Tuisku Salminen

Digital Signing of Documents as a Service

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

10.11.2016

Metropolia

Author(s) Title Number of Pages Date	Tuisku Salminen Digital Signing of Documents as a Service 34 pages 10 November 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kimmo Sauren, Lecturer Tero Hakkarainen, Software Engineer

The goal of this thesis was to create a web application which would allow digital signing to take over physical signing of documents in Metropolia University of Applied Sciences.

The modern development world is full of different software frameworks and development tools for different programming languages. This thesis aimed to describe the software technologies and developing tools used in the process of creating a service for digital signing of documents, as well as to give a general idea of the whole signing process before and after the implementation of this system.

This project was developed with Java and its software framework JSF 2.0. In order to make the front-end look better for the user, a framework called Bootstrap was used to generate a theme for the application. The database was created with the help of a database management system, MySQL. The project's version controlling was done with the help of Apache Subversion. The project was built and its libraries were managed by Apache Maven and the project was eventually hosted on an Apache Tomcat –instance.

As a result of this project, a functional web application was created to take advantage of digital signing in graduate diplomas.

Keywords	software development, development tools, digital signature

Metropolia

Tekijä Otsikko Sivumäärä Aika	Tuisku Salminen Digitaalisen allekirjoituksen toteutus palveluna 34 sivua 10.11.2016
Tutkinto	Bachelor of Engineering
Koulutusohjelma	Information Technology
Suuntautumisvaihtoehto	Software Engineering
Ohjaajat	Lehtori Kimmo Sauren Ohjelmistokehittäjä Tero Hakkarainen

Insinöörityön tarkoituksena oli luoda verkkosovellus, joka mahdollistaisi digitaalisen allekirjoituksen käyttämisen fyysisen allekirjoituksen sijasta dokumenteissa Metropolia Ammattikorkeakoulussa.

Nykyinen maailma on täynnä erilaisia sovelluskehyksiä ja kehitystyökaluja eri ohjelmointikielille. Insinöörityössä perehdyttiin ohjelmistoteknologioihin ja kehitystyökaluihin, joita käytettiin digitaalisen allekirjoituksen mahdollistavan palvelun kehityksessä. Insinöörityöraportti esittelee allekirjoitusprosessin ennen työssä luotua palvelua ja palvelun luomisen jälkeen.

Verkkosovellus kehitettiin Javalla ja sen ohjelmistokehyksellä JSF 2.0:lla. Jotta käyttäjälle näkyvästä puolesta saatiin paremman näköinen, ohjelmistokehystä nimeltä Bootstrap käytettiin ohjelmiston teeman luonnissa. Ohjelmiston tietokanta luotiin tietokantaohjelmisto MySQL:n avulla. Projektin versiohallintaan käytettiin Apache Subversionia. Sovellus koottiin ja sen kirjastoja hallinnoitiin Apache Mavenin avulla. Lopulta sovellus asennettiin toimimaan Apache Tomcat -instanssiin.

Insinöörityön tuloksena syntyi toiminnallinen verkkosovellus, jonka avulla digitaalista allekirjoitusta pystytään hyödyntämään valmistuvien opiskelijoiden tutkintotodistuksessa.

Avainsanat	ohjelmistokehitys, kehitystyökalut, digitaalinen allekirjoitus

Metropolia

Contents

1 Introduction

There are approximately 3000 students graduating yearly from Metropolia University of Applied Sciences. [9] A complete list of the degree programmes in Metropolia includes the following: Arts and Culture, Business and Electrical Engineering, Civil Engineering and Conservation, Health Services, Media and ICT, Technology and Design and Well-being. In addition to these, Metropolia also offers Open Studies, which give one an opportunity to build up the facts and skills for work-life, study as a hobby and / or generally help you in your possible future studies. Each one of the graduation diplomas is hand signed by the president of Metropolia as well as a few other personnel of Metropolia included in the process, such as the head of the graduate's study program.

Modern computer science has made it possible to sign documents digitally rather than by hand. Digital signing is the exact same thing as the traditional way of signing documents by hand, but the actual signing is done digitally via a web service for example. The signing system needs to meet the demands set in law to make the signed document valid legally. The Finnish government has developed a service for this that goes by the name "Vetuma". If a company wants to make use of Vetuma, they have to make a contract with the government for the rights to use it. [4]

1.1 Objectives and the Assignment

The goal of this project was to design and implement digital signing software for Metropolia University of Applied Sciences. The developed software was initially set to be used by the president of Metropolia to sign the diplomas through a web service. The service was designed in the way that the digital signing was a separate module, so the use of it could be extended to other documents as well. The authentication of the signer is handled via Vetuma-service through the use of an HST card. The abbreviation HST comes from Finnish phrase "Henkilön Sähköinen Tunnistaminen", which stands for "Digital Identifying of a Person". The whole project consisted of first designing the software in theory and through illustrations, such as workflow diagrams and database concepts and afterwards implementing the design in practice as a web service. As the project was intended to only be available to the staff and students of Metropolia, the project took advantage of a service already implemented in Metropolia called CAS-login. This fulfills the required

necessary properties for a strong authentication method, which is required by law for the kind of services that involve digital signing of legally valid documents.

1.2 Structure of this Document

In the thesis I will first go through the required demands for the project and the most important part of the project, the signature. A brief explanatory of what a framework is and the used frameworks in this project will be given. The service's current state and the future state will then be discussed. After this, the thesis will introduce the developing tools used in this project and a possible alternative to one used tool. In the end, the future plans for the applications will be given briefly and a conclusion to the thesis.

The technologies and tools used will be discussed briefly to keep the thesis in a manageable size. The introduced technologies and tools will have examples, such as written code, to help the reader in getting a basic understanding of the given technology or tool.

2 Electronic Signature

2.1 Demands for the Signature

The signature has to meet the demands set in law by the Finnish government. To make the service viable in replacing the old printed version of a diploma, the document has to be created in a special PDF-form, PDF/A. This makes the document legally valid and in this way it can be used for long term preservation.

2.2 Strong Authentication

Strong authentication is done in the electronic signature service via Metropolia's Central Authentication Service, CAS-login. CAS itself is a form of strong authentication, so there was no need to generate a separate specific strong authentication login for this service or use the one provided by Vetuma.

Metropolia's CAS-login service is used widely across different services from Metropolia. It is a single sign-on, SSO-service. Once the user is logged in, he/she can use every system and service that uses the same SSO-service without the need to log in again, as long as the given CAS-ticket is valid.

2.3 Central Authentication Service

The main purpose of CAS is to permit the user to access all affiliated applications while only providing their credentials once. The whole service basically involves three different sections and can be summarized as follows: At first, the client's browser is redirected to a CAS-authentication website. After the website is loaded and the user has submitted his credentials, the browser sends an authentication request to the associated CAS server. The server then checks the given credentials and other possible data and confirms the validity of them. And finally, in a successful login, the server gives back a valid service ticket that can be used to authenticate the used browser to every associated CAS-secured webpages.

There are multiple different CAS-library implementations available for multiple different programming languages. The one used in this project was called "Jasig CAS Client".

2.4 Vetuma-service

Vetuma, stands for “Verkkotunnistautuminen ja –maksaminen” in Finnish, is a service designed and implemented by Fujitsu Finland Oy. It was developed for Finnish citizens to help with the use of e-services, such as online authentication and online payments. The person using any of the services provided by Vetuma can authenticate themselves by one of three options: Online banking id’s, mobile diploma or by a HST-card, which stands for “henkilön sähköinen tunnistus” in Finnish, equipped with a microchip. However, with online banking id’s used as the authentication method, service used in this project, the undisputed electric signature, cannot be used. This is a limitation set by Fujitsu Finland Oy, the company behind Vetuma. At its current state, only a HST-card can be used as the method to give a valid undisputed electric signature through Vetuma. As stated above, Vetuma also supports functionalities such as authentication and online payments. [4]

2.5 PDF/A Format

PDF/A is a standardized version of PDF, which has a few subcategories. The three subcategories of PDF/A-standard are:

- PDF/A-1
- PDF/A-2
- PDF/A-3

From these the PDF/A-1-version has two subcategories of its own, which are PDF/A-1a and PDF/A-1b. Each of the subclasses is standardized in a different way, but every one of them can be used for long term preservation, which makes them able to be stored as legally valid documents. This means that any document that is made to fit these standards can be digitally preserved for a set period of time without the need for a paper version of the document. [5]

Every future iteration of the PDF/A version has to be backwards compatible. This means that even if the document was created as PDF/A valid at the release of PDF/A, for example, in 2005, it will still be valid in the future, no matter how much time passes. Even if the documents made, for example, in 2012 have to follow different standards, the PDF/A valid documents made in 2005 will still be valid. In general, the PDF/A format can

be summarized as a format that was created to get rid of the need for paper copies around the world and effectively conserve nature because the sheer amount of paper documents in the world is appalling.

The summary of the specifications for PDF/A-valid documents are:

- the fonts used in the document have to be embedded
- the document must use a specified color-set, which has to be set in the documents metadata
- the document must have its author/creator, title and different set of information set in metadata, depending on the document's content. [5]

3 Frameworks

A software framework is defined as a platform for developing software applications. It provides a sort of sandbox, which the developer can use to create different functionalities for their software. For example, a framework can contain ready-made classes and functions or methods, which the user can use to create something they need. A fine example of such is a framework called JavaServer Faces, JSF, which was used in this project to build the user interface through the back-end's Java code. More of JSF and its features will be discussed in more detail in section 3.1.

3.1 JavaServer Faces 2.0

JavaServer Faces, JSF, was developed as a J2EE-standard for building server-side user interfaces. in this way the whole user interface can be built from Java code by generating JavaScript using JSF's built-in methods. This tends to bind the front-end and the back-end to the same project, which can seem quite old-fashioned in modern software development. More of this subject in section 3.1.1.

In all simplicity, the creation of a really basic Hello, me web application can be summarized in three listings, listings 1-3. Listing 1 represents the first view the user using the application gets in the browser. It is a fairly simple web page with a single text input –box for typing the user's name in. When clicking the "welcome"-button, the entered value is sent to a JavaBean for storing. After submitting the page, the user is sent to the screen represented in code-form in listing 2. The page shown has a text saying "Welcome" and gives the submitted text as an addition to the welcome-text. In order to get the container to understand the deployed project as a JSF-project, the user needs to create a deployment descriptor, web.xml, which is pictured in listing 3. In this file, the user maps all the necessary elements from the project under their correct xml-elements, such as the first page to display to the user.

```
?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">

    <h:head>
        <title>JSF 2.0 Test Project</title>
    </h:head>
    <h:body>
      <h2>JSF 2.0 Test Project</h2>
      <h:form>
         <h:inputText value="#{testBean.name}"></h:inputText>
         <h:commandButton value="Welcome Me" action="welcome"></h:commandButton>
      </h:form>
    </h:body>
</html>
```

Listing 1. An example of JSF 2.0 input-page.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">

    <h:head>
      <title>JSF 2.0 Test Project</title>
    </h:head>
    <h:body bgcolor="white">
      <h2>JSF 2.0 Test Project</h2>
      <h2>Welcome #{testBean.name}</h2>
    </h:body>
</html>
```

Listing 2. An example of a JSF 2.0 after submitting a value.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
      http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      id="WebApp_ID" version="2.5">

      <display-name>JavaServerFaces</display-name>

      <!-- Change to "Production" when going live -->
      <context-param>
            <param-name>javax.faces.PROJECT_STAGE</param-name>
            <param-value>Development</param-value>
      </context-param>

      <!-- Welcome page -->
      <welcome-file-list>
            <welcome-file>faces/test.xhtml</welcome-file>
      </welcome-file-list>

      <!-- JSF mapping -->
      <servlet>
            <servlet-name>Faces Servlet</servlet-name>
            <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>

      <!-- Map these files with JSF -->
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>/faces/*</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>*.jsf</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>*.faces</url-pattern>
      </servlet-mapping>
      <servlet-mapping>
            <servlet-name>Faces Servlet</servlet-name>
            <url-pattern>*.xhtml</url-pattern>
      </servlet-mapping>

</web-app>
```

Listing 3. An example of JSF 2.0 web.xml-file.

3.2 Reasons for and Against JSF

The user interface is a combination of generated JavaScript, HTML and CSS-files from the Java code. In this way of developing web applications is gradually being replaced by calling the back-end from the frontend with REST-interfaces and implementing the front-end of the software with help of frameworks, such as AngularJS. JSF has received a large amount of critique due to trying to abstract away from HTML, CSS and HTTP by binding them all under a single node.

There are a number of reasons why this style of server-side generated webpages is constantly disappearing more and more. Some respectable reasons include: the performance and developing style is quite slow compared to a client-side generated REST-calling way of developing. For example, if some client-side change was needed, the whole project would need to be compiled and built again, compared to the use of something such as AngularJS, when the client-side change can be seen by just refreshing the page in the browser. This is a major burden in a developing environment, as it requires a longer set of time for the build to be completed. The debugging can be problematic as well, since there are no simple REST-calls in JSON-format to go through and the flow from front-end to back-end is not always logical as it is generated from the server-side Java code. [1]

However, putting the potential problems away and considering the developed application, the choice of using JSF was not illogical. As this application did not need to constantly refresh the data displayed on each page and the flow of the application itself was relatively simple, JSF worked well in this case and provided a solid base for understanding how the back-end and front-end communicate in a web application.

3.3 Managed JavaBeans

A managed JavaBean is a Java class registered to the JSF framework. In all simplicity, the JavaBeans are used to bring basic Java class functionalities from the back-end for a web application’s front-end to use. In the example above, the submitted text input was sent to a backing bean name “testBean.name”. This bean is shown in code-format below in listing 4.

The bean is registered using an annotation “@ManagedBean”. This lets the application know that when referring to “testBean”, it is calling this class and its methods. In this example a getter and a setter is needed for the bean, as the application is first sending the input text to the bean’s property “name” with “setName”. After submitting the page, the bean’s getter, “getName”, is used to get the value of the submitted field. [2]

```
package com.test.project;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import java.io.Serializable;

@ManagedBean
@SessionScoped
public class TestBean implements Serializable {

	private static final long serialVersionUID = 1L;

	private String name;

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

Listing 4. TestBean-implementation of a Managed JavaBean.

3.4 ActiveJDBC

ActiveJDBC is a framework that aims to make it easier for developers to connect to MySQL-databases. It is an implementation of Active Record design pattern and it does not require any specific configuration file. Nonetheless, while it does not require a configuration file, it is general good practices of coding to include a configuration file that involves parameters, such as server ip-addresses, usernames and passwords that are used to connect to the server. At its current state, ActiveJDBC supports SQLServer-, MySQL-, Oracle-, PostgreSQL-, H2- and SQLite3-databases. [12]

The simplified usage of ActiveJDBC can be seen in listing 5. The example code only shows a brief look at ActiveJDBC, but even from this example, the obvious advantages can be seen code-wise. The code is easier to read, the code is more understandable and the casting to the model-class “TestTable” is done without any separate code. This could be done in the upper part of the example in the while-loop, but it would be highly inefficient in a large query or even when executing the simplest queries often in a short period of time.

There is only one search executed in this example, but ActiveJDBC supports a whole bunch of different ready-to-execute SQL-commands in a simplified method-manner. To mention a few, there are parametrized methods for querying the database for entities and simple save-, update- and delete-methods. [10]

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdatabase", "testuser", "testpassword");
Statement statement = connection.createStatement();
ResultSet resultSet = statement.execute("SELECT * FROM `testdatabase.TestTable`");
statement.close();
connection.close();

while (resultSet.next()) {
    //Do something with the result
}


Base.open("com.mysql.jdbc.Driver", "jdbc:mysql://localhost/testdatabase", "testuser", "testpassword");
List<TestTable> testTableEntries = TestTable.findAll();
Base.close();

for (TestTable entry : testTableEntries) {
    //Do something with the result
}
```

Listing 5. A comparison of the traditional SQL-query and ActiveJDBC-way.

3.5 Bootstrap

Bootstrap is a free open-source collection of front-end web developing tools. It was originally built at a company called Twitter by a small group of developers. They were trying to develop a tool that would help developers with consistency across different internal components in projects. At first it was named “Twitter Blueprint”, from where it was changed into “Twitter Bootstrap” and after time into its current form “Bootstrap”. [3]

The Bootstrap framework is aimed at making web development easier. It provides a good-looking modern stylesheet that has a very rich set of ready-styled components. The up-to-date version of Bootstrap is more focused on building mobile-first projects, which means that the framework is very dynamic. It can easily be used with different screen sizes without a large amount of work put into developing for every screen size.

An example of Bootstrap's styling can be seen below, in figure 1.

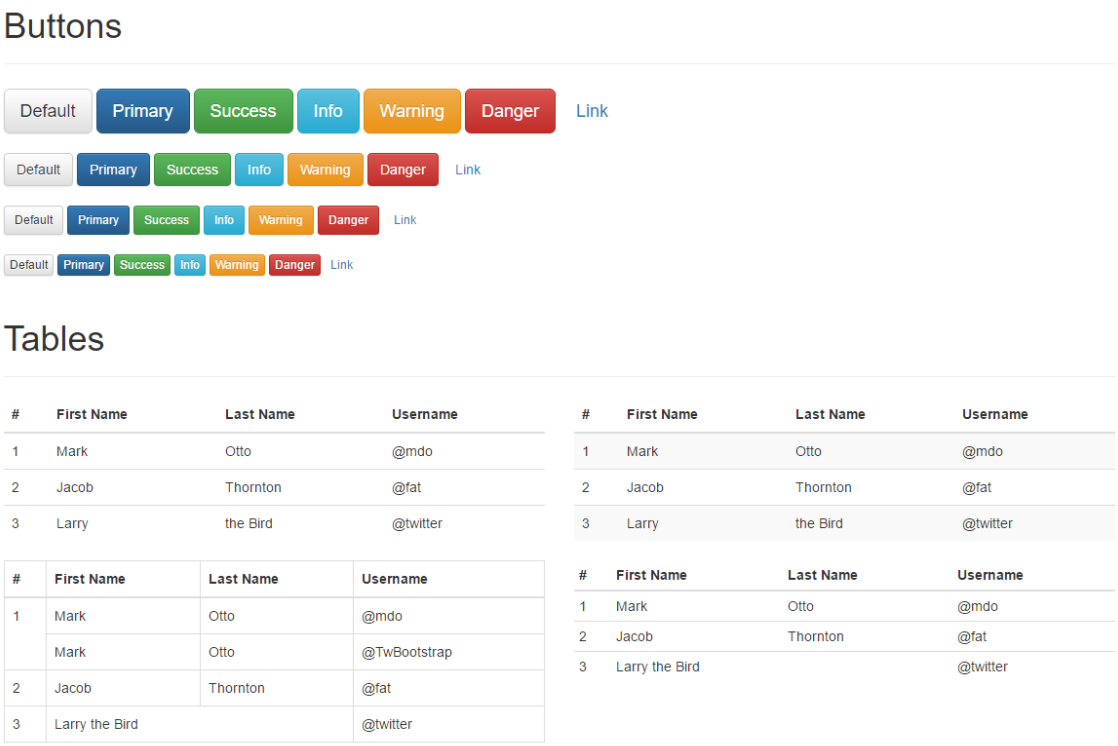

Figure 1. Examples of Bootstrap's styling.

4 Service Implementation

4.1 The Process Before Implementing the Service

The graduation process before this project is illustrated in figure 2. It starts off with the graduating student checking himself/herself in to the registration system of Metropolia, valmistuminen.metropolia.fi. From there on the student's application to graduate goes on to the Student Affairs Office, which takes care of checking that the graduating student can indeed graduate, i.e. all the studies are completed and everything is returned appropriately. After the student has been checked, the application will move on in the system flagged under "graduating". From here on the student's information is passed on to the printing services, who then print out the diploma and the required appendices affiliated with the student's completed studies etc. Four copies of the graduation diploma are printed. One in Finnish, one in English and one copy of each, which will be stored in the archives. The copies are then mailed to the president's office and each paper is hand-signed by the president. These four diplomas are all in paper form and after the signing is complete, they will then be mailed to the Student Affairs Office, who will then forward them on to other personnel who need to sign the documents and eventually to the graduating student on the day of graduation.

This process has a few essential problems. First and most importantly, the process requires the president to hand-sign all the documents, which causes a huge strain on the writer's wrist. The second big issue comes in the form of nature preservation. The sheer amount of paper printing during the process is huge. As the world evolves, the use of paper copies in the modern age is becoming more and more of a problem. Businesses and people in general want to use less and less of the traditional paper prints and prefer using digital copies of such documents. The process afterwards complies with Metropolia's Green Office way of acting. The third problem worth mentioning is the amount of time the whole process takes. Because of the manual printing, mailing the papers from premises a to premises b and onwards and hand-signing the documents, the process takes weeks to complete. Just by not having to mail the diplomas back and worth and not having to hand-sign each document individually, the time to complete the process will be cut by a large margin.

Upon the aforementioned problems, there is also the matter of work time reduced spent to this process. By not having to individually generate the diploma by ticking multiple

checkboxes related to the student and letting the system generate the diploma based on the student, the Student Affairs Office will only spend a fraction of the time creating the actual diplomas, not to even mention the time saved from the president's work day and load by replacing the old hand signing with this digital

system. The process before implementing this new system was cumbersome, to say the least. A flowchart of the process can be seen below, in figure 2.

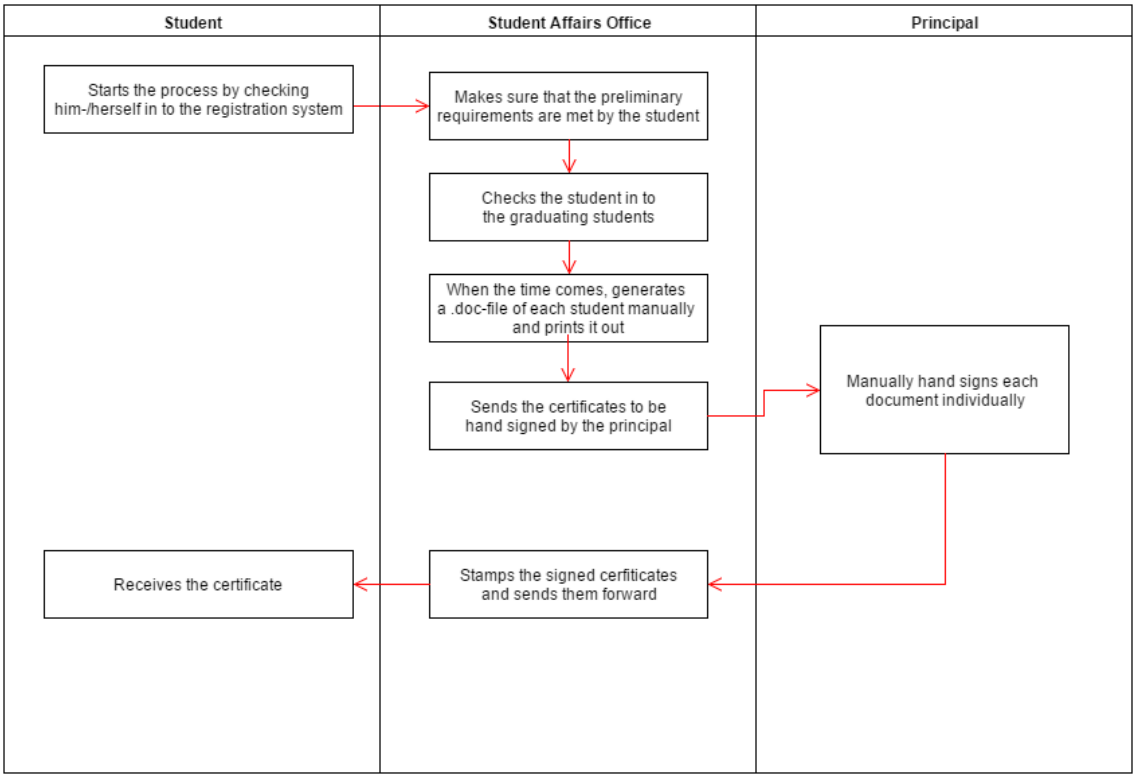

Figure 2. A flowchart of the process before implementing the service.

4.2 The Process After Implementing the Service

After the system has taken over the old way of completing the process, the workflow of the graduation system is significantly simplified for each participant of the process. The graduating student checks in to the registration system in the same way as before. After the Student Affairs Office has confirmed that the student meets the preliminary requirements to graduate, the rest of the graduating process will be handled by the implemented system. The student gets the “graduating” flag as before, but unlike before, no papers are printed to be mailed to a different location at this point. At this point, the president can log in to the system and sign the checked graduates digitally, which will then be stored in to the database. After completing the signature process, the graduating student moves on in the system’s database flagged as “signed”. At this point the system is ready to generate a PDF/A valid document of the graduating students’ diploma to be printed and stored digitally in Metropolia’s servers. The Student Affairs Office can see a preview of the files or a specific file or just go ahead and select all of them with checkboxes and download an archive containing all the diplomas for the graduating students. At least a single copy per student will be printed out, however, and handed over to the student.

The whole process can be done multiple times per graduating date, if for example a student checks into the graduating system late from the schedule. The signing can also be done out of office, if there is a somewhat of an emergency with the needed signature, such as the previous example of a student signing in late into the graduating system. This makes the whole system more robust and flexible on everyone’s end.

A flowchart of the process can be seen in figure 3.

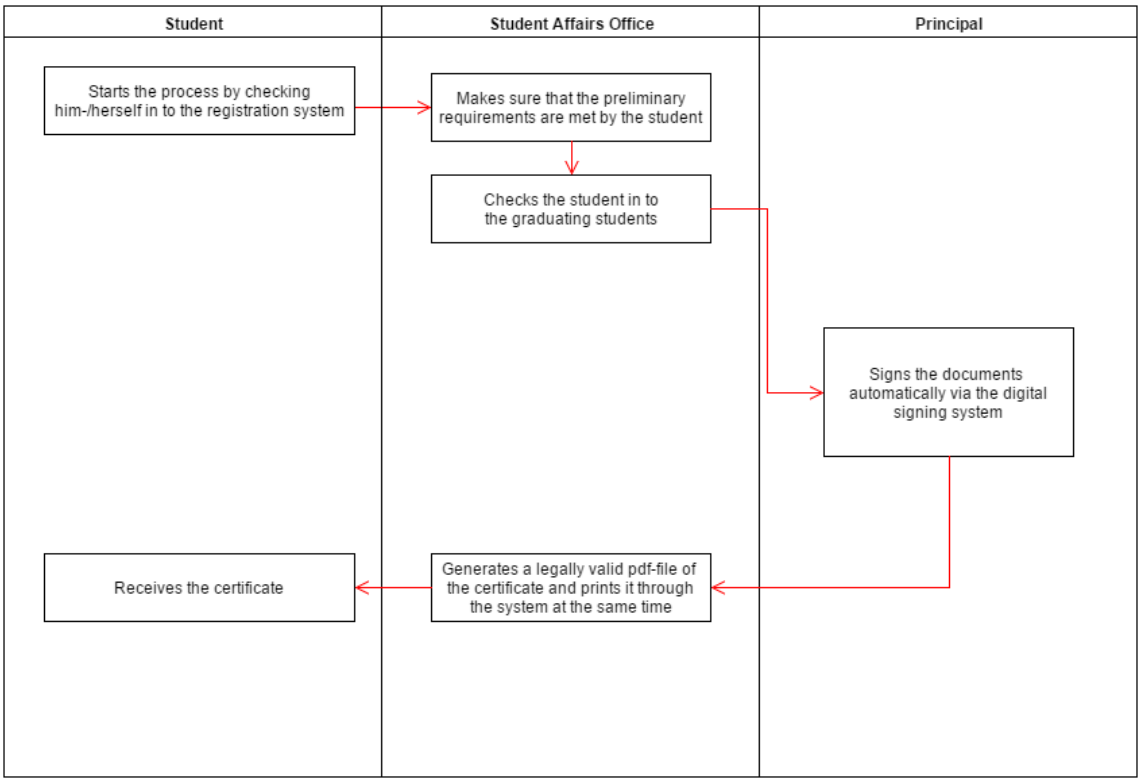

Figure 3. A flowchart of the process after implementing the service.

4.3 Usage of Technologies

Maven was used in the project as the management tool for third party libraries and for managing the build of the project. As seen in an example in section 5.1, downloading the dependencies and selecting which version of them to use is extremely simple with Maven. The build itself is done by running a command “mvn install”, which will generate a .war-file with the name that is specified in the same pom.xml-file. [7] The generated package is then deployed to a container instance, which was Tomcat in this project, and from there on the further steps are managed automatically by Tomcat, as seen in section 5.2.

The developing itself was done with JSF 2.0, which is a framework that takes care of the back-end and the front-end. The front-end is generated in the back-end with the framework from the given tags with prefix “h:”. An example of using the framework can be seen in section 3.1. The process only involved a single user and a generated field based on their roles in the system, which was one of the reasons why this technology was used. The user management was

simple and very efficient with the help of the Jasig CAS Client library. The project itself was basically a view based on the user's role and a few extra views from there with some selectable objects along the way. This led to the usage of JSF 2.0, as the process was fairly simple to build with the help of the framework. It also helped me significantly in understanding the back-end's and front-end's relationship in a web application, after I had to make sense of the whole process of generating the front-end's views with the framework and calling the back-end's services to get data from it.

Web services were used to make the project more dynamic and future proof for future implementations. In this way the data is always sent from a specific source where any application with the given rights can get the information they need from this system. It also simplifies the whole process heavily, when the requesting end does not have to parse through the response and assign the correct value for their correct properties, as the web service returns an object in a model type. In this way the correct values are already set to their correct properties.

The actual view of the project was made to be better looking with the help of Bootstrap. It is widely used in today's web development for a reason. The default look of components is simple and very usable and there is a wide variety of components to use. The customization of the components is done by creating a specific css, Cascading Style Sheets, file that includes the wanted modifications or by modifying them directly to Bootstrap's style.css file. This only requires a basic understanding of css files.

Apache Subversion was used to handle the project version management. As the project was a simple one developer project, I couldn't fully take advantage of the tool's features. I could however create different branches for myself when I was developing a specific feature, which I wanted temporarily to keep away from the main branch. Even though Subversion and its alternatives are best used in projects that include more than one developer, it was helpful to use such a tool to get a feel for version control and how it works.

5 Development Tools Used

5.1 Apache Maven

Apache Maven is an open-source package managing tool designed and aimed at building and helping library management in any Java based projects. Maven projects come with a file named “pom.xml”. The name “pom” stands for “Project Object Model”. It is the equivalent to Ant’s build.xml but significantly easier to use and more versatile [6]. In a simplified manner, for example, only four lines of xml code that represent dependencies are needed to include a library to a project. Maven then automatically downloads the specified library from its own repositories, so the programmer does not need to worry about downloading the libraries and adding them to the project. This in effect helps heavily when one can simply change a version number of a library and Maven will download the library, instead of manually downloading the package from the internet and applying it to a project. Through pom.xml the programmer can also specify different build settings such as testing the application and ultimately creating the .war-file for the project that is used to deploy the project to the web server. An example of a very basic pom can be seen below, in listing 6.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.test.project</groupId>
      <artifactId>JavaServerFaces</artifactId>
      <packaging>war</packaging>
      <version>1.0-SNAPSHOT</version>
      <name>JavaServerFaces Maven Webapp</name>
      <url>http://maven.apache.org</url>

      <dependencies>
            <dependency>
                  <groupId>com.sun.faces</groupId>
                  <artifactId>jsf-api</artifactId>
                  <version>2.1.7</version>
            </dependency>
            <dependency>
                  <groupId>com.sun.faces</groupId>
                  <artifactId>jsf-impl</artifactId>
                  <version>2.1.7</version>
            </dependency>
            <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>jstl</artifactId>
                  <version>1.2</version>
            </dependency>
            <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>servlet-api</artifactId>
                  <version>2.5</version>
            </dependency>
            <dependency>
                  <groupId>javax.servlet.jsp</groupId>
                  <artifactId>jsp-api</artifactId>
                  <version>2.1</version>
            </dependency>
      </dependencies>

      <build>
            <finalName>JavaServerFaces</finalName>
            <plugins>
                  <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>2.3.1</version>
                        <configuration>
                              <source>1.6</source>
                              <target>1.6</target>
                        </configuration>
                  </plugin>
            </plugins>
      </build>
</project>
```

Listing 6. An example of Maven's pom.xml-file.

When running a very simplified build and install -command “mvn clean install” will create a package ready to be deployed to the container. Maven then should in all its simplicity let the user know the information found on figure 4, if the build was completed successfully.

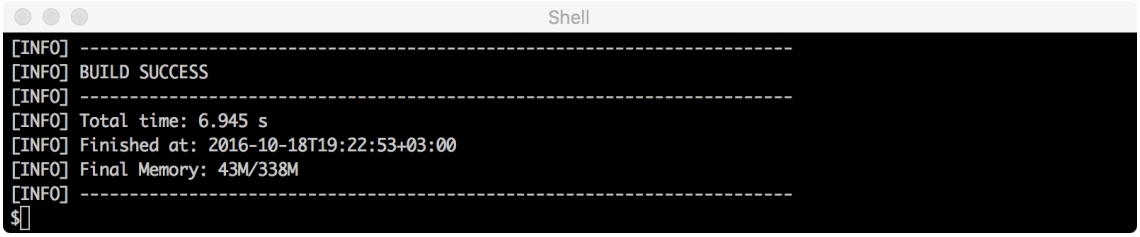

Figure 4. Example output of a successful build with Maven.

5.2 Apache Tomcat

Apache Tomcat is an open-source web server and servlet container. It provides a pure Java HTTP web server environment for Java applications to run. It is lightweight in terms of using resources and in many cases meets the needs for any application. It is also extremely easy and quick to set up and the deploying process is really straightforward with the provided user interface.

The startup and shutdown of tomcat is seen in figure 5 and 8. The examples are successful runs of the commands. In case there is something wrong with the process, the command line will let the user know the error message. Figure 6 shows a running tomcat instance and its default page. Figure 7 shows the deploy view of tomcat’s default installation. [7]

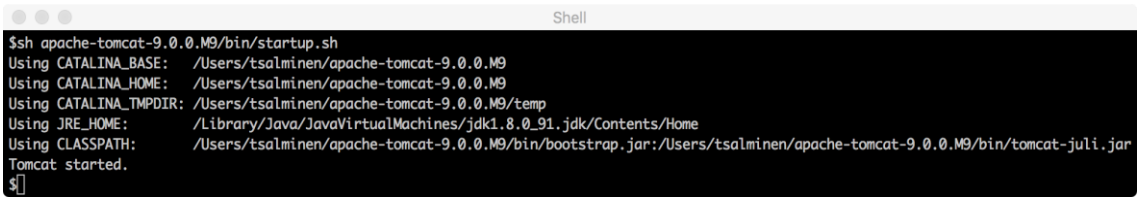

Figure 5. Example output of Tomcat’s startup.

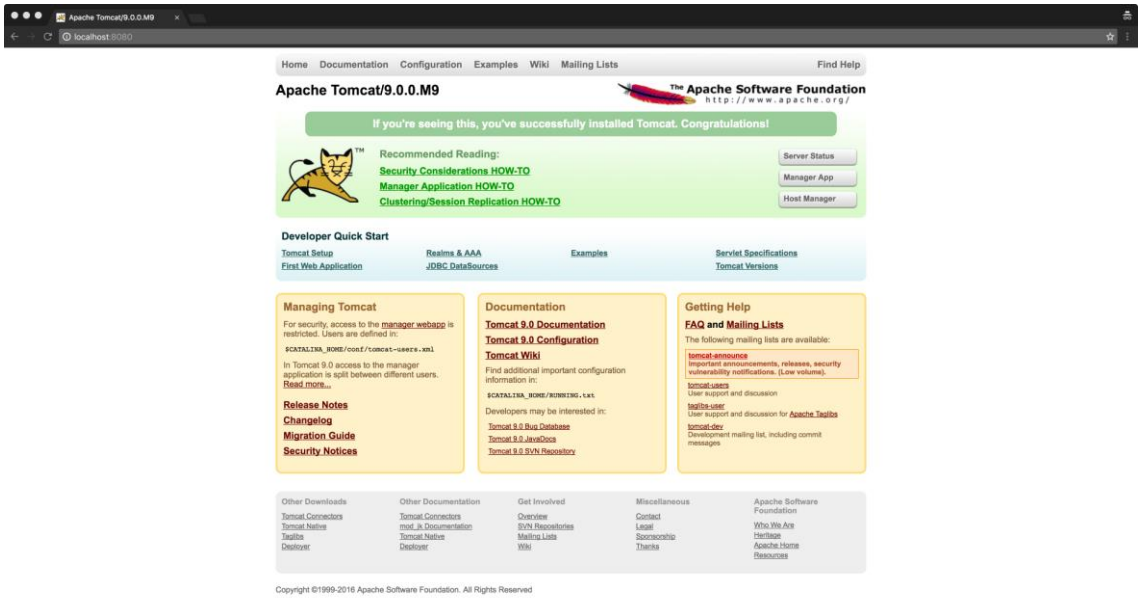

Figure 6. The main page of a running Tomcat container.

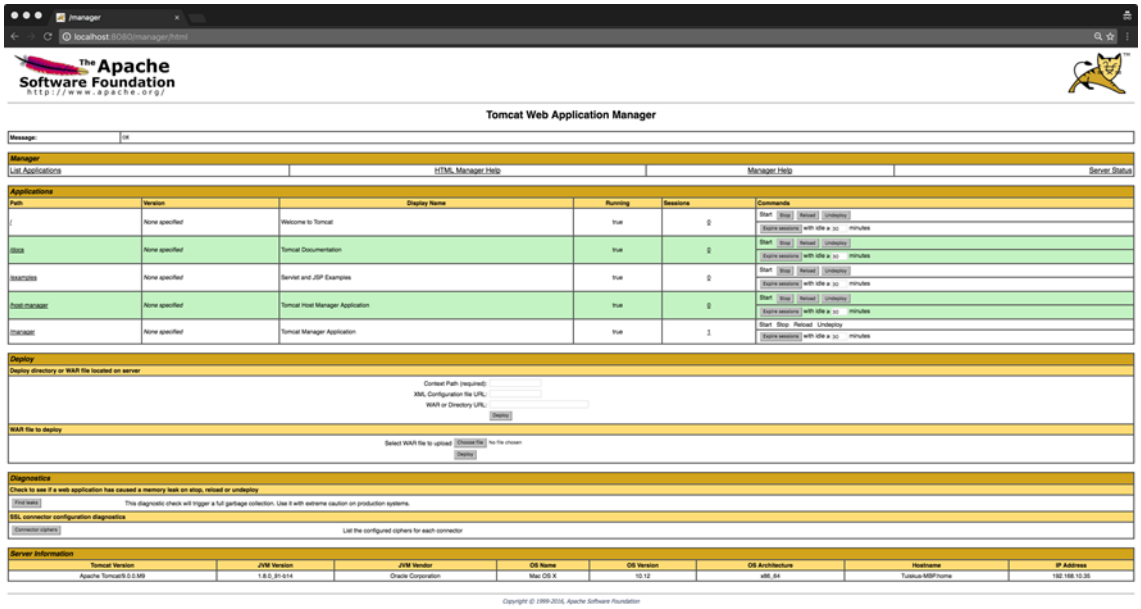

Figure 7. The management page of a running Tomcat container.

Figure 8. Example output of Tomcat's shutdown.

5.3 Apache Subversion

Apache Subversion, often abbreviated as SVN, is a version control software. It aims to make developing easier for developers by providing a repository for the project's code. Different developers can access the code from the repository and make a local copy to their hard disks. In this way the developers can work simultaneously on the same project. [8]

Subversion's available commands can be seen below in figure 9. A basic checkout of a project would happen with a command shown in figure 10 in the root of the location on the hard disk intended to be used for the project. If there is need for validation, the user can run a command "svn info" on the same path. The given information would be similar to figure 11.

Subversion was created in the year 2004, which makes it quite old in terms of software. Subversion is losing popularity year after year due to the rise of Git, an alternative tool to Subversion. A novice user in version control might find Subversion easier to adopt, since the commits are always made to the remote repository. Git has a local repository version of the system compared to Subversion's way of only having a single local version of the project. The user can first create a commit to their local repository and from there on can make a push-request to the remote repository. More of Git and how Subversion compares to it in section 5.6.1. [8]

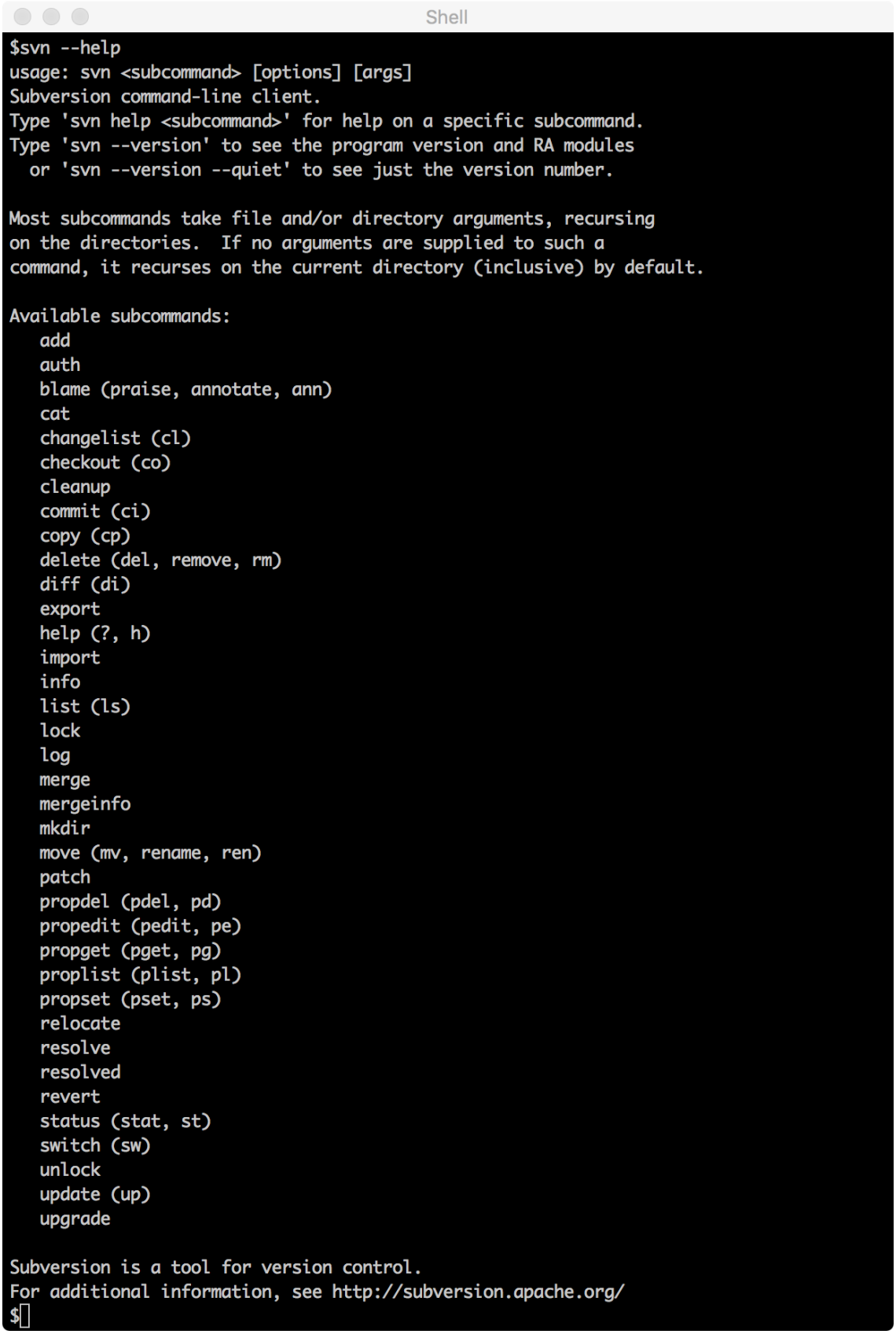

Figure 9. A list of Subversion’s available commands.

```
svn checkout <svnrepositoryurl> --username=username --password
```

Figure 10. A command to checkout a SVN-project.

```
Path: .
URL: <svnrepositoryurl>/trunk
Repository Root: <svnrepositoryurl>
Repository UUID: 12345678-abcd-4321-a123-1234c123c123
Revision: 1
Node Kind: directory
Schedule: normal
Last Changed Author: User
Last Changed Rev: 0
Last Changed Date: 2010-01-01 00:00:00 +0200 (Fri, 01 Jan 2010)
```

Figure 11. An example output of running a command “svn info”.

5.4 MySQL

MySQL is an open-source relational database management system developed by the Oracle Corporation. It was initially released in 1995 and it is still widely used across the world, over 20 years later.

A database is a separate application used to store a collection of data. In MySQL’s case it holds a collection of tables, which each hold a set of columns. An entry or a row is a group of data in the form of the given table. An entry can hold data in terms of each column or it can be empty. This is all done in the table’s terms from the database. A column can be made to be required, i.e. not empty or it can be made to accept null-value as the “data”.

5.5 Web services

The designed system implements access to databases and one form of the signing methods as web services. Web services are a type of communication tool between the client (this application) and the server side (application that returns the needed information). The client side accesses the web service through the implemented methods and gives the parameters needed by the method accordingly. In all of the calls, an api-key is also

provided, which acts as an authentication key for the service, i.e. makes sure the accessing client has the rights to get the information from the web service. If the api-key provided to the web service is invalid, i.e. not found in the web service's api key list, the web service will return an empty response.

This application follows Simple Access Protocol, SOAP, standard for defining the message architecture and format. There is also a standard called REST, which does not require any xml standard for definitions.

SOAP is used in the application because the web services used are tightly coupled with specific information from the applications database. The whole idea of using web services in this application is to prevent the need for the application to contact databases directly. This makes the application more open to changes in the future and makes the application more secure.

5.5.1 Web Services Description Language

Web Services Description Language, WSDL, is a format that describes the services that are affiliated with the web service. The programming language used for this is XML. As everything regarding the web service is defined here, the WSDL-document must include at least the following definitions:

- Definitions
- Type
- Message
- Port Type
- Binding
- Service
- Port. [11]

5.5.2 Simple Object Access Protocol

Simple Object Access Protocol, SOAP, is a protocol for accessing web services. It is based on XML. SOAP was created to accomplish a way of communication between applications through HTTP, which is supported by all browsers. A SOAP message is con-

structed using the following elements: Envelope, Header, Body, Fault. Envelope identifies the xml document as a SOAP message. The header element contains header information and can be set to “mustUnderstand”. This requires the recipient to understand and process the given header attribute in order to function correctly. If a header is present, it must be the first child element of the envelope element. The body element contains all call and response information. The fault element contains the possible generated error and status information. After a request in the right format is sent, the generated response must be parsed with the appropriate ways in the request sender's end. [11]

5.5.3 Representational State Transfer

Representational State Transfer, REST, is another protocol for accessing web services. Usually the request and response calls are made in JSON-format, but REST also works with XML and HTML for example. REST is gaining more and more popularity in the modern web application architecture, where it is generally considered good practice to separate the back-end from the front-end. REST-calls provide an easy way for the front-end to gain information from the back-end of the application through JSON-format. [14]

5.6 Git - an Alternative to Subversion

Like Subversion, Git is version control software. It is mainly aimed at distributed, non-linear workflows, so in order to take full advantage of all its features, it is best used in a team project. In this way the team members can work on their separate versions of the project and keep their own local repositories up to date from the remote repository. When the feature is done, the user can review the changes done to the project and merge them to the remote repository.

As software development evolves more and more towards team work and agile work methods, the success of Git can be easily understood. It is however somewhat more complicated than subversion to learn.

6 Future Implementations for the Application

The project was eventually meant to be included in every document signing that happens in Metropolia. To support this and make the software more dynamic and easier to develop in the future, REST interfaces were created to work in parallel with the SOAP interfaces. The REST interfaces make handling documents simpler and is more robust than SOAP. However, at the time of the development, SOAP was requested as the interface, so it was the main priority in this project.

After a test period of the initial release of the software, user surveys will be carried out and the system will be developed with the results in mind.

7 Conclusion

Replacing outdated ways of handling documents is an important part in keeping up with the technology-driven world in the modern age. This is particularly important for universities, such as Metropolia University of Applied Sciences, as students are most likely to expect modern methods used to their advantage.

The goal of developing this system was to make the graduating process easier for both, the student and the faculty associated with the graduation process. In this thesis, the project workflow, the project's functionalities and the technologies associated with the project were discussed in a brief, but explanatory fashion. The required standards were explained individually one by one. The workflow was split into two states, before implementing the service and after implementing the service. The steps associated with these states were explained with text and workflow diagrams. The technologies used in the project were introduced with usage examples, which were explained.

The thesis explains why the chosen technologies and tools were used in this project. The thesis can be used to understand the developed system in compliance with the source code of the system. Further studying of the subjects is required to get a better understanding of the technologies and how to use them.

In the future the service will take advantage of the REST interfaces that already exist in the project. These interfaces make it easier to adapt the service to work as a universal tool in digital signing of documents, instead of just taking advantage of the service in diplomas.

References

1. Oracle Corporation. JavaServer Faces Technology. [Online]. Oracle Corporation, CA, US.
   URL: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Accessed 27 October 2016.

2. Oracle Corporation. JavaBeans Spec. [Online]. Oracle Corporation, CA, US.
   URL: http://www.oracle.com/technetwork/articles/javaee/spec-136004.html. Accessed 27 October 2016.

3. What is Bootstrap and How Do I Use It? [Online]. Tania Rascia; 9 November 2015.
   URL: https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/. Accessed 27 October 2016.

4. Kansalaisen tunnistus- ja maksamispalvelu Vetuma. [Online]. Valtori, Jyväskylä, Finland; 24 February 2016.
   URL: http://www.valtori.fi/palvelut/vetuma. Accessed 27 October 2016.

5. PDF Association. PDF/A FAQ. [Online]. PDF Association, Berlin, Germany.
   URL: https://www.pdfa.org/pdfa-faq/. Accessed 27 October 2016.

6. The Apache Software Foundation. What is Maven? [Online]. The Apache Software Foundation, MD, US; 16 October 2016.
   URL: http://maven.apache.org/what-is-maven.html. Accessed 27 October 2016.

7. The Apache Software Foundation. Tomcat Web Application Deployment. [Online]. The Apache Software Foundation, MD, US; 14 September 2016.
   URL: https://tomcat.apache.org/tomcat-7.0-doc/deployer-howto.html. Accessed 27 October 2016.

8. The Apache Software Foundation. Apache Subversion FAQ. [Online]. The Apache Software Foundation, MD, US.
   URL: https://subversion.apache.org/faq.html. Accessed 27 October 2016.

9. Metropolia University of Applied Sciences. Annual Report 2013. [Online]. Metropolia University of Applied Sciences, Helsinki, Finland; 2013.
   URL: http://vuosikertomus.metropolia.fi/2013/en.html. Accessed 27 October 2016.

10. Java IoT. ActiveJDBC: New Java ORM. [Online]. Max Katz, 18 July 2011.
    URL: http://java.sys-con.com/node/1912289. Accessed 27 October 2016.

11. Tutorialspoint. WSDL Example. [Online]. Tutorialspoint, 2016.
    URL: https://www.tutorialspoint.com/wsdl/wsdl_example.htm. Accessed 27 October 2016.

12. JavaLite. ActiveJDBC. [Online]. JavaLite, 2016.
    URL: http://javalite.io/activejdbc. Accessed 27 October 2016.

13. InfoWorld. Best practices in using RESTful services. [Online]. InfoWorld, 2016.
    URL: http://www.infoworld.com/article/2946856/application-architecture/best-practices-in-using-restful-services.html. Accessed 27 October 2016.