian.com/software/fisheye

Atlassian main page, JIRA 2017. Referred 20.12.2017 https://www.atlassian.com/software/jira

Binary repository manager, Wikipedia 2017. Referred 12.12.2017 https://en.wikipedia.org/wiki/Binary_repository_manager

Book of Vaadin 2017, chapter 1.1. Referred 20.12.2017

Client-server model, Wikipedia 2018. Referred 20.1.2018 https://en.wikipedia.org/wiki/Client%E2%80%93server_model

Database, Wikipedia 2017. Referred 13.12.2017 https://en.wikipedia.org/wiki/Database

Definition of relational database, WhatIs.com 2006. Referred 13.12.2017 http://searchsqlserver.techtarget.com/definition/relational-database

Description for Maven, Apache Maven main page 2017. Referred 7.12.2017 https://maven.apache.org/

Distributed builds, Jenkins wiki 2017. Referred 20.12.2017 https://wiki.jenkins.io/display/JENKINS/Distributed+builds

Eclipse, Wikipedia 2017. Referred 7.12.2017 https://en.wikipedia.org/wiki/Eclipse_(software)

Everything you need to know about sharding (11:19 – 22:27, YouTube), by MongoDB 2014. Referred 21.12.2017 https://www.youtube.com/watch?v=W3HhqMvwZP8&t=1091s

File server, Wikipedia 2017. Referred 13.12.2017 https://en.wikipedia.org/wiki/File_server

Git main page 2017. Referred 16.12.2017 https://git-scm.com/

Git, Wikipedia 2017. Referred 16.12.2017 https://en.wikipedia.org/wiki/Git

Introduction to the build lifecycle, Apache Maven introduction page 2017. Referred 8.12.2017 https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Iterative design, Wikipedia 2018. Referred 20.1.2018 https://en.wikipedia.org/wiki/Iterative_design

Java platform enterprise edition, Wikipedia 2017. Referred 17.12.2017 https://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

Java programming language, Wikipedia 2017. Referred 7.12.2017 https://en.wikipedia.org/wiki/Java_(programming_language)

Jenkins Master CI for DevOps and Developers section 1 lecture 5, Udemy 2017. Referred 16.12.2017 https://www.udemy.com/the-complete-jenkins-course-for-developers-and-devops/learn/v4/t/lecture/6159084?start=0

Jenkins software, Wikipedia 2017. Referred 16.12.2017 https://en.wikipedia.org/wiki/Jenkins_(software)

JIRA in a Nutshell demo (YouTube), by Atlassian 2012. Referred 20.12.2017 https://www.youtube.com/watch?v=xrCJv0fTyR8

Log4j Logging Levels, Apache Logging Docs 2012. Referred 10.12.2017 https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Level.html

Logging and log4j.properties explained, by Brandan Jones 2015 (YouTube). Referred 10.12.2017 https://www.youtube.com/watch?v=-GkRuFU_sUg&t=39s

Manage user and roles, MongoDB docs 2017. Referred 22.12.2017

https://docs.mongodb.com/manual/tutorial/manage-users-and-roles/

Master Jenkins CI for DevOps and Developers section 1 lecture 6, Udemy 2017. Referred 20.11.2017 https://www.udemy.com/the-complete-jenkins-course-for-developers-and-devops/learn/v4/t/lecture/6159120?start=0

MongoDB authentication, MongoDB docs 2017. Referred 22.12.2017

https://docs.mongodb.com/manual/core/authentication/

MongoDB overview, Tutorialspoint 2017. Referred 18.12.2017 https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

MongoDB, Wikipedia 2017. Referred 18.12.2017 https://en.wikipedia.org/wiki/MongoDB

Most popular web application testing tools, Softwaretestinghelp.com 2017. Referred 20.1.2018 http://www.softwaretestinghelp.com/most-popular-web-application-testing-tools/

Project management software, Capterra.com 2018. Referred 20.12.2017 https://www.capterra.com/project-management-software/

Proxy server, Wikipedia 2017. Referred 13.12.2017 https://en.wikipedia.org/wiki/Proxy_server

Quick introduction to spring framework section 4 lecture 16, Udemy 2017. Referred 19.12.2017 https://www.udemy.com/spring-tutorial-for-beginners/learn/v4/t/lecture/8068606?start=0

Relational database management system, Wikipedia 2017. Referred 13.12.2017 https://en.wikipedia.org/wiki/Relational_database_management_system

Relational vs. non-relational database, Pluralsight 2014 by Jacqueline Homan. Referred 15.12.2017 https://www.pluralsight.com/blog/software-development/relational-non-relational-databases

Replication in MongoDB, MongoDB docs 2017. Referred 20.12.2017 https://docs.mongodb.com/manual/replication/

Replica set members, MongoDB docs 2017. Referred 20.12.2017 https://docs.mongodb.com/manual/core/replica-set-members/

SDLC iterative model, tutorialspoint 2017. Referred 10.12.2017 https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

SDLC overview, tutorialspoint 2017. Referred 10.12.2017 https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

SDLC spiral model, tutorialspoint 2017. Referred 10.12.2017 https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm

SDLC waterfall model, tutorialspoint 2017. Referred 10.12.2017 https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Simplifying Java Development chapter 1.1, Spring in Action 4th Edition 2014. Referred 19.12.2017 https://livebook.manning.com/#!/book/spring-in-action-fourth-edition/chapter-1/8

Server computing, Wikipedia 2017. Referred 20.1.2018 https://en.wikipedia.org/wiki/Server_(computing)

Server definition, Whatis.com 2017. Referred 12.12.2017 http://whatis.techtarget.com/definition/server

Sharding in MongoDB, MongoDB docs 2017. Referred 21.12.2017 https://docs.mongodb.com/v3.0/core/sharding-introduction/

Software framework, Wikipedia 2017. Referred 9.12.2017 https://en.wikipedia.org/wiki/Software_framework

Spiral model, Wikipedia 2017. Referred 20.1.2018 https://en.wikipedia.org/wiki/Spiral_model

Spring Boot Maven Plugin, Spring Boot docs 2017. Referred 20.11.2017 https://docs.spring.io/spring-boot/docs/current/reference/html/build-tool-plugins-maven-plugin.html

Systems development lifecycle, Wikipedia 2018. Referred 20.1.2018 https://en.wikipedia.org/wiki/Systems_development_life_cycle

Top 8 continuous integration tools, Vladimir Pecanac, code-maze.com 2016. Referred 16.12.2017 https://www.code-maze.com/top-8-continuous-integration-tools/

Using JIRA for requirements management, Confluence.atlassian.com 2017. Referred 20.1.2018 https://confluence.atlassian.com/jirakb/using-jira-for-requirements-management-193300521.html

V – model (software development), Wikipedia 2018. Referred 20.1.2018 https://en.wikipedia.org/wiki/V-Model_(software_development)

Version control systems distributed vs centralized, Oshyn blog 2012, Diego Vergara. Referred 14.12.2017 https://www.oshyn.com/blogs/2012/june/version_control_systems__distributed_vs__centralized

Version control, Wikipedia 2017. Referred 14.12.2017 https://en.wikipedia.org/wiki/Version_control

Waterfall model, Wikipedia 2017. Referred 20.1.2018 https://en.wikipedia.org/wiki/Waterfall_model

Appendix 1. Source code of the date-storage application

pom.xml

```
1.  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.or
    g/2001/XMLSchema-instance"
2.      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.
    org/xsd/maven-4.0.0.xsd">
3.      <modelVersion>4.0.0</modelVersion>
4.      <groupId>com.tmattila</groupId>
5.      <artifactId>date-project</artifactId>
6.      <version>0.0.1-SNAPSHOT</version>
7.
8.      <packaging>war</packaging>
9.
10.     <parent>
11.         <groupId>org.springframework.boot</groupId>
12.         <artifactId>spring-boot-starter-parent</artifactId>
13.         <version>1.5.8.RELEASE</version>
14.     </parent>
15.
16.     <prerequisites>
17.         <maven>3</maven>
18.     </prerequisites>
19.
20.     <properties>
21.         <java.version>1.8</java.version>
22.         <vaadin.version>8.1.6</vaadin.version>
23.         <vaadin.plugin.version>8.1.6</vaadin.plugin.version>
24.         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
25.         <maven.compiler.source>1.8</maven.compiler.source>
26.         <maven.compiler.target>1.8</maven.compiler.target>
27.     </properties>
28.
29.     <dependencyManagement>
30.         <dependencies>
31.             <dependency>
32.                 <groupId>com.vaadin</groupId>
33.                 <artifactId>vaadin-bom</artifactId>
34.                 <version>${vaadin.version}</version>
35.                 <type>pom</type>
36.                 <scope>import</scope>
37.             </dependency>
38.             <dependency>
39.                 <groupId>org.apache.logging.log4j</groupId>
40.                 <artifactId>log4j-bom</artifactId>
41.                 <version>2.10.0</version>
42.                 <scope>import</scope>
43.                 <type>pom</type>
44.             </dependency>
45.         </dependencies>
46.     </dependencyManagement>
47.
48.     <dependencies>
49.         <dependency>
50.             <groupId>org.springframework.boot</groupId>
51.             <artifactId>spring-boot-starter-web</artifactId>
52.         </dependency>
53.         <dependency>
54.             <groupId>org.springframework.boot</groupId>
55.             <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
56.             <scope>provided</scope>
57.         </dependency>
58.         <dependency>
59.             <groupId>com.vaadin</groupId>
60.             <artifactId>vaadin-spring-boot-starter</artifactId>
61.         </dependency>
62.         <dependency>
63.             <groupId>com.vaadin</groupId>
64.             <artifactId>vaadin-spring</artifactId>
65.         </dependency>
66.         <dependency>
67.             <groupId>org.springframework</groupId>
68.             <artifactId>spring-context</artifactId>
69.         </dependency>
70.         <dependency>
71.             <groupId>org.apache.logging.log4j</groupId>
72.             <artifactId>log4j-api</artifactId>
73.         </dependency>
74.         <dependency>
75.             <groupId>org.apache.logging.log4j</groupId>
76.             <artifactId>log4j-core</artifactId>
77.         </dependency>
78.         <dependency>
79.             <groupId>org.springframework.boot</groupId>
80.             <artifactId>spring-boot-starter-data-mongodb</artifactId>
81.         </dependency>
82.         <dependency>
83.             <groupId>junit</groupId>
84.             <artifactId>junit</artifactId>
85.         </dependency>
86.         <dependency>
87.             <groupId>org.springframework.boot</groupId>
88.             <artifactId>spring-boot-starter-test</artifactId>
89.             <scope>test</scope>
90.         </dependency>
91.     </dependencies>
92.
93.     <build>
94.         <plugins>
95.             <plugin>
96.                 <groupId>org.springframework.boot</groupId>
97.                 <artifactId>spring-boot-maven-plugin</artifactId>
98.                 <configuration>
99.                     <executable>true</executable>
100.                 </configuration>
101.             </plugin>
102.         </plugins>
103.     </build>
104.
105. </project>
```

App.java

```
1.  package com.tmattila;
2.
3.  import org.springframework.boot.SpringApplication;
4.  import org.springframework.boot.autoconfigure.SpringBootApplication;
5.  import org.springframework.boot.builder.SpringApplicationBuilder;
6.  import org.springframework.boot.web.support.SpringBootServletInitializer;
7.  import org.springframework.data.mongodb.repository.config.EnableMongoRepositor
    ies;
8.  import org.springframework.web.bind.annotation.RequestMapping;
9.  import org.springframework.web.bind.annotation.RestController;
```

```
10.
11. /***
12.  * Date storage application.
13.  * @author Tapio Mattila
14.  *
15.  */
16. @SpringBootApplication
17. @EnableMongoRepositories({"com.tmattila"})
18. public class App extends SpringBootServletInitializer {
19.
20.     /***
21.      * Main method for the application.
22.      * @param args
23.      */
24.     public static void main(final String[] args) {
25.         SpringApplication.run(App.class, args);
26.     }
27.
28.     @Override
29.     protected final SpringApplicationBuilder configure(final SpringApplication
    Builder builder) {
30.         return builder.sources(App.class);
31.     }
32. }
```

Dates.java

```
1.  package com.tmattila.model;
2.
3.  import java.text.DateFormat;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Date;
6.
7.  import org.springframework.data.annotation.Id;
8.  import org.springframework.data.mongodb.core.mapping.Document;
9.  import org.springframework.data.mongodb.core.mapping.Field;
10.
11. /***
12.  * Dates model with a mongodb @Document annotation to declare collection name
    to dates.
13.  * @author Tapio Mattila
14.  *
15.  */
16. @Document(collection = "dates")
17. public class Dates {
18.
19.     /***
20.      * String id, db field value "id".
21.      */
22.     @Id
23.     private String id;
24.
25.     /***
26.      * String title, db field value "Title:".
27.      */
28.     @Field(value="Title:")
29.     private String title;
30.
31.     /***
32.      * Variable Date for setting new date.
33.      */
34.     private Date date;
35.
```

```
36.     /***
37.      * String formattedDate, db field value "Date:".
38.      */
39.     @Field(value = "Date:")
40.     private String formattedDate;
41.
42.     /***
43.      * Dates constructor.
44.      */
45.     public Dates() {
46.
47.     }
48.     /***
49.      * Getter for the title.
50.      * @return title
51.      */
52.     public final String getTitle() {
53.         return title;
54.     }
55.
56.     /***
57.      * Setter for the title.
58.      * @param title
59.      */
60.     public final void setTitle(final String title) {
61.         this.title = title;
62.     }
63.
64.     /***
65.      * Getter for the date.
66.      * @return date
67.      */
68.     public final Date getDate() {
69.         return this.date;
70.     }
71.
72.     /***
73.      * Setter for the date.
74.      * @param date
75.      */
76.     public final void setDate(final Date date) {
77.         this.date = new Date();
78.     }
79.
80.     /***
81.      * Getter for the formatted date.
82.      * @return formattedDate
83.      */
84.     public final String getFormattedDate() {
85.         return this.formattedDate;
86.     }
87.
88.     /***
89.      * Setter for the formatted date.
90.      * @param formattedDate
91.      */
92.     public final void setFormattedDate(final String formattedDate) {
93.         Date date = new Date();
94.         DateFormat dateF = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
95.         this.formattedDate = dateF.format(date);
96.     }
97.
98.     @Override
```

```
99.     public final String toString() {
100.            return "Dates [title=" + title + ", date=" + formattedDate + "]"
    ;
101.        }
102.    }
```

DateRepository.java

```
1.  package com.tmattila.repository;
2.
3.  import java.util.Date;
4.
5.  import org.springframework.data.mongodb.repository.MongoRepository;
6.  import org.springframework.stereotype.Repository;
7.
8.  import com.tmattila.model.Dates;
9.
10. /***
11.  * DateRepository interface that uses the functionality of MongoRepository.
12.  * @author Tapio Mattila
13.  *
14.  */
15. @Repository
16. public interface DateRepository extends MongoRepository<Dates, Date>{
17.
18. }
```

DateService.java

```
1.  package com.tmattila.service;
2.
3.  import com.tmattila.model.Dates;
4.
5.  /***
6.   * DateService interface
7.   * @author Tapio Mattila
8.   *
9.   */
10. public interface DateService {
11.     /***
12.      * saveDateToRepository method that uses Dates model as a parameter
13.      * @param dateDAO
14.      */
15.     void saveDateToRepository(Dates dateDAO);
16. }
```

DateServiceImpl.java

```
1.  package com.tmattila.service;
2.
3.  import org.apache.logging.log4j.LogManager;
4.  import org.apache.logging.log4j.Logger;
5.  import org.springframework.beans.factory.annotation.Autowired;
6.  import org.springframework.stereotype.Service;
7.
8.  import com.tmattila.model.Dates;
9.  import com.tmattila.repository.DateRepository;
10. import com.tmattila.utils.DateStringUtils;
11. import com.tmattila.utils.LoggingMessages;
12.
13. /***
```

```
14.  * DateServiceImpl service class that handles the connection to database.
15.  * @author Tapio Mattila
16.  *
17.  */
18. @Service
19. public class DateServiceImpl implements DateService {
20.
21.     /***
22.      * Logger class initialization for DateServiceImpl class.
23.      * This class is used to print log messages.
24.      */
25.     private static final Logger logger = LogManager.getLogger(DateServiceImpl.
    class);
26.
27.     /***
28.      * Autowired DateRepository interface to use repository methods inside Dat
    eServiceImpl classs
29.      */
30.     @Autowired
31.     private DateRepository dateRepository;
32.
33.     @Override
34.     public final void saveDateToRepository(final Dates dateDAO) {
35.
36.         logger.debug(LoggingMessages.SAVEDATETOREPOSITORY_ENTER.getString());

37.
38.         Dates dates = new Dates();
39.         logger.info(LoggingMessages.DATES_OBJECT_CREATED.getString());
40.         dates.setTitle(DateStringUtils.DATE_TITLE.getString());
41.         dates.setFormattedDate(dateDAO.getFormattedDate());
42.         logger.info(dates.toString());
43.         dateRepository.save(dates);
44.
45.         logger.debug(LoggingMessages.SAVEDATETOREPOSITORY_EXIT.getString());
46.     }
47. }
```

MainUI.java

```
1.  package com.tmattila.ui;
2.
3.  import java.text.DateFormat;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Date;
6.
7.  import org.apache.logging.log4j.LogManager;
8.  import org.apache.logging.log4j.Logger;
9.  import org.springframework.beans.factory.annotation.Autowired;
10.
11. import com.tmattila.model.Dates;
12. import com.tmattila.service.DateService;
13. import com.tmattila.utils.DateStringUtils;
14. import com.tmattila.utils.LoggingMessages;
15. import com.vaadin.annotations.Theme;
16. import com.vaadin.annotations.Title;
17. import com.vaadin.server.VaadinRequest;
18. import com.vaadin.shared.ui.ContentMode;
19. import com.vaadin.spring.annotation.SpringUI;
20. import com.vaadin.ui.Button;
21. import com.vaadin.ui.HorizontalLayout;
22. import com.vaadin.ui.Label;
23. import com.vaadin.ui.UI;
```

```
24. import com.vaadin.ui.VerticalLayout;
25.
26. /***
27.  * MainUI class that holds all the ui related components. Extends vaadin UI cl
    ass.
28.  * @author Tapio Mattila
29.  *
30.  */
31. @SuppressWarnings("serial")
32. @Theme("valo")
33. @Title("Date Project")
34. @SpringUI(path = "/ui")
35. public class MainUI extends UI {
36.
37.     /***
38.      * Logger class initialization for MainUI class.
39.      * This class is used to print log messages.
40.      */
41.     private static final Logger logger = LogManager.getLogger(MainUI.class);
42.
43.     /***
44.      * MainLayout component rootLayout.
45.      */
46.     private VerticalLayout rootLayout;
47.
48.     /***
49.      * HorizontalLayout that holds the headerLayout.
50.      */
51.     private HorizontalLayout headerLayout;
52.
53.     /***
54.      * Own horizontalLayout to hold buttonLayout.
55.      */
56.     private HorizontalLayout buttonLayout;
57.
58.     /***
59.      * HorizontalLayout that holds dateMarkings.
60.      * Is used to show the dates in the browser.
61.      */
62.     private HorizontalLayout dateMarkings;
63.
64.     /***
65.      * Label that holds header text.
66.      */
67.     private Label headerLabel;
68.
69.     /***
70.      * Button that is used to print dates to browser and save to database.
71.      */
72.     private Button dateButton;
73.
74.     /***
75.      * Date class that is recording the currect time and date.
76.      */
77.     private Date date;
78.
79.     /***
80.      * DateFormat class that uses SimpleDateFormat to format the date in custo
    m form.
81.      */
82.     private DateFormat dateF = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
83.
84.     /***
```

```
85.       * Dates entity that holds the information that is saved in to database.
86.       */
87.      private Dates dates;
88.
89.      @Override
90.      protected final void init(final VaadinRequest request) {
91.
92.          logger.debug(LoggingMessages.START_PROGRAM.getString());
93.
94.          dates = new Dates();
95.
96.          rootLayout = new VerticalLayout();
97.          headerLayout = new HorizontalLayout();
98.
99.          headerLabel = new Label("<h2><b>" + DateStringUtils.HEADER_TEXT.getStr
    ing() + "</b></h2>", ContentMode.HTML);
100.             headerLayout.addComponent(headerLabel);
101.
102.             buttonLayout = new HorizontalLayout();
103.             buttonLayout.setMargin(false);
104.             buttonLayout.setSpacing(false);
105.
106.             dateButton = new Button(DateStringUtils.BUTTON_TEXT.getString())
    ;
107.             dateMarkings = new HorizontalLayout();
108.
109.             dateButton.addClickListener(e -> {
110.
111.                 try {
112.                     buildLayout();
113.                 } catch (Exception error) {
114.                     logger.error(LoggingMessages.DATE_PRINT_ERROR.getString(
    ));
115.                     error.printStackTrace();
116.                 }
117.
118.             });
119.
120.             buttonLayout.addComponent(dateButton);
121.
122.             rootLayout.addComponent(headerLabel);
123.             rootLayout.addComponent(buttonLayout);
124.             setContent(rootLayout);
125.         }
126.
127.         /***
128.          * Autowired DateService interfac to be able to save dates in to dat
    abase through service class.
129.          */
130.         @Autowired
131.         private DateService dateService;
132.
133.         /***
134.          * BuildLayout method to seperate layout creation to its own method.

135.          */
136.         private void buildLayout() {
137.
138.             logger.debug(LoggingMessages.DATE_PRINT_ENTER.getString());
139.
140.             date = new Date();
141.             String formattedDate = dateF.format(date);
142.
```

```
143.                Label timeLabel = new Label("<b>" + DateStringUtils.DATE_TEXT.ge
    tString() + " </b>", ContentMode.HTML);
144.                Label dateLabel = new Label(timeLabel.getValue() + formattedDate
    , ContentMode.HTML);
145.
146.                saveDate();
147.
148.                dateMarkings.addComponent(dateLabel);
149.                rootLayout.addComponent(dateLabel);
150.
151.                logger.info(LoggingMessages.DATE_PRINT.getString() + ": " + form
    attedDate);
152.                logger.debug(LoggingMessages.DATE_PRINT_EXIT.getString());
153.            }
154.
155.            /***
156.             * SaveDate method to seperate saving data to database on its own me
    thod and to include try catch block to check errors.
157.             */
158.            private void saveDate() {
159.                try {
160.                    dateService.saveDateToRepository(dates);
161.                    logger.debug(LoggingMessages.DATE_SAVED_TO_DB.getString());

162.                } catch (Exception e) {
163.                    e.printStackTrace();
164.                    logger.error(LoggingMessages.DATE_SAVE_ERROR.getString());
165.                }
166.            }
167.        }
```

DateStringUtils.java

```
1.  package com.tmattila.utils;
2.
3.  /***
4.   * Enum class DateStringUtils that holds the string representations that are u
    sed in the application.
5.   * @author Tapio Mattila
6.   *
7.   */
8.  public enum DateStringUtils {
9.
10.     HEADER_TEXT("Welcome to date storage application "),
11.     BUTTON_TEXT("Click to save the time of button press to database"),
12.     DATE_TEXT("Date and time of button press: "),
13.     DATE_TITLE("DATE"),
14.     DATE_TEST_TITLE("TEST");
15.
16.     /***
17.      * Set the string variable to use.
18.      */
19.     private final String string;
20.
21.     /***
22.      * DateStringUtils constructor.
23.      * @param string
24.      */
25.     private DateStringUtils(final String string) {
26.         this.string = string;
27.     }
28.
29.     /***
```

```
30.      * GetString method to show the string representation.
31.      * @return string
32.      */
33.     public String getString() {
34.         return string;
35.     }
36. }
```

LoggingMessages.java

```
1.  package com.tmattila.utils;
2.
3.  /***
4.   * Enum class LoggingMessages that holds the string representations that are u
    sed in the application.
5.   * @author Tapio Mattila
6.   *
7.   */
8.  public enum LoggingMessages {
9.
10.     START_PROGRAM("Start program"),
11.     DATE_PRINT("Date created and printed on the browser"),
12.     DATE_PRINT_ERROR("Something went wrong, error printing date to browser"),

13.     DATE_PRINT_ENTER("Enter the button press, buildLayout() method."),
14.     DATE_PRINT_EXIT("Exit buildLayout()"),
15.     DATE_SAVED_TO_DB("Date saved to database"),
16.     DATE_SAVE_ERROR("Error saving date to database"),
17.     DATES_OBJECT_CREATED("New Dates object created"),
18.     SAVEDATETOREPOSITORY_ENTER("Enter saveDateToRepository()"),
19.     SAVEDATETOREPOSITORY_EXIT("Exit saveDateToRepository()"),
20.     ENTER_REPOSITORY_TEST("Enter repository test"),
21.     REPOSITORY_TEST_COMPLETE("Test complete"),
22.     EXIT_REPOSITORY_TEST("Exit test");
23.
24.     /***
25.      * Set the string variable to use.
26.      */
27.     private final String string;
28.
29.     /***
30.      * DateStringUtils constructor.
31.      * @param string
32.      */
33.     private LoggingMessages(final String string) {
34.         this.string = string;
35.     }
36.
37.     /***
38.      * GetString method to show the string representation.
39.      * @return string
40.      */
41.     public String getString() {
42.         return string;
43.     }
44. }
```

application.properties

```
1.  spring.application.name=date storage application
2.
3.  #Mongo config
```

```
4.  spring.data.mongodb.host=tapio-dell-xps
5.  spring.data.mongodb.port=20120
6.  spring.data.mongodb.database=datedb
7.  
8.  #Server configuration
9.  spring.data.mongodb.authentication-database=admin
10. 
11. #User configuration
12. spring.data.mongodb.password=mongo123
13. spring.data.mongodb.username=tapioR
```

log4j2.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Configuration status="WARN">
3.  
4.      <Properties>
5.          <Property name="filename">C:/date-project-
    logging/logging.log</Property>
6.      </Properties>
7.  
8.      <Appenders>
9.          <File name="File" fileName="${filename}">
10.             <PatternLayout>
11.                 <pattern>%d{yyyy-MM-dd HH:mm:ss} %-
    5p %C{1}:%L [%t] - %msg%n</pattern>
12.             </PatternLayout>
13.         </File>
14.         <Console name="Console" target="SYSTEM_OUT">
15.             <PatternLayout>
16.                 <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-
    5level %C{1}:%L [%t] - %msg%n</pattern>
17.             </PatternLayout>
18.         </Console>
19.     </Appenders>
20. 
21.     <Loggers>
22.         <Root level="debug">
23.             <AppenderRef ref="Console" />
24.             <AppenderRef ref="File" />
25.         </Root>
26.     </Loggers>
27. 
28. </Configuration>
```

DateToRepositoryTest.java

```
1.  package test.com.tmattila.datetorepositorytest;
2.  
3.  import java.text.DateFormat;
4.  import java.text.SimpleDateFormat;
5.  import java.util.Date;
6.  
7.  import org.apache.logging.log4j.LogManager;
8.  import org.apache.logging.log4j.Logger;
9.  import org.junit.After;
10. import org.junit.Assert;
11. import org.junit.Before;
12. import org.junit.Test;
13. import org.junit.runner.RunWith;
14. import org.springframework.beans.factory.annotation.Autowired;
15. import org.springframework.boot.test.context.SpringBootTest;
```

```
16. import org.springframework.test.context.junit4.SpringRunner;
17.
18. import com.tmattila.App;
19. import com.tmattila.model.Dates;
20. import com.tmattila.repository.DateRepository;
21. import com.tmattila.utils.DateStringUtils;
22. import com.tmattila.utils.LoggingMessages;
23.
24. @RunWith(SpringRunner.class)
25. @SpringBootTest(classes = App.class)
26. public class DateToRepositoryTest {
27.
28.     private static final Logger logger = LogManager.getLogger(DateToRepository
    Test.class);
29.
30.     @Autowired
31.     DateRepository dateRepository;
32.
33.     @Before
34.     public void setUp() throws Exception {
35.     }
36.
37.     @After
38.     public void tearDown() throws Exception {
39.     }
40.
41.     @Test
42.     public void test() {
43.         logger.debug(LoggingMessages.ENTER_REPOSITORY_TEST.getString());
44.         Date date = new Date();
45.         DateFormat dateF = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
46.         String formattedDate = dateF.format(date);
47.         Dates dates = new Dates();
48.         dates.setTitle(DateStringUtils.DATE_TEST_TITLE.getString());
49.         dates.setFormattedDate(formattedDate);
50.         Assert.assertNotNull(dateRepository.save(dates));
51.         logger.info(formattedDate);
52.         logger.info(LoggingMessages.REPOSITORY_TEST_COMPLETE.getString());
53.         logger.debug(LoggingMessages.EXIT_REPOSITORY_TEST.getString());
54.     }
55. }
```

application.properties

```
spring.application.name=date storage application

#Mongo config
spring.data.mongodb.host=tapio-dell-xps
spring.data.mongodb.port=20120
spring.data.mongodb.database=datedb-tests

#Server configuration
spring.data.mongodb.authentication-database=admin

#User configuration
spring.data.mongodb.password=mongo123
spring.data.mongodb.username=tapioR
```

Appendix 2. Environmental variable scripts

set_env_variables.ps1

```
setx JAVA_HOME "C:\Program Files\Java\jdk1.8.0_144"
setx M3 "C:\Maven\apache-maven-3.5.0"
setx M3_HOME "C:\Maven\apache-maven-3.5.0\bin"
setx GIT_HOME "C:\Program Files\Git"

setx /S tapio-dell-xps JAVA_HOME "C:\Program Files\Java\jdk1.8.0_144"
/M
setx /S tapio-dell-xps M3_HOME "C:\Maven\apache-maven-3.5.0" /M
setx /S tapio-dell-xps GIT_HOME "C:\Program Files\Git" /M
setx /S tapio-dell-xps MONGO_HOME "C:\Program Files\MongoDB\Server\3.4"
/M

Start-Job -FilePath "C:\date-project-script\java_path.ps1"
Start-Job -FilePath "C:\date-project-script\maven_path.ps1"
Start-Job -FilePath "C:\date-project-script\git_path.ps1"
Start-Job -FilePath "C:\date-project-script\mongodb_path.ps1"
```

java_path.ps1

```
[Environment]::SetEnvironmentVariable("Path", $env:Path + ";C:\Program
Files\Java\jdk1.8.0_144\bin", [EnvironmentVariableTarget]::Machine)
```

maven_path.ps1

```
[Environment]::SetEnvironmentVariable("Path",        $env:Path        +
";C:\Maven\apache-maven-3.5.0\bin",
[EnvironmentVariableTarget]::Machine)
```

git_path.ps1

```
[Environment]::SetEnvironmentVariable("Path", $env:Path + ";C:\Program
Files\Git\bin", [EnvironmentVariableTarget]::Machine)
```

mongodb_path.ps1

```
[Environment]::SetEnvironmentVariable("Path", $env:Path + ";C:\Program
Files\MongoDB\Server\3.4\bin", [EnvironmentVariableTarget]::Machine)
```

jenkins_env.ps1

```
setx JENKINS_HOME "C:\Jenkins_data"
```

Appendix 3. MongoDB database scripts

create_folders.sh

```
mkdir shard-a/
mkdir shard-b/
mkdir shard-c/
mkdir shard-d/
mkdir log
mkdir shard-a/shard0
mkdir shard-a/shard1
mkdir shard-a/shard2
mkdir shard-b/shard0
mkdir shard-b/shard1
mkdir shard-b/shard2
mkdir shard-c/shard1
mkdir shard-c/shard0
mkdir shard-c/shard2
mkdir shard-d/shard0
mkdir shard-d/shard1
mkdir shard-d/shard2
mkdir conf-serv
mkdir conf-serv/conf-0
mkdir conf-serv/conf-1
mkdir conf-serv/conf-2
mkdir keyfile
```

create_keyfile.sh

```
openssl rand -base64 741 > /c/date-project-db/keyfile/keyfile
```

start_conf-serv.sh

```
mongod --replSet conf-serv --logpath C:/date-project-db/log/conf-serv0.log --dbpath C:/date-project-db/conf-serv/conf-0 --port 26050 --configsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet conf-serv --logpath C:/date-project-db/log/conf-serv1.log --dbpath C:/date-project-db/conf-serv/conf-1 --port 26051 --configsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet conf-serv --logpath C:/date-project-db/log/conf-serv2.log --dbpath C:/date-project-db/conf-serv/conf-2 --port 26052 --configsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

start_shard-a.sh

```
mongod --replSet a --logpath C:/date-project-db/log/shard-a0.log --dbpath C:/date-project-db/shard-a/shard0 --port 27001 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

```
mongod --replSet a --logpath C:/date-project-db/log/shard-a1.log --dbpath C:/date-project-db/shard-a/shard1 --port 27002 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet a --logpath C:/date-project-db/log/shard-a2.log --dbpath C:/date-project-db/shard-a/shard2 --port 27003 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

start_shard-b.sh

```
mongod --replSet b --logpath C:/date-project-db/log/shard-b0.log --dbpath C:/date-project-db/shard-b/shard0 --port 27101 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet b --logpath C:/date-project-db/log/shard-b1.log --dbpath C:/date-project-db/shard-b/shard1 --port 27102 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet b --logpath C:/date-project-db/log/shard-b2.log --dbpath C:/date-project-db/shard-b/shard2 --port 27103 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

start_shard-c.sh

```
mongod --replSet c --logpath C:/date-project-db/log/shard-c0.log --dbpath C:/date-project-db/shard-c/shard0 --port 27201 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet c --logpath C:/date-project-db/log/shard-c1.log --dbpath C:/date-project-db/shard-c/shard1 --port 27202 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet c --logpath C:/date-project-db/log/shard-c2.log --dbpath C:/date-project-db/shard-c/shard2 --port 27203 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

start_shard-d.sh

```
mongod --replSet d --logpath C:/date-project-db/log/shard-d0.log --dbpath C:/date-project-db/shard-d/shard0 --port 27301 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet d --logpath C:/date-project-db/log/shard-d1.log --dbpath C:/date-project-db/shard-d/shard1 --port 27302 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &

mongod --replSet d --logpath C:/date-project-db/log/shard-d2.log --dbpath C:/date-project-db/shard-d/shard2 --port 27303 --shardsvr --smallfiles --keyFile C:/date-project-db/keyfile/keyfile &
```

start_mongos.sh

```
mongos --logpath C:/date-project-db/log/mongos.log --configdb conf-serv/tapio-dell-xps:26050,tapio-dell-xps:26051,tapio-dell-xps:26052 --port 20120 --keyFile C:/date-project-db/keyfile/keyfile &
```

Appendix 4. Configuring Jenkins

Tomcat 8.0.43 is extracted to apache folder

Jenkins.war is downloaded from internet https://updates.jenkins-ci.org/download/war/

Latest version

Jenkins.war is copied and pasted to webapps folder in tomcat-8.0.45

in the bin folder of tomcat-8.0.43 folder (jenkins) (after copy pasting the jenkins.war file in to the webapps folder)

Git bash console is opened in this folder

./startup.sh

-> Jenkins is running.

Initial Jenkins port 8080 is changed to 8484

URL localhost:8484/jenkins

Browser is asking for initial password

Password is got from C:\Jenkins Data\secrets\initialPassword

Pasted in to the textfield

Ok

X is pressed to skip the plugins install phase.

In Jenkins

Admin (on the right top corner) and then configure to change the initialpassword

admin

Sign out

(Test it)

admin, admin

Next initial configurations are made on Jenkins

Control jenkins

Overall configaration

Conf JDK

localJDK

Java path

conf MAVEN

localMAVEN

Maven path

Jenkins configuration

Install plugins

Available plugins:

Github plugin, is typed

Overall jenkins configuration

Conf git

localGIT

git.exe

Install plugins:

Available

Unleash maven, is typed

Unleash maven plugin, install without restart

Freestyle build is started

Date-storage-project

Description: Jenkins build for the date storage project

Check: Loose old translations

Translations to keep: 7

Source code management:

Check: Git

Repository URL: URL from the git repository is given (clone or download section in GitHub)

Build section:

Maven version: localMAVEN

Goals: clean package

Save

Mongodb instance needs to be running in port 20120

When it is up

Date-storage-project

Build now

MongoDB instance is opened:

In the test collection of dates database there should be a test date saved

Jenkins build was a successful.

After this, date-storage-project is configured again

Jenkins is made to pull changes automatically from git repository whenever changes in the repository is detected

Configurations:

Build Triggers:

Check: Poll SCM

* * * * *, is typed

This will check the repository every minute

Save

Next checkstyle is installed to check the quality of the code base.

Jenkins main page.

Configure jenkins

Install plugins

Available

checkstyle plugin

Install without restart

To the Jenkins main page

Date-storage-project

Configure

BUILD

Goals: clean package checkstyle:checkstyle

POST-BUILD ACTIONS:

Publish Checkstyle analysis results

Save

Date-storage-project

Build now

Page is refreshed after the build is ran

New tab is added

Checkstyle warnings

(Check the warnings and resolve them)

Now the project is working with automatic testing

Next step is to make the project to work with continuous delivery to staging server and to the production server.

First to archive the project artifacts

Date-storage-project

Configure

Post-BUILD SECTION

Add post-build action

Archive the artifacts

Files to archive: **/*.war

Save

Date-storage-project

Trigger a new build

Build now

There is a new information about the build artifacts after the build is complete

Date-storage-project-0.0.1-SNAPSHOT.war is showing

Console output

Last thing on the console output should be

Archiving artifacts

Finished: SUCCESS

To deploy the artifact to staging URL (area), tomcat need to be configured

(First tomcat server needs to be downloaded in to local machine)

Tomcat-8.0.24 is downloaded and extracted in to apache folder in C drive

In server.xml

Following things needs to be changed:

Tomcat 8.0.24 staging port changed:

```
<Server port="8005" shutdown="SHUTDOWN">
-->
<Server port="8025" shutdown="SHUTDOWN">

 <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
--->
```

```
<Connector port="8090" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
-->
<Connector port="8029" protocol="AJP/1.3" redirectPort="8443" />
```

Save

In context.xml (to prevent the second + other times for crashing when running the program)

```
<Context antiResourceLocking="true">
```

Save

In tomcat-users.xml

Roles and passwords are changed in the following way

OLD

```
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

NEW

```
<role rolename="manager-script"/>
              <role rolename="admin-gui"/>
              <user username="tomcat" password="tomcat" roles=
              "manager-script, admin-gui"/>
```

</tomcat-users>

Save

In jenkins

Couple of new plugins are installed

Copy artifact plugin

Deploy to container plugin

Then

Another Jenkins job is created to deploy the artifact to tomcat

(Freestyle project)

Name: date-storage-project_deploy-to-staging

Loose old translations

-> 7

Build section

BUILD:

Add build step

-> Copy artifacts from another project

Project name: date-storage-project

Artifacts to copy: **/*.war

Post-build actions:

Deploy war/ear to a container

WAR/EAR files: **/*.war

Context path: date-storage

Containers: add Container:

Tomcat 7.x (will also work with tomcat 8)

Add

Username: tomcat

Password: tomcat

Ok

From credetial list: tomcat 7

Tomcat URL: http://localhost:8090

Save

Main jenkins page

Date-storage-project

Post-build Actions

Add post build action

Build other projects

Projects to build: deploy-to-staging_maven-udemy-project

(Trigger only if build is stable (checked))

Save

Tomcat staging server is started

(From git bash ./startup.sh)

Main jenkins page

Date-storage-project build is scheduled from the Jenkins main page

After that is successfully built, the date-storage-project_deploy-to-staging will run

Date-storage-project_deploy-to-staging is pressed

Build is successful, and it's been triggered by date-storage-project

Jenkins build-pipeline:

Plugins, install:

Build pipeline plugin

Jenkins main page

Plus tab is pressed next to all tab to add a view

Name: build pipeline

(Build Pipeline View, is checked)

Ok

Initial job:

Date-storage-project

(Others are left as default)

Ok

(Pipeline is showing now)

To test pipeline

Run

Now the pipeline is working such that the testing and the checkstyle is going to run first and then the deployment to tomcat container staging area is going to run.

Pipeline is next changed so that the checkstyle is running parallel to the deployment of staging area. (The checkstyle is not test, but the quality check so this is acceptable.)

Configure page of the date-storage-project

Build section: checkstyle:checkstyle, is removed

In post-build action:

Publish checkstyle analysis results, is removed

Save

New item is created that only runs checkstyle

Main page

New item.

Name: date-storage-project_static-analysis

Source Code Management section:

Git.

Date-storage-project URL (https://github.com/turc0ss/date-storage-project.git)

Build section:

Add build step, Invoke top-level Maven targets.

Maven version: localMaven

Goals: checkstyle:checkstyle

Advanced (option)

POM: date-storage-project/pom.xml

Post-build: publish checkstyle analysis report.

Save

Date-storage-project

Configure

Post-build-actions:

Projects to build: date-storage-project_deploy-to-staging, date-storage-project_static-analysis,

Save

Main page:

Build pipeline view tab is pressed next to the all tab

To test the work in progress run is pressed

Deploy to Production

Tomcat-8.0.24 server is shutdown

Bin folder

(In git bash)

./shutdown.sh

Git bash (in bin folder) is closed

Tomcat 8.0.24 folder is renamed to

Tomcat 8.0.24-staging

Folder is copied

Copy is renamed to tomcat 8.0.24-production

Tomcat 8.0.24 production ports are changed:

```
<Server port="8025" shutdown="SHUTDOWN">
-->
<Server port="8095" shutdown="SHUTDOWN">

 <Connector port="8090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
--->
<Connector port="9090" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />

<Connector port="8029" protocol="AJP/1.3" redirectPort="8443" />
-->
<Connector port="8099" protocol="AJP/1.3" redirectPort="8443" />
```

Both servers are started (staging and production)

Jenkins main page:

New job

Name: date-storage-project_deploy-to-production

Freestyle project

Build section:

Add build step -

Copy artifacts from another project

Project name: date-storage-project

Artifacts to copy: **/*.war

Post build action section:

Add post-build action -

Deploy war/ear to a container

WAR/EAR files: **/*.war

Context path: date-storage

Containers: Tomcat 7.x

Same credentials

URL: http://localhost:9090/

Save

Jenkins main page:

Date-storage-project_deploy-to-staging

Configuration

Post build actions:

Add post-build action

Build other projects (manual step)

Downstream Project Names: date-storage-project_deploy-to-production

Save

Jenkins main page

Build pipeline tab

Run

All the other builds have completed and gone green

The last build is triggered manually so it's not run

Run is pressed on the last build

To check that all the servers are running

localhost:8090/

localhost:9090/

To check that the application is working on both servers

http://localhost:8090/server-test/dateui

http://localhost:9090/server-test/dateui