Examensarbete, Högskolan på Åland, Utbildningsprogrammet för Informationsteknik

VIDAREUTVECKLING AV ADMINISTRATIONSAPPLIKATION

Dag Karlsson

15:2018

Datum för godkännande: 30.05.2018
Handledare: Joakim Isaksson

EXAMENSARBETE
Högskolan på Åland

Utbildningsprogram:	Informationsteknik
Författare:	Dag Karlsson
Arbetets namn:	Vidareutveckling av administrationsapplikation
Handledare:	Joakim Isaksson
Uppdragsgivare:	Extern uppdragsgivare

Abstrakt

Syftet med examensarbetet är att vidareutveckla en administrationsapplikation för att förenkla hämtning för slutanvändares mobila enheter samt förenkla hämtning och uppdatering av slutanvändare i ett datalager genom personnummer.

Utvecklingen av denna applikation har genomförts huvudsakligen med hjälp av programmeringsspråket Java och ramverket Spring.

Arbetet medförde utökad funktionalitet i applikationen och slutresultatet levde upp till beställarens alla specifikationer och krav.

Nyckelord (sökord)

Java, Spring, MVC, MyBatis

Högskolans serienummer:	ISSN:	Språk:	Sidantal:
15:2018	1458-1531	Svenska	29 sidor

Inlämningsdatum:	Presentationsdatum:	Datum för godkännande:
30.05.2018	16.05.2018	30.05.2018

DEGREE THESIS
Åland University of Applied Sciences

Study program:	Information Technology
Author:	Dag Karlsson
Title:	Further Development of Back Office Application
Academic Supervisor:	Joakim Isaksson
Technical Supervisor:	External employer

Abstract

The purpose of this thesis is to further develop a backoffice application in order to simplify data collection of end user's mobile devices and to simplify the collection and updating of end user data in a data warehouse by social security numbers.

The development of this application has been completed primarily by using the programming language Java and the application framework Spring.

This work has brought extended functionality to the application and the end result has lived up to the client's every specification and demand.

Keywords

Java, Spring, MVC, MyBatis

Serial number:	ISSN:	Language:	Number of pages:
15:2018	1458-1531	Swedish	29 pages

Handed in:	Date of presentation:	Approved on:
30.05.2018	16.05.2018	30.05.2018

INNEHÅLLSFÖRTECKNING

1. INTRODUKTION **6**
1.1 Syfte 6
1.2 Metod 6
1.2 Avgränsningar 7

2. TEKNOLOGIER OCH RAMVERK **8**
2.1 Spring MVC 8
2.2 Spring Security 8
2.3 Spring Profiles 9
2.4 MyBatis 9
2.5 Apache Tiles 9
2.6 Java Server Pages 10
2.7 Jira 10
2.8 Distributionsmiljöer 10
2.8.1 Development environment 10
2.8.2 Test environment 11
2.8.3 Stage environment 11
2.8.3 Production environment 11
2.9 JBoss 11

3. ARKITEKTURER OCH DESIGNMÖNSTER **12**
3.1 Domändriven design 12
3.1.1 Gränssnitt 13
3.1.2 Applikation 13
3.1.3 Domän 13
3.1.4 Infrastruktur 13
3.2 Inversion of Control 14

4. UTVECKLINGEN AV APPLIKATIONEN **15**
4.1 Förberedelser inför arbetet 15
4.2 Visning av slutanvändares mobila enheter 16
4.2.1 Persistenshantering genom MyBatis 16
4.2.2 Hantering av data genom Spring MVC 17
4.2.3 Hantering av vyer genom Spring MVC och Apache Tiles 18
4.2.4 Förändringar under resans gång 19
4.3 Uppdatering av slutanvändare i ett datalager 20
4.3.1 Förändringar av utvecklargränssnitten 20
4.3.2 Konfiguration för utvecklargränssnitten 21
4.3.3 Hantering av data i Spring MVC 22

4.3.4 Hantering av filer 24
4.3.5 Autentisering av användare 25

5. SLUTSATSER 27
5.1 Resultat 27
5.2 Reflektioner 27

KÄLLOR 29

1. INTRODUKTION

1.1 Syfte

Examensarbetets syfte var att vidareutveckla en del av en administrationsapplikation för att möjliggöra flera sätt för användare av applikationen att hämta information om slutanvändare. Administrationsapplikationen används både av beställaren och beställarens kunder.

Användare skall kunna hämta och visa tidigare svåråtkomlig information om slutanvändares mobila enheter, inloggningar och inloggningarnas status.

Användare skall kunna hämta och uppdatera information i ett datalager. Tidigare har detta gjorts på daglig basis genom automatiserade batchjobb. Funktionalitet för detta skulle lyftas in i applikationen genom en manuell och lättanvänd tjänst. En behörig användare skall kunna mata in enskilda slutanvändares personnummer eller filer som innehåller många personnummer som skall uppdateras och användaren skall därefter få information om operationernas resultat. Filerna skall kunna vara i Excelformat eller vanliga textdokument.

1.2 Metod

Det första som behövde göras var att skaffa sig en helhetlig uppfattning om vad som egentligen hörde till arbetet och vad beställaren ville få ut av den nya funktionaliteten och sammanställa detta i en kravspecifikation som behövde godkännas av beställaren.

En del efterforskning av den nuvarande applikationens implementation behövde även göras för att försöka hålla det nya arbetet så konsekvent och enhetligt som möjligt samt även för att möjliggöra återanvändning av existerande funktionalitet. En stor del av arbetet gick ut på att ta reda på hur strukturen såg ut i den redan existerande funktionaliteten för att uppdatera slutanvändare i datalagret och hur man skulle kunna återanvända sig av det.

Implementationen av den nya funktionaliteten har gjorts i Java med hjälp av MyBatis för att hämta data ur databaserna, gränssnittslogiken har gjorts med hjälp av Spring MVC och säkerheten har implementerats med hjälp av Spring Security. För att kunna separera vilka kunder som skall använda den nya funktionaliteten har Spring Profiles använts och vyerna i det webbaserade användargränssnittet har använt teknologier såsom Java Server Pages(JSP), JSP Standard Tag Library(JSTL) samt JSP Taglib Directive. I enlighet med beställarens standard och kodningsregler har detta arbetes struktur följt riktlinjerna i mönstret Domain Driven Design (DDD).

1.2 Avgränsningar

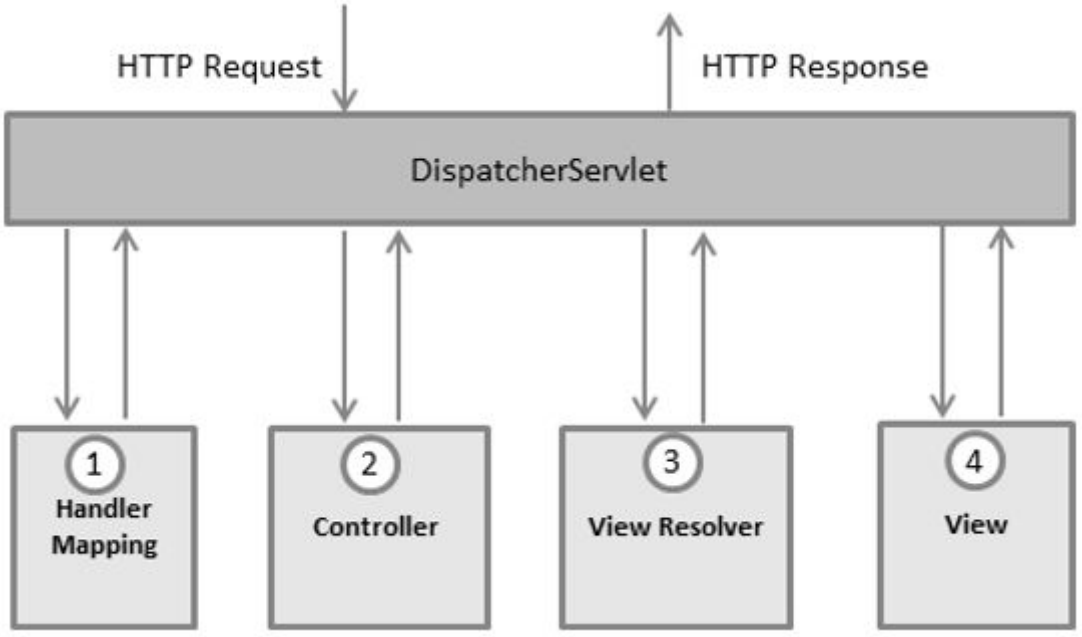
Det framkom i ett ganska tidigt skede att jag behövde göra en del förändringar i olika gränssnitt som jag skulle utnyttja. Även om det fanns behov av att grundligt göra om och förbättra dessa gränssnitt avgjorde jag i ett tidigt skede att tiden inte skulle räcka till.

Jag fick nöja mig med att göra den förändring som absolut krävdes och lämna resten. Även inuti applikationen jag skulle vidareutveckla så upptäckte jag att vissa ramverk som använts tidigare var föråldrade och skulle behövts bytas ut och förbättras, men jag beslöt att bara börja nyttja dessa nyare ramverk för den nya utvecklingen som jag skulle göra och låta den äldre delen av applikationen fortsätta använda dessa föråldrade ramverk.

2. TEKNOLOGIER OCH RAMVERK

2.1 Spring MVC

Spring MVC är ett ramverk som låter utvecklaren sammankoppla allt som krävs för att skapa en webbsida. Ramverket hanterar anrop till en specifik webbadress och kontrollerar vilken data som skall ges vidare till vyn och slutligen också vilken vy som skall användas. Detta är ett ramverk som använder sig av ren Java och vanligtvis returneras en vy som är gjord med teknologin Java Server Pages till användaren. Figur 1 illustrerar denna funktionsprincip. (Walls, 2014)



Figur 1. Funktionsprincip för Spring MVC (Spring MVC Framework, u.d)

2.2 Spring Security

Spring Security är ett säkerhetsramverk som säkrar trafiken till webbsidan, sköter om autentisering av användare och skyddar webbsidan från attacker av olika slag (Walls, 2014). Jag har inte själv behövt göra så mycket arbete med ramverket då det redan var konfigurerat för applikationen och därför så har det mesta ordnats automatiskt. Funktionaliteten för att

hämta och uppdatera användares information krävde en högre nivå av åtkomst så därför behövde jag använda Spring Security för att verifiera att användaren hade den åtkomst som krävdes.

2.3 Spring Profiles

Spring Profiles är ett ramverk som kan styra vilken funktionalitet en viss kontext skall ha tillgång till i en applikation (Walls, 2014). Detta ramverk blev därför ett naturligt val att använda när funktionaliteten för att hämta slutanvändares mobila enheter och inloggningar endast kunde användas av två av företagets kunder. Då det är väldigt enkelt att lägga till nya kunder blev detta ett bra val då denna funktionalitet kanske kommer att finnas för flera kunder i framtiden.

2.4 MyBatis

MyBatis är ett ramverk som sparar, hämtar och hanterar data ur databaser (MyBatis documentation, 2018). Detta ramverk har jag tidigare stött på när jag arbetat för detta företag och blev därför ett naturligt val för denna applikation. Med MyBatis sköts konfigurationerna med sk. XML-filer inuti vilka man specificerar namngivna anrop. Inuti dessa anrop specificerar utvecklaren SQL-satser med dynamiska variabler som används för att ange sökparametrar när utvecklaren vill hämta data, samt Java objekt som resulterande data skall sättas in i. Ett exempel på en sådan konfiguration visas i figur 5.

2.5 Apache Tiles

Apache Tiles är ett ramverk för att knyta ihop användargränssnittets vyer på ett enhetligt och enkelt sätt (Apache Tiles documentation, 2017). Detta ramverk har jag tidigare använt och användes redan i applikationen sedan tidigare, så det blev ett naturligt val att fortsätta använda mig av det.

2.6 Java Server Pages

Java Server Pages är en Javateknologi som används för att skapa en dynamisk webbsida baserad på tillgänglig data. När webbsidan har skapats så består den i princip av traditionell HTML som kan skickas vidare till användaren för visning i webbläsaren (Walls, 2014) .

JSP Tag Library är en teknologi som utökar den traditionella JSP-teknologin. Denna teknologi använde jag mig av för att skapa en egen *taglib*, en sorts samling med funktioner för att hantera Spring Profiles från vyerna. Med denna taglib kunde jag välja för vilken kund vissa delar av vyerna faktiskt skulle visas.

2.7 Jira

Jira är en mjukvara som beställaren använder för att organisera och synkronisera arbetsuppgifter. Med hjälp av Jira kan man skapa olika arbetsuppgifter och dela upp stora arbeten till mindre stycken som är mera hanterbara. Jag har skapat ett s.k. *case* för varenda delmoment som jag gjort. Om man har haft frågor eller funderingar kring något har det varit enkelt att hänvisa andra till delmomentet jag behövde hjälp med.

2.8 Distributionsmiljöer

Vid utveckling finns oftast ett flöde där koden blir verifierad genom några olika typer av distibutionsmiljöer. De fyra miljöerna beställaren använder är följande (Deployment environment, 2018):

- *Development environment*
- *Testing environment*
- *Staging environment*
- *Production environment*

2.8.1 Development environment

För beställaren avser *development environment* egentligen varje utvecklares lokala maskin. Då en utvecklare utvecklar något så håller utvecklaren det enbart på sin egen lokala maskin tills det finns något som verkar fungera och då kan utvecklaren ladda upp den nya koden till ett gemensamt *code repository*.

2.8.2 Test environment

Denna miljö uppdateras dagligen och kommer alltid att köra den senaste versionen av koden i beställarens *code repository*. Denna miljö används för att testa helheten och här kommer beställarens personal att utföra tester av många olika slag.

2.8.3 Stage environment

Denna miljö skall likna den riktiga produktionsmiljön så mycket som möjligt. Efter att koden har legat i testnings miljön en längre tid och blivit testad och verifierad som stabil hamnar den i beställarens *stage environment*. Det är i denna miljö som beställarens kunder kommer att göra utförliga tester. När detta är gjort och koden har funnits i denna miljö en längre tid flyttas den slutligen till produktionsmiljön.

2.8.3 Production environment

Detta är den riktiga miljön som alla slutanvändare faktiskt använder. Tanken är att när ny kod kommer till denna miljö skall den vara utförligt testad och ha legat så pass länge i de tidigare miljöerna att man vet att den är stabil och inte kommer att leda till överraskningar i form av buggar eller säkerhetsbrister. I en perfekt värld skall koden vara felfri när den når denna miljö.

2.9 JBoss

JBoss är en applikationsserver, vilket innebär att det är en mjukvara som erbjuder en miljö för webbapplikationer att köra på. JBoss är specifikt utvecklad för att köra Java Enterprise Edition applikationer. Java Enterprise Edition är i princip Java applikationer som erbjuder möjligheten att köras som webbapplikationer. JBoss är en väldigt kraftfull applikationsserver som har möjlighet att köra många olika webbapplikationer samtidigt (JBoss documentation, u.d.).

3. ARKITEKTURER OCH DESIGNMÖNSTER

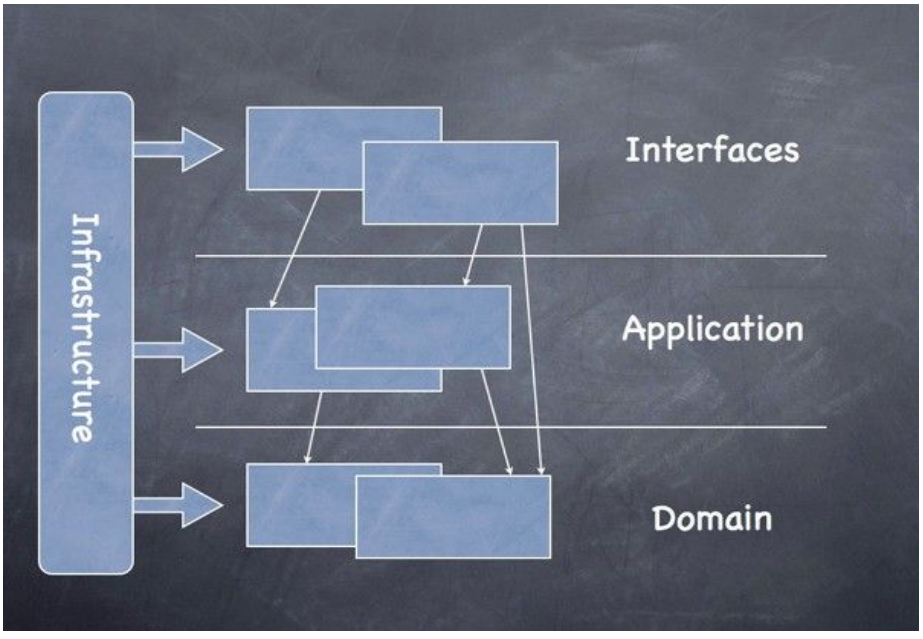
3.1 Domändriven design

Domändriven design är arkitekturen som beställaren följer vid nyutveckling, och således tillämpas denna även för den nya funktionaliteten i applikationen.

Domändriven design handlar först och främst om att programutveckling skall vara nära knuten till det verksamhetsområde som programmet utvecklas för. Denna designmodell ger en ganska specifik metod för hur man skall bygga upp och strukturera applikationer, vilket i sin tur ger en väldigt enhetlig kodbas även för ett väldigt stort företag (Evans & Fowler, 2003).

Enligt beställarens riktlinjer skall deras applikationer vara uppdelade i följande lager som följer domändriven design, se figur 2:

- Gränssnitt
- Applikation
- Domän
- Infrastruktur



Figur 2. Översikt av de olika lagren. (DDD architecture, 2009)

3.1.1 Gränssnitt

I detta lager lägger man allting som ligger närmast vyerna, dvs. detta är det lager som användare ser. Detta lager innehåller typiskt kontrollkomponenter i Spring MVC (*Controllers*) och skall inte innehålla någon tung logik. Den logik som passar in i detta lager är t.ex. validering av indata, transport av validerad indata till applikationslagret och transport av utdata tillbaka till vyerna.

3.1.2 Applikation

Applikationslagret är ett lager som agerar som en typ av knytpunkt. Lagret knyter samman data och kommunikation med domänlagret och infrastrukturlagret. Detta lager erbjuder sedan gränssnitt och funktionalitet som kan användas av gränssnitts-lagret för att hantera data på ett enkelt sätt. En funktion såsom skapaBeställning(...) skulle finnas i detta lager, och sådan funktionalitet skulle typiskt använda sig av både domän- och infrastrukturlagren. En viktig orsak till varför man vill hålla detta lager separat är för att det skall vara enkelt att integrera ett applikationslager med flera olika gränssnitt.

3.1.3 Domän

All affärslogik (*Business Logic*) hör hemma i domänlagret. Det är alltså i detta lager som det mesta händer när man gör t.ex. en beställning. Man delar ytterligare upp detta lager i mindre helheter med sammanhörande funktionalitet. Man kanske t.ex. har en samling eller ett paket med funktionalitet för beställningar.

3.1.4 Infrastruktur

Detta lager skall inte ha fasta kopplingar till något annat lager. Det är i detta lager vi konfigurerar inställningar för olika tjänster och databaser. Implementation av persistenshanterare likt MyBatis är gjorda i detta lager, och principerna för *Dependency Injection* (Se kapitel 3.2) används för att inte skapa fasta beroenden till domän-lagret.

3.2 Inversion of Control

Inversion of Control är ett designmönster för att hantera och minimera beroenden mellan objekt, genom att klasser inte skall känna till implementationen direkt, utan bara förlita sig på ett gränssnitt som är specialiserat på uppgiften som skall göras. På detta vis så har man en väldigt lös koppling till den egentliga implementationen och implementationen kan enkelt bytas ut mot en annan.

Dependency Injection är en tillämpning av Inversion of Control som används för att förenkla utbyte av den konkreta implementationen (Fowler, 2004). I praktiken innebär detta att om man t.ex. har en klass Developer, som använder klassen CoffeeMaker, så känner Developer endast till ett gränssnitt som CoffeeMaker använder och man skickar in den implementarande klassen genom exempelvis konstruktorn (se figur 3). På detta vis är det väldigt enkelt att byta ut implementationen mot en annan klass som förverkligar det gemensamma gränssnittet.

```java
public static void main(String[] args) {
    final CoffeeMaker coffeeMaker = new MoccamasterCoffeeMaker();
    final Developer developer = new Developer(coffeeMaker);
    developer.drinkCoffe();
}

public class Developer {

    private final CoffeeMaker coffeeMaker;

    public Developer(final CoffeeMaker coffeeMaker) {
        this.coffeeMaker = coffeeMaker;
    }

    public void drinkCoffe() {
        coffeeMaker.makeCoffe();
        drink();
    }
}

public interface CoffeeMaker {
    void makeCoffe();
}

public class MoccamasterCoffeeMaker implements CoffeeMaker {

    @Override
    public void makeCoffe() {
        // Makes coffee in a specific wat for this coffee maker
    }
}
```

Figur 3. Exempel på IoC och DI.

4. UTVECKLINGEN AV APPLIKATIONEN

4.1 Förberedelser inför arbetet

När jag skulle börja med detta arbete fanns ingen tydlig kravspecifikation, eller ens en tydlig bild av vad den nya delen av applikationen skulle göra och hur den skulle fungera. Jag var därför tvungen att sätta mig ned i ett videosamtal med beställaren för att försöka få en tydlig bild av vad han hade för förhoppningar och krav på att den nya applikationen skulle göra. När jag väl hade fått en tydligare bild började jag skriva en kravspecifikation för applikationen som skickades fram och tillbaka mellan mig och beställaren, tills vi hade kommit fram till en kravspecifikation som vi båda var nöjda med.

När jag väl hade kravspecifikationen klar började jag analysera den nuvarande kodbasen för att få en bild över hur jag enklast kunde genomföra arbetet jag hade framför mig. Det var många saker som var väldigt osäkra för mig men jag fick vidare handledning av beställaren, och jag räknade med att många osäkerheter skulle klarna under arbetets gång.

För att få lite mera struktur och någon form av plan för arbetet som skulle göras började jag med att skapa mindre arbetsuppgifter i Jira för att bryta ner den stora helheten så mycket som möjligt, se figur 4.

Figur 4. Arbetsuppgifterna i Jira vid avklarat arbete.

4.2 Visning av slutanvändares mobila enheter

4.2.1 Persistenshantering genom MyBatis

Det första som jag började undersöka när jag skulle börja med denna del av arbetet var att kontrollera vilka databaser och tabeller jag behövde hämta data från och vilken data som faktiskt behövdes, samt att kontrollera hur kopplingen mellan de olika tabellerna såg ut.

När jag väl fått ihop några olika SQL-satser för att hämta den data som skulle komma att krävas av applikation började jag konfigurera applikationen för MyBatis. Denna del visade sig vara mera komplicerad än förväntat, eftersom varken miljö eller applikation faktiskt hade åtkomst till datakällorna som krävdes.

Efter att jag hade fått hjälp på traven av några kollegor som hade rättigheter att ge JBoss-miljön *(Se kapitel 2.9*) åtkomst till datakällorna ifråga kunde det riktiga arbetet börja och jag hade nu möjlighet att börja skriva mappningskonfigurationen som krävdes för den data jag skulle använda. Slutligen kunde jag testa detta och se att information kunde hämtas på det sätt som jag hade tänkt. Ett exempel på en mappningskonfiguration visas i figur 5.

```
<resultMap id="mobileDevice" type="[redacted]customerinfo.backoffice.domain.model.MobileDevice">
    <result property="registrationId" column="REGISTRATION_ID"/>
    <result property="applicationId" column="APPLICATION_ID"/>
    <result property="authenticationLevel" column="AUTHENTICATION_LEVEL"/>
    <result property="authenticationMethods" column="AUTHENTICATION_METHODS"/>
    <result property="lastUsed" column="LAST_USED"/>
    <result property="registeredAt" column="REGISTERED_AT"/>
    <result property="state" column="STATE"/>
    <result property="deviceManufacturer" column="DEVICE_MANUFACTURER"/>
    <result property="deviceModel" column="DEVICE_MODEL"/>
    <result property="operatingSystem" column="OPERATING_SYSTEM_TYPE"/>
</resultMap>
<sql id="selectMobileDeviceValues">
    SELECT
    AUTH_REGISTRATION.REGISTRATION_ID,
    APPLICATION_ID,
    AUTHENTICATION_LEVEL,
    AUTHENTICATION_METHODS,
    LAST_USED,
    REGISTERED_AT,
    STATE,
    DEVICE_MANUFACTURER,
    DEVICE_MODEL,
    OPERATING_SYSTEM_TYPE,
    FROM AUTH_REGISTRATION INNER JOIN
     AUTH_SMART_DEVICE ON AUTH_REGISTRATION.REGISTRATION_ID = AUTH_SMART_DEVICE.REGISTRATION_ID
    WHERE USER_ID = ${customerId}
</sql>
```

Figur 5. Mappningskonfiguration för mobila enheter.

4.2.2 Hantering av data genom Spring MVC

I den här delen av applikationen fanns inte särskilt mycket logik eftersom den endast skulle hämta data baserat på vissa sökvillkor. Detta innebar att jag kunde börja med funktionaliteten som hörde hemma i gränssnittslagret för att användare enkelt skulle kunna söka och visa data på olika sätt. De olika varianterna av sökningar var följande:

- Översikt av en användares alla mobila enheter
- Översikt av alla mobila sessioner under ett visst tidsspann
- Detaljerad information om en mobil enhet med hjälp av registrerings id
- Detaljerad information om en mobil session med hjälp av sessions id

Indata från användaren var enkelt att hantera då funktionaliteten för att söka efter ett tidsspann redan fanns i den äldre delen av applikationen och kunde återanvändas för mina egna syften. Detaljsökningarna var enkla att genomföra då jag endast behövde skapa en länk för varje mobil enhet respektive session som skickar vidare ett identifikationsnummer till korrekt ändpunkt i min Spring MVC kontrollkomponent. Exempel på hur denna implementation ser ut för mobila enheter kan ses i figur 6.

```
/** Method to fetch data about a specific mobile device.
 * @param model model
 * @param registrationId The registration id for the mobile device
 * @return the tiledefinition string
 */
@RequestMapping(value = "/[redacted]MobileDetails", method = RequestMethod.GET)
public String [redacted]MobileDetails(final Model model, @RequestParam(required = true) final String registrationId) {

    final DataResponse<MobileDeviceDTO> response = [redacted]InfoFacade.getMobileDevice(registrationId);
    addResponseInfoToModel(model, response, "mobileDevice");
    return TilesNames.[redacted]_MOBILE_DETAILS;
}
```

Figur 6. Ändpunkt i Spring MVC kontrollkomponent.

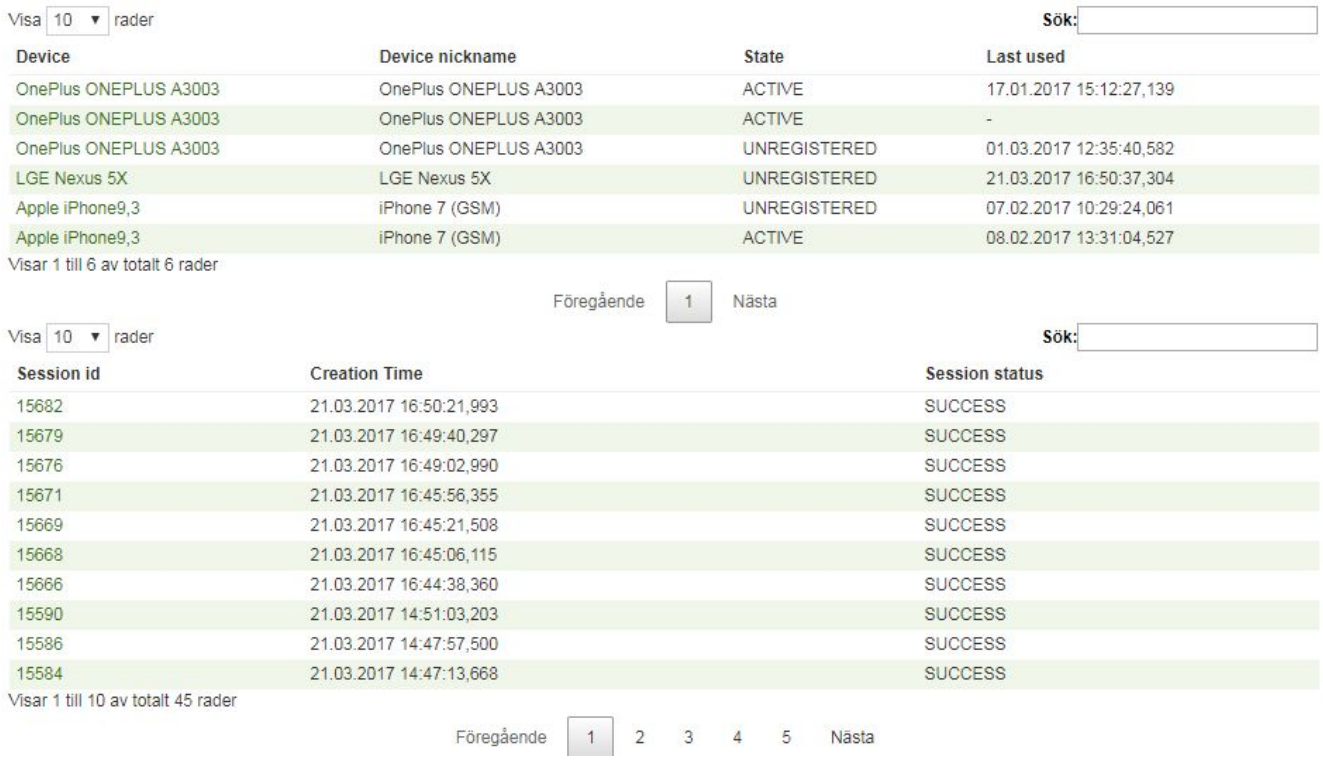
4.2.3 Hantering av vyer genom Spring MVC och Apache Tiles

Att skapa vyerna var relativt enkelt då jag i princip endast behövde skapa ett enhetligt och snyggt sätt att visa den information som hämtades. Eftersom jag utnyttjade Apache Tiles så behövde jag endast döpa min vy och utnyttja det namnet i min Spring MVC kontrollkomponent för att aktivera rätt vy, vilket illustreras i figur 6. Figur 7 visar ett exempel på en Apache Tiles-definition och figur 8 och 9 visar resultatet på denna vy.

```
<definition name="customerinfo.[redacted].mobiledetails.def" extends="mainLayout">
    <put-attribute name="pageTitleKey" value="lbl_customerinfo_title" />
    <put-attribute name="body" value="/WEB-INF/customerinfo/jsp/[redacted]MobileDetails.jsp" />
</definition>
```

Figur 7. Apache Tiles definition för detaljvy över en mobil enhet.

Själva vyerna blev väldigt enkla, men de genomför sitt syfte. Slutresultatet för översiktsvyn visas i figur 8 och detaljvyn för mobila sessioner visas i figur 9.


Visa 10 rader Sök:

Device	Device nickname	State	Last used
OnePlus ONEPLUS A3003	OnePlus ONEPLUS A3003	ACTIVE	17.01.2017 15:12:27,139
OnePlus ONEPLUS A3003	OnePlus ONEPLUS A3003	ACTIVE	-
OnePlus ONEPLUS A3003	OnePlus ONEPLUS A3003	UNREGISTERED	01.03.2017 12:35:40,582
LGE Nexus 5X	LGE Nexus 5X	UNREGISTERED	21.03.2017 16:50:37,304
Apple iPhone9,3	iPhone 7 (GSM)	UNREGISTERED	07.02.2017 10:29:24,061
Apple iPhone9,3	iPhone 7 (GSM)	ACTIVE	08.02.2017 13:31:04,527

Visar 1 till 6 av totalt 6 rader

Föregående 1 Nästa

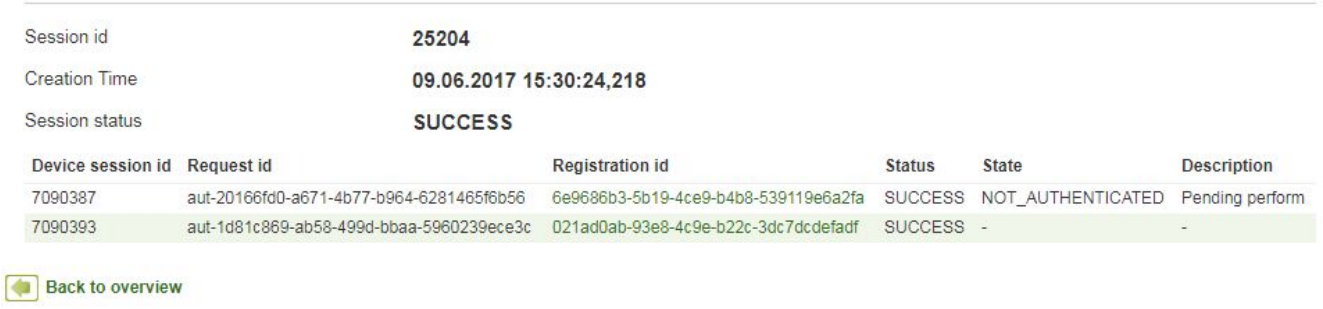
Visa 10 rader Sök:

Session id	Creation Time	Session status
15682	21.03.2017 16:50:21,993	SUCCESS
15679	21.03.2017 16:49:40,297	SUCCESS
15676	21.03.2017 16:49:02,990	SUCCESS
15671	21.03.2017 16:45:56,355	SUCCESS
15669	21.03.2017 16:45:21,508	SUCCESS
15668	21.03.2017 16:45:06,115	SUCCESS
15666	21.03.2017 16:44:38,360	SUCCESS
15590	21.03.2017 14:51:03,203	SUCCESS
15586	21.03.2017 14:47:57,500	SUCCESS
15584	21.03.2017 14:47:13,668	SUCCESS

Visar 1 till 10 av totalt 45 rader

Föregående 1 2 3 4 5 Nästa

Figur 8. Översiktsvy som resultat av en sökning på mobila enheter och inloggningar.


Session id **25204**

Creation Time **09.06.2017 15:30:24,218**

Session status **SUCCESS**

Device session id	Request id	Registration id	Status	State	Description
7090387	aut-20166fd0-a671-4b77-b964-6281465f6b56	6e9686b3-5b19-4ce9-b4b8-539119e6a2fa	SUCCESS	NOT_AUTHENTICATED	Pending perform
7090393	aut-1d81c869-ab58-499d-bbaa-5960239ece3c	021ad0ab-93e8-4c9e-b22c-3dc7dcdefadf	SUCCESS	-	-

Back to overview

Figur 9. Detaljvy för en mobil session.

4.2.4 Förändringar under resans gång

När jag var klar med vyerna och hanteringen av indata i applikationen hade beställaren ändrat sig angående hur vi skulle hämta data till applikationen, då han ansåg att det inte var så bra att introducera ett helt nytt beroende till datakällan från applikationen. Detta innebar att datan skulle hämtas genom några olika utvecklargränssnitt från en annan applikation som redan hade detta beroende. Utvecklingen av dessa utvecklargränssnitt sköttes av en kollega för att inte öka mängden arbete för mig mer än nödvändigt.

När jag väl skulle använda mig av den nya funktionaliteten från applikationen vi skulle hämta data från, så upptäckte jag ytterligare ett bekymmer. Vi hade inte tidigare använt oss av denna applikation så även här introducerades ett nytt beroende och jag behövde ta reda på hur jag skulle få allting konfigurerat för att kunna hämta data från denna applikation. Även när detta var klart fortsatte problemen från den nya förändringen att visa sig, genom att det inte längre var riktigt samma data som hämtades och den kunde inte hanteras och visas på samma sätt som tidigare. Detta innebar ganska mycket förändringar i vyerna som jag hade skapat och även i hanteringen av indata genom Spring MVC.

4.3 Uppdatering av slutanvändare i ett datalager

4.3.1 Förändringar av utvecklargränssnitten

De existerande Spring MVC-baserade utvecklargränssnitt som jag skulle utnyttja för denna del av applikationen var det jag var mest osäker på, så därför började jag med att titta över dessa. Problemet som uppstod med dessa gränssnitt och deras bakomliggande funktionalitet var att dom tidigare endast varit använda för automatiserade batchjobb och därför gav inte den nuvarande funktionaliteten någon form av respons på resultatet av uppdateringen. Jag behövde göra om ganska mycket för att få olika typer av respons då ett fel faktiskt har uppstått.

Jag ansåg att det bästa vore att göra helt nya utvecklargränssnitt för mina egna behov, så att de skulle hållas helt separata från befintliga gränssnitt även om funktionaliteten i bakgrunden var ungefär densamma. Efter att uppdragsgivaren hade godkänt detta förslag kunde jag påbörja arbetet med dessa.

Det mest utmanande med förändringarna i den bakomliggande funktionaliteten var att lyckas hålla den logiska funktionaliteten exakt likadan som den tidigare varit. Det visade sig i efterhand att det fanns några gränsfall vid användningen av denna funktionalitet som blev annorlunda och påverkade såväl nya som gamla gränssnitt, men det var ganska enkelt att lösa.

4.3.2 Konfiguration för utvecklargränssnitten

Utvecklargränssnitten skulle anropas genom en särskild komponent som hanterar själva trafiken mellan applikationerna. Denna komponent behövde konfigureras för den specifika

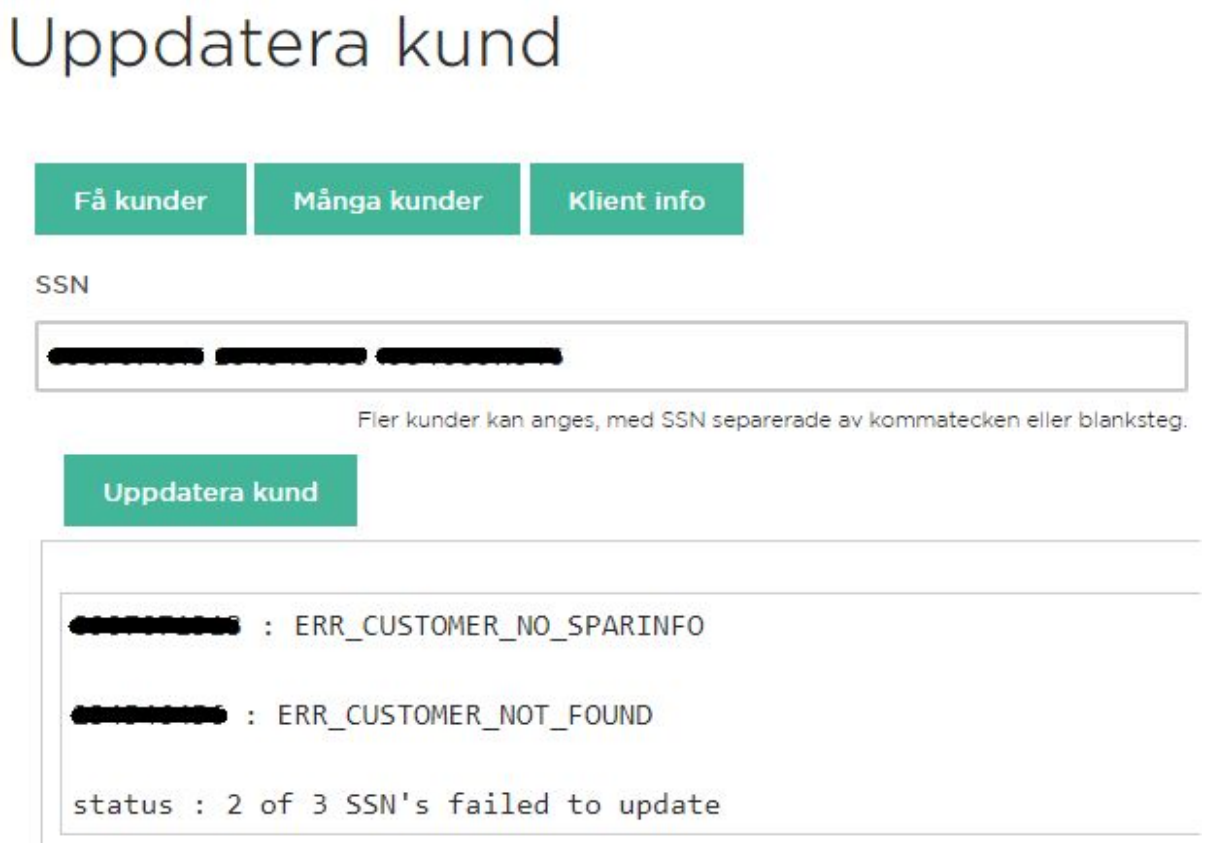
kund som använder den nya funktionaliteten, samt för var den hittar utvecklargränssnitten. Komponenten behövde även konfigureras specifikt för varje distributionsmiljö. (Se kapitel 2.8)

Alla dessa miljöer skulle konfigureras annorlunda och därför behövde jag även undersöka hur dessa konfigurationer skulle göras.

4.3.3 Hantering av data i Spring MVC

I detta skede hade jag allting klart för att påbörja utvecklingen av den egentliga applikationen. Nu kunde jag börja skapa en kontrollkomponent med hjälp av Spring MVC. När jag väl hade lyft in all funktionalitet som jag behövde i denna komponent kunde jag skapa vyer som användaren skulle utnyttja sig av för att ange slutanvändare som skulle uppdateras.

I denna del av applikationen behövde jag dock hantera data mera utförligt än vad jag tidigare gjort. Användarens indata kräver sortering och uppdelning ifall användaren angett flera personnummer eller om användaren har skickat in en fil som innehåller många personnummer (*Se kapitel 4.3.4*). När informationen väl är sorterad så behöver den endast skickas vidare till utvecklargränssnitten som jag skapat, och sedan skicka tillbaka responsen till vyerna så att användaren kan se resultatet. Uppdateringen av ett fåtal slutanvändare visas i figur 10, och uppdateringen av många slutanvändare visas i figur 11.



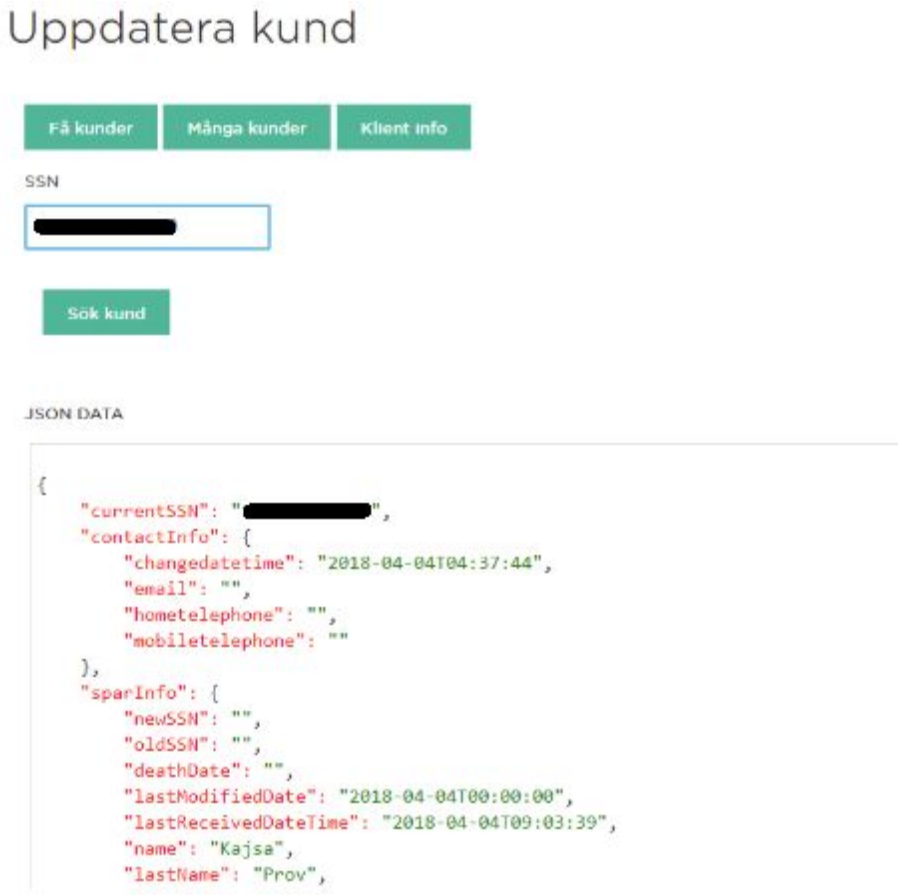
Figur 10. Uppdatering av få slutanvändare.

Figur 11. Uppdatering av många slutanvändare.

Eftersom en användare även skulle kunna hämta rådata om en slutanvändare från datalagret i JSON-format behövde jag även skapa en vy för detta. Detta var en väldigt enkel process då

denna information enligt specifikation endast skulle behövas visas i JSON-format vilket innebar att jag inte behövde hantera denna data på något annat sätt utöver att visa den för användaren.

Denna data var däremot väldigt svårläst för en människa då den endast bestod av en lång sträng med tecken och text utan mellanslag eller radbyten. För denna vy skapade jag därför även lite mera avancerad javascript-kod för att visa datan på ett snyggt vis. Vad denna kod gör är att lägga till indenteringar, radbyten och olika färgsättningar för olika typer av data, såsom heltal, decimaltal och strängar. Denna kod är till största del utformad med några olika reguljära uttryck (*regular expressions)*. Resultatet visas i figur 12.



Figur 12. Hämtning av rådata från datalagret.

4.3.4 Hantering av filer

Kravspecifikationen som jag och uppdragsgivaren hade kommit fram till hade en specifik punkt om att en användare skulle ha möjlighet att ange filer som skulle innehålla många olika personnummer som skulle uppdateras. Jag behövde nu alltså skapa en filhanterare för många olika format på filer för att hantera dem på rätt sätt.

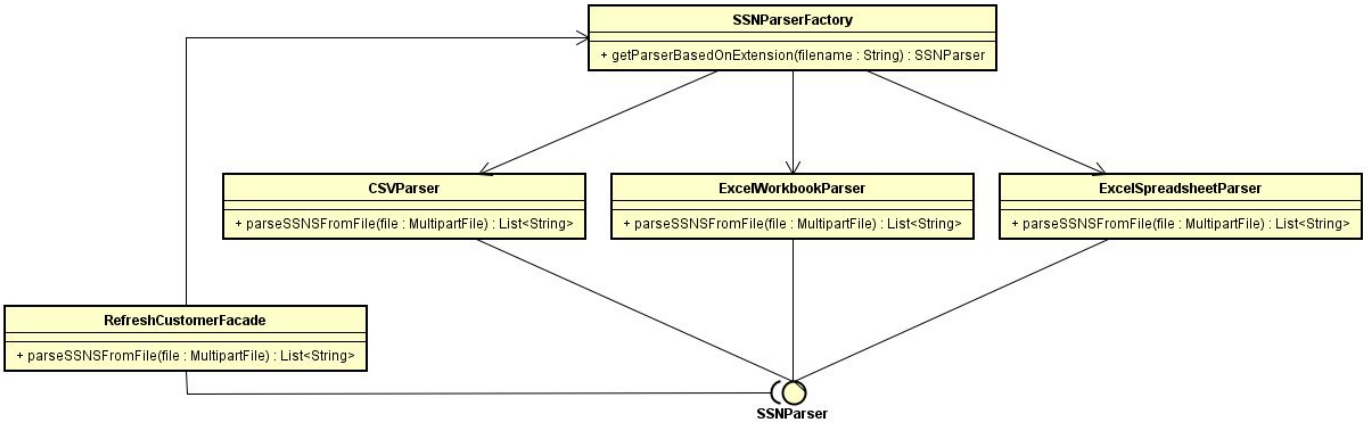
Denna implementation blev slutligen tre olika filhanterare och en *Factory*-design som bygger upp dessa filhanterare baserat på vilken filtyp användaren anger. Denna fabrik returnerar ett gränssnitt med korrekt implementation som man sedan skickar in filen till och får tillbaka en lista med personnummer som sedan kan skickas vidare för uppdatering. Den enkla användningen av denna implementation visas i figur 13.

```
@Override
public List<String> parseSSNSFromFile(final MultipartFile file) throws RefreshCustomerException {
    final SSNParser ssnParser = ssnParserFactory.getParserBasedOnExtension(file.getOriginalFilename());
    return ssnParser.parseSSNSFromFile(file);
}
```

Figur 13. Skapa en filhanterare och hämta personnummer från fil.

Orsaken till att det krävdes tre olika implementationer för något så enkelt som att läsa en fil var att denna filhanterare även skulle kunna hantera Excel-formatet. Detta format har genomgått många olika förändringar sedan det blev populärt och därför ger den första hanteraren endast stöd för de nyare formaten, medan den andra hanteraren ger stöd för de äldre formaten.

Den tredje hanteraren ger stöd för det välkända formatet *Comma Separated Values* och helt vanliga textfiler, då dessa kan hanteras genom en enkel separering av kommatecken och mellanslag. Ett övergripande diagram för denna design visas i figur 14.



Figur 14. Klassdiagram över Factory-designen för filhantering.

4.3.5 Autentisering av användare

Denna funktionalitet skall inte vara tillgänglig för alla användare av applikationen. Därmed behöver användaren autentiseras före denna får tillgång till applikationen. Denna autentisering har jag överlåtit till Spring Security. Rättigheterna som krävdes för att få tillgång till applikationen existerade redan hos vissa användare, så detta var endast en fråga om att låta Spring Security jämföra existerande rättigheter med den rättighet som faktiskt krävdes. Denna autentisering behövdes även i vyerna som användaren ser för att undvika att visa länkar och funktionalitet som användaren ändå inte har behörighet till eller kan utnyttja sig av. Att låta Spring Security autentisera användaren var väldigt enkelt efter jag hade klargjort vilken rättighet som skulle användas, då jag endast behövde lägga till en annotering till metoderna som skulle kräva autentisering. Ett exempel på denna teknik visas i figur 15.

```
/**
 * When posting request for customer information.
 *
 * @param model model
 * @param refreshCustomerRequest request
 * @return Tiledefinition string
 */
@RequestMapping(value = "/getClientInfo.do")
@PreAuthorize(CamadminConstants.PRE_AUTHORIZE_WRITE)
public String getClientInfo(final Model model, final RefreshCustomerRequest refreshCustomerRequest) {
```

Figur 15. Låta Spring Security sköta autentiseringen av en användares rättigheter

5. SLUTSATSER

5.1 Resultat

Detta arbete har uppfyllt beställarens alla krav och förväntningar. Beställaren har i efterhand rapporterat några mindre buggar och mindre saker som skulle förbättra upplevelsen för användaren, men när allting var klart var beställaren väldigt nöjd. All funktionalitet som specificerats enligt kravspecifikation har implementerats och den nya funktionaliteten är väldigt lättanvänd. Resultatet som detta arbete har givit kommer att förenkla det vardagliga arbetet för många utvecklare och även för beställarens kunder.

5.2 Reflektioner

Personligen har detta examensarbete gett mig djupare insikt i ganska många teknologier och ramverk. Det som har varit mest intressant och lärorikt för min del har varit intern kommunikation via utvecklargränssnitt mellan flera olika applikationer. Detta var ett område som jag länge har funderat på hur man gör på ett elegant vis. Inom detta arbete har jag fått utnyttja två helt olika komponenter som blivit utvecklade för just detta. Jag har även fått sätta mig in hur man konfigurerar olika datakällor för en JBoss-miljö, vilket också var lärorikt och troligen användbart i framtiden.

Jag har varit ganska mycket i kontakt med MyBatis sedan tidigare, men aldrig skapat något nytt med hjälp av detta ramverk, så det har även varit intressant att få lära sig mera om hur man konfigurerar och använder det.

Spring Profiles var ett begrepp som jag hade hört talas om före jag började med detta arbete, men jag har haft lite fördomar om hur bra det faktiskt är att använda sig av denna typ av teknologi. Efter att jag hade insett att jag inte skulle kunna lösa vissa bekymmer utan detta verktyg och även fått pröva på hur enkelt det var att utnyttja fick jag slutligen övervinna mina fördomar och erkänna att det är ett riktigt kraftfullt och användbart verktyg.

Att få pröva på att skapa en egen JSP Taglib var även det riktigt intressant och jag upptäckte att jag fick en märklig tillfredsställelse av att skapa logik med hjälp av Java och kunna utnyttja den enkelt direkt i vyerna.

Domändriven design har varit ett välkänt begrepp för mig under en längre tid, men det var egentligen först genom detta examensarbete som jag faktiskt fått god förståelse för vad begreppet egentligen innebär.

Att hålla i ett lite större projekt har även givit mig mycket mera förståelse för strukturering av projekt, särskilt när projektet utnyttjar domändriven design. I detta projekt har jag även fått hålla i ganska mycket på egen hand, vilket varit bra för att ta bort stödhjulen och tvinga mig själv till att ta flera egna beslut.

KÄLLOR

Craig Walls. (2014). *Spring in action:* Manning.

DDD architecture. (2009). Hämtad från http://dddsample.sourceforge.net/architecture.html

Deployment environment. (2018). Hämtad från

https://en.wikipedia.org/wiki/Deployment_environment

Eric Evans, & Martin Fowler. (2003). *Domain-driven design* (1st ed.). England:

Addison-Wesley Educational Publishers Inc.

JBoss documentation. (u.d.). Hämtad från

https://access.redhat.com/documentation/en-us/red_hat_jboss_enterprise_application_platfo

rm/7.1/html/introduction_to_jboss_eap/overview_of_general_concepts

MyBatis documentation. (2018). Hämtad från

http://www.mybatis.org/mybatis-3/getting-started.html

Spring MVC Framework (u.d.). Hämtad från

https://www.tutorialspoint.com/spring/spring_web_mvc_framework