CHALMERS

Smart lagerhållningssystem med RFID

Kandidatarbete inom data- och informationsteknik

Gabriel Andersson
Philip Dahlstedt
Fredrik Ek
Marc Jamot
Daniel Jonsson
Martin Ljungdahl

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Smart Logistiksystem med RFID

Gabriel Andersson
Philip Dahlstedt
Fredrik Ek
Marc Jamot
Daniel Jonsson
Martin Ljungdahl

Examiner: Arne Linde

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract

The aim of this bachelor thesis was to create an automated warehouse management system based on available RFID technology.

Today a lot of effort is being spent on stocktaking due to manual steps within the warehouse management process. The goal with the project was to eliminate the manual steps that emerge from the usage of barcodes and replace them with the automatic system developed in this project.

A prototype has been developed from extensive research and realistic testing of both soft- and hardware. The project has had the privilege to do these tests in a realistic environment at the logistics company Dan Cargo in Gothenburg.

The prototype shows a simple concept that makes automated warehouse management possible, which results in a more effective production both in the short and the long run. The system consists of two parts: a hardware part and a software part. The hardware part consists of an RFID reader and multiple RFID tags that are attached to pallets and pallet slots. The software part consists of a forklift client, station unit, server, database and an administration interface. The system can automatically register a pallet's movement and store it's current wherabouts in the database.

With this foundation, further development is only a small step away and is a likely scenario in the future.

Sammanfattning

Syftet med detta kandidatarbete var att skapa ett system för automatisering av lagerhållning med hjälp av tillgänglig RFID-teknik.

Idag läggs mycket resurser på inventering inom lagerhållning. Detta till följd av manuella moment vid hantering av varor och streckkoder. Målet med projektet var att eliminera de manuella moment som uppstår i samband med användning av streckkoder och ersätta dem med projektets automatiserade system.

En prototyp har tagits fram genom omfattande studier och realistiska tester av såväl hård- som mjukvara. Dessa tester har genomförts på logistikföretaget Dan Cargo i Göteborg, där projektet har haft förmånen att verka i en realistisk lagermiljö.

Prototypen visar på ett enkelt koncept som innebär att lagerhållning sker automatiskt. Något som dessutom innebär en mer effektiv produktion, på såväl kort som på lång sikt. Systemet består av en hårdvarudel och en mjukvarudel. Hårdvaran består av RFID-läsare och flertalet RFID-taggar som fästes på pallar och pallplatser. Mjukvaran består av en truckklient, stationsenhet, server, databas och administrationsgränssnitt. Systemet kan automatiskt registrera förflyttningar av pallar och lagra dessa i databasen.

Med detta som grund så är en vidareutveckling till färdig produkt ett litet steg och något som känns som ett troligt scenario i framtiden.

Förord

Denna rapport är en sammanställning av ett kandidatarbete som genomförts av studenter vid institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola, Datateknik 300 hp och Informationsteknik 300 hp. Arbetets omfattning är 15 högskolepoäng per teknolog (ca 400 h) och har pågått under vårterminen 2014. Projektet som sådant innefattar såväl teoretiska inslag som praktiska tillämpningar.

Vi skulle vilja passa på att tacka vår handledare Lennart Hansson för den hjälp och de synpunkter han bistått med. Vi vill även tacka vår examinator Arne Linde och institutionen för att de bistått oss ekonomiskt, något som underlättat för oss på ett tidigt stadie i projektet.

Ett speciellt tack går även ut till Dan Cargo och deras medarbetare. Inte minst platschef, David Leifsson, som bistått oss med såväl kunskap och idéer. Samt låtit oss utnyttja Dan Cargos lokaler och utrustning för testning och fullföljning av projektet i en realistisk miljö.

Vi vill också tacka Lars-Göran Johansson på företaget RFID Systems som har hjälpt oss med hårdvarurelaterade frågor, speciellt rörande frågor kring användning av RFID i miljöer med mycket metall.

Ordlista/Förkortningar

Beställningslager:
Ett lager vars uppgift är att ta emot och skicka beställningar från kunder i form av företag och privatpersoner.

Entitet:
Ett objekt i en databas. Lagras som en rad i en tabell.

Framework:
Ramverk - Ett ramverk fungerar som en ram för en applikationen och bidrar ofta med en standardstruktur och bibliotek för sådant som databashantering och återanvändning av kod, samt automatisk sammanlänkning av applikationens olika delar.

Mellanlager:
Syftar till lager vars uppgift är att mellanlagra varor då de skickas från exempelvis tillverkare, för att sedan skickas till butiks- och beställningslager.

MVC:
Model view controller - Ett koncept inom webbapplikationsutveckling som innebär att man använder:

- **Modeller** : Domänspecifika objekt, sådant man vill kunna skapa i applikationen, i vårt fall såsom pallar, pallplatser, artiklar etc. Sparas oftast i någon form av databas.
- **Vyer** : Det som visas i applikationens olika vyer med hjälp HTML, JavaScript, CSS etc.
- **Kontroller** : Varje vy har en kontroller som fungerar som dess back-end och avgör hur den ska bete sig. Exempelvis såsom när olika delar av sidan ska laddas.

ROM:
Read Only Memory - Minne som enbart går att läsa från.

RWM:
Read Write Memory - Minne som går att båda läsa och skriva till.

Innehåll

1

Inledning

Idag finns teknik som skulle kunna användas inom logistikbranschen för effektivisering av lagerhållning, men trots detta används ofta äldre teknik av de flesta logistikföretag. Denna outnyttjade potential till effektivisering av verksamheten, som kan ge mer noggrann data om varuflödena på lagret och minska behovet av inventering, är något som projektgruppen identifierat. Projektgruppen har inte bara identifierat denna potential utan dessutom tagit fram en teknisk lösning som teoretiskt skulle kunna innebära eliminering av mänskliga misstag som i sin tur leder till tidsförödande inventering.

1.1 Bakgrund

Den vanligaste och mest aktuella metod för lagerhållning är idag, enligt D. Leifsson (2014) som är platschef på Dan Cargo Göteborg, att märka pallar och pallplatser med streckkoder.

Projektgruppen genomförde inledningsvis ett studiebesök på Dan Cargo för att få bättre insikt i hur arbetsgången ser ut. För att registrera en pall på en pallplats skannar truckföraren manuellt pallens streckkod följt av pallplatsens streckkod. Lagersystemet registerar att den skannade pallen numera står på den skannade pallplatsen. Med detta arbetssätt måste, för varje pall som ställs på en ny lagerplats, truckföraren manuellt skanna både pallen och platsen som pallen ställs på. Detta måste ske varje gång en pall flyttas.

Dan Cargo, som projektgruppen samarbetat med, har ett flöde på cirka 500 pallar som kommer in till lagret och 500 pallar som åker ut från lagret varje dag. Utöver detta sker enligt D. Leifsson (2014) åtskilliga runtflyttningar inne på själva lagret varje dag.

1.2 Syfte

Syftet med projektet är att med hjälp av RFID-teknik skapa ett tillförlitligt lagerhållningssystem som effektiviserar arbetet på ett lager. Lösningen fokuserar på hantering av pallar och kommer därmed inte vara till lika stor användning vid ett *beställningslager*, kontra ett lager som hanterar *mellanlagring*.

Målet är en självständig produkt som ska fungera oberoende av det system som redan används på logistikföretaget. Lösningen ska dessutom kunna fungera som en insticksmodul till nuvarande existerande logistiksystem vid behov.

1.3 Problem

De problem som projektgruppen ställts inför innefattar allt från att köpa in lämplig hårdvara till att utforma olika delar av mjukvara, i olika programmeringsspråk. Dessa olika delar måste sedan länkas samman så att de fungerar tillsammans som ett komplett system.

Projektgruppen har behövt lära sig mycket om en, för dem, främmande bransch. Detta för att bland annat kunna ta ställning till placering av RFID-läsare och -tagg för att möta lagermässiga problem som kan inträffa. Samt lösa tekniska problem som att den mjukvara som implementerats ser till att pallar och platser registreras på ett sätt som gör lagerhållning automatisk.

Den färdiga lösningen ska vara mer kostnadseffektiv än det system som används i nuläget. Slutkostnaden sätts således i relation till befintliga system gällande felsäkerhet, stabilitet och tidsbesparning. Installationsprocessen måste även vara resonabel; inte bara ur ett ekonomiskt perspektiv, utan även ur ett praktiskt. Praktiskt på så vis att personalen inte skall vara i behov av utbildningar för att använda systemet. Samt att installation ska kunna utföras utan att det försvårar eller stoppar upp arbetet.

Tillsammans med Dan Cargo sattes en kravspecifikation upp i början av projektet, se bilaga A.

1.4 Logistikbranschen idag

Dan Cargo, liksom de flesta andra logistikföretag, använder metoden med streckkoder som beskrevs i avsnitt 1.1. Det arbetsätt som beskrivs medför att det krävs flera tusen manuella skanningar dagligen för att tillgodose lagerhållning på ett företag som Dan Cargo.

Det finns även exempel på lager i Sverige där de förlitar sig på att personalen som sköter lagerhållning skriver ner rätt uppgifter på papper och att detta sedan manuellt läggs in i det system som används. Detta bekräftar både D. Leifsson (2014), platschef

för Dan Cargo Göteborg, och E. Laago (2014), lagerchef på IKEA Torsviksterminalen, i personliga intervjuer.

Oavsett vilken av ovanstående beskrivna metoder som används genereras fel på grund av mänskliga faktorer. Tid och resurser får därmed läggas på att leta efter artiklar som försvunnit, samtidigt som behovet av inventering ökar.

1.5 Alternativa lösningar

Enligt D. Leifsson (2014), platschef Dan Cargo Göteborg, har RFID-baserade logistiksystem blivit vanligare på senare tid och det handlar då främst om märkning av containrar. Gällande lagerhållningssystem är det däremot inte lika vanligt, där endast ett fåtal stora företag har tillämpat sådan teknik. Det handlar då om företag som Walmart, Amazon och andra stora butikskedjor där hela logistikkedjan använder sig av tekniken [1].

De företag som erbjuder installation av liknande tjänster ger begränsad information kring implementationen. Ett exempel på ett företag som erbjuder RFID-baserade lösningar för logistikbranschen är det finska Vilant [2].

Det finns dessutom lagersystem som är effektivare på andra sätt än med hjälp av RFID. Exempel på detta är automatiserade lager där lagerhållning sker med hjälp av robotar hos bland annat IKEA, enligt E. Laago, och Komplett [3].

1.6 Samarbete med Dan Cargo

Då syftet var att bygga en prototyp tillkom möjligheten att verifiera funktionaliteten hos företaget Dan Cargo. Dan Cargo är ett danskägt tredjepartslogistikföretag med huvudkontor på Hisingen i Göteborg. Där har även deras logistikexperter kunnat tillföra inspiration och kunskap till projektet. Samarbetet uppkom genom att en av personerna i projektgruppen tidigare varit anställd hos Dan Cargo och därigenom etablerat kontakt med företaget.

Samarbetet har gått ut på att Dan Cargo, genom projektgruppens kontinuerliga studiebesök, fungerat som ett bollplank då tekniska lösningar har presenterats. Utöver detta har de även bistått projektet genom att tillåta genomförning av tester i deras lokaler, med deras verktyg och maskiner. Detta har varit till stor nytta för projektet då det hade varit svårt att simulera en lagermiljö i en sådan stor skala.

1.7 Arbetsgång

Projektet inleddes med inläsning och efterforskning. Något som i sin tur ledde till att medlemmar av projektgruppen fattade intresse för olika delar av projektet. Projektet delades således upp i två delar där två personer arbetade med att ta fram hårdvara

och fyra personer arbetade med att ta fram en modell för mjukvara. Efter detta delades projektet upp ytterligare genom att mjukvarugruppen blev två mindre grupper. Den ena fick ansvaret att ta fram klientmjukvara medan den andra fick ansvaret till att bygga upp servern och administrationsverktyget. Denna arbetsmetod följdes inledningsvis, men ganska så snart jobbade alla mer eller mindre även över gruppgränserna.

Projektarbetet har utförts agilt med veckovisa möten. Vid varje möte har arbetsuppgifter delats ut och produkten har iterativt byggts fram och förbättrats efter önskemål och förslag. För att underlätta arbetet inom grupperna har verisionshanteringssystemet Git [4] använts. Utöver detta har även ärendehanteringssystemet hos Bitbucket [5] använts där uppgifter och fel lagts in som sedan kunnat ges prioritet samt tilldelas olika medlemmar i projektgruppen.

1.8 Rapportens upplägg

Kapitel 2 om teknisk bakgrund ger en bakgrund till RFID-teknik, begränsat till vad som anses relevant för projektet. Dels beskrivs dess egenskaper, vad som skiljer de olika frekvensstandarderna åt, samt användningsområden för RFID.

Kapitel 3 om systemdesignen ger läsaren en överblick över hur systemet fungerar på en hög nivå. Systemets arkitektur beskrivs, vilket innefattar både mjuk- och hårdvara. Vidare redovisas de avgränsningar som gjorts för att hålla projektet inom ramen för ett kandidatarbete.

Kapitel 4 om implementationen beskriver hur systemets delar faktiskt är implementerade på en teknisk nivå. En beskrivning ges av hur mjukvarudelarna är uppbyggda, vilket består av ett administrationsverktyg, en databasmodell och ett par klienter som kommunicerar med både server och hårdvara. Utöver mjukvaran beskrivs även den inköpta hårdvaran, vilket innefattar RFID-läsare och -taggar.

Kapitel 5 om slutprodukten visar upp projektets slutprodukt ur en användares perspektiv och hur användaren kommer uppleva produkten. Kapitlet fokuserar på demonstration av både administrationsverktyget och mjukvaran som kommunicerar mellan RFID-läsare och server. Med hjälp av skärmdumpar demonstreras arbetsflödet och prototypens funktioner.

Kapitel 6 innehar en diskussion av olika aspekter inom projektet som inte framgått i de tidigare kapitlen. Detta är bland annat avvägningar vad gäller mjukvara och hårdvara, alternativ som undersökts och testats, hur projektet kan utökas och hur väl projektgruppen tror att produkten kan fungera på ett riktigt lager.

Kapitel 7 är rapportens slutsats och innehåller en avrundning av rapporten där några sista ord ges.

2

Teknisk bakgrund

RFID står för *Radio-frequency identification*. Det är en teknik där radiovågor används för att läsa information och skicka data i syfte att automatiskt spåra och identifiera taggar fästa på objekt. Detta är inte helt olikt tekniken med streckkoder som också används för liknande ändamål, till exempel för att spåra matvaror, lagerpallar, biblioteksböcker och produkter i detaljhandeln. RFID har dessutom ytterligare egenskaper som möjliggör vidare användningsområden.

2.1 RFID-teknikens historia

Forskning kring RFID är långt ifrån ny. Det var i Storbritannien under andra världskrigets som radiovågor för första gången användes för identifikation [6]. Britterna utrustade varje flygplan med en radiomottagare som i sin tur sände ut en signal som svar när en signal från en av radarstationerna på marken mottogs. Detta gjorde det möjligt att identifiera vilka flygplan som tillhörde de allierade och vilka som tillhörde fienden [7]. Trots denna tidiga användning av radiofrekvenser dröjde det dock fram till 23:e januari 1973 innan det första patentet beviljades där en passiv tagg användes [8].

Idag, år 2014, är RFID-marknaden värderad till 9,2 miljarder USD globalt och förväntas stiga till 30,24 miljarder USD år 2024 [9]. En enkel RFID-läsare går idag att köpa för under 15 USD [10]. År 2005 kostade en tagg 0,22 USD [1] och år 2014 kan en passiv tagg köpas för 0,15 USD när de beställs i stor volym [11].

2.2 Övergripande egenskaper hos RFID

Till skillnad från streckkoder och laser använder RFID-tekniken radiovågor för dataöverföring och därför behöver det inte vara fri sikt vid informationsöverföring [12][1]. En

annan egenskap är att vissa RFID-läsare kan läsa flera taggar samtidigt [6]. RFID stöter dock på en del problem i miljöer med metall då metallen delvis kan absorbera den elektromagnetiska strålning som läsaren alstrar för att kunna läsa av taggen. Detta ger effekten av minskat läsavstånd, förvrängda signaler eller ingen signal alls [6].

En RFID-tagg består av minst två delar: en integrerad krets för att spara och bearbeta data och en antenn [6]. Minnet på den integrerade kretsen kan vara av typen ROM och kommer då förprogrammerat med ett unikt ID-nummer [6]. Det finns även taggar som har ett minne som går att skriva till, för att lagra information från sensorer eller information om objektet det är fäst på, så kallade RWM [6]. RWM-taggar är dyrare då de innehåller fler komponenter än de förprogrammerade [6].

Det finns tre typer av taggar: passiva, aktiva och semipassiva [6][1]. Passiva taggar saknar egen strömförsörjning [6][1]. Vid användning av passiva taggar skapar läsaren ett elektromagnetiskt fält som via induktans driver taggens kretsar om fältet är tillräckligt starkt [6][1]. Maximalt läsavstånd är för passiva taggar, med de regleringar som finns på maxeffekt för läsare, 10 meter [13].

Aktiva taggar har egen strömförsörjning, oftast i form av ett batteri, och de är därmed inte beroende av läsarens elektromagnetiska fält för att kunna kommunicera med läsaren [1]. Därmed kan signaler från aktiva taggar nå läsare som är upp till 1 kilometer ifrån [1]. Aktiva taggar är på grund av batteriet både dyrare och större än passiva taggar [14].

I semipassiva taggar drivs enbart mikrochipet i taggen av ett batteri, inte sändningen av signaler [1]. Dessa taggar används ofta när de behöver vara försedda med sensorer för temperatur eller liknade [1].

2.3 Frekvenser

Inom EU finns det för närvarande fyra olika frekvensband som får användas för RFID [15]. De två högre frekvensbanden inom RFID som får användas har, på grund av skillnader i frekvensregleringar, olika frekvensindelningar i olika delar av världen [16]. Kommunikation inom dessa fyra frekvensband innebär olika för- och nackdelar då egenskaper inom banden är olika. Dessa är kort nämnda i tabell 2.1.

2.3.1 Lågfrekvent 125 kHz

Det största användningsområdet för lågfrekvent RFID är inom passersystem och djurhållning [6]. På grund av den låga frekvensen har den också låg interferens med metall eller vätskor [6]. Räckvidden är max 60 centimeter [6]. En tagg kostar cirka 1 USD per styck [18]. Lågfrekvent RFID är det enda frekvensbandet som inte har någon reglering på maxeffekt [15].

Tabell 2.1: Tabellen visar generella skillnader mellan olika frekvensband. Frekvenserna som anges gäller för EU. Inspirerad av "RFID - en teknologi för förbättrad lagerhantering" [17].

	Lågfrekvent (LF) 125 -135 kHz	**Högfrekvent (HF) 13,56 MHz**	**Ultrahögfrekvent (UHF) 865-868 MHz**	**Superultrahögfrekvent (Microvågor) 2,45 GHz**
Räckvidd	>1m	1-1,5m (passiv)	1,5-7m (passiv) < 100 meter (aktiv)	1-10 meter (passiv) 30m (aktiv)
Exempel på ändamål	Passersystem, djur	Logistik, Bibliotek, Smart cards	Logstik, Biltullar, Containrar	Biltullar, Tåg, Biltävlingar, Containrar
Penetration	Mycket god, även genom vätska	God, även genom vätska	Måttlig, stoppas av vätska eller människokropp	Dålig
Strålvinkel	Mycket rundstrålande (ca 300°)	Ca 30° sektor åt två håll	Smal stråle (passiv) 120° (aktiv)	Mycket smal stråle
Aktiv eller passiv	Oftast passiv	Oftast passiv	Aktiv eller passiv	Aktiv eller passiv
Datakomm. Hastighet	Låg	Högre	Ännu högre	Högst
Lästa per sek	Få	Fler	Ännu fler	Flest

2.3.2 Högfrekvent 13,56 MHz

Högfrekventa system är mest använt på grund av att taggarna är billigare än lågfrekventa system och de har funnits på maknaden längre än ultrahög- och superultrahögfrekventa system [6]. Ett högfrekvent system används exempelvis ofta på bibliotek, i passerkort och inom logistik [6]. Överföringshastigheten ligger på 26 kb/s [19]. Räckvidden är upp till en meter med passiva taggar [6]. En sådan tagg kostar idag cirka 0,5 USD [18].

2.3.3 Ultrahögfrekvent 865-868 MHz

Största användningsområden är inom biltullar och spedition, speciellt i spedition med containrar [6]. Exempelvis har Maersk ett system där de sätter en tagg på varje container, vilket gör att de sen kan skanna ett helt skepp med hjälp av en avancerad läsare, enligt Olof Laago i en personlig intervju den 4 april 2014. Räckvidden för en passiv tagg är upp till 10 meter [6]. Med aktiva taggar kan en räckvidd på upp till 100 meter nås [17]. En passiv tagg kostar idag cirka 0,15 USD [18].

2.3.4 Superultrahögfrekvent 2450-5800 MHz

Ett superultahögfrekvent system bygger på mikrovågor och används bland annat för biltullar, tåg och containrar [17]. På grund av dess höga dataöverföringshastighet kan en sådan läsare snabbt läsa många taggar inom detta band[17]. På grund av den höga frekvensen kan en läsare dessutom läsa taggar som är bakom, eller till och med begravda i någonting annat, så länge det inte är metall eller vätska [1]. Räckvidden för en passiv tag är max 10 meter [6].

3

Systemdesign

I detta projekt finns flertalet distinkta komponenter som arbetar tillsammans för att den slutgiltiga produkten ska fungera. Det är RFID-hårdvaran, en server som bearbetar och lagrar data, ett administrationsverktyg, och slutligen två typer av klienter som interagerar med både servern och hårdvaran. För att hålla projektet inom en rimlig storlek har vissa avgränsningar även gjorts. Kapitlet ämnar att ge en övergripande bild av hur systemet har tagits fram.

3.1 Problemdomän

Följande stycken behandlar den analys som gjorts av projektgruppen gällande mjuk- och hårdvara.

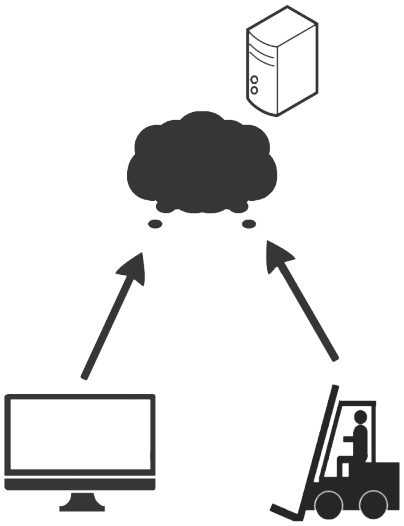
3.1.1 Mjukvara

Då ett lagersystem kräver aktuell data behöver systemet en central databas där data om RFID-taggar, pallar, pallplatser och pallars förflyttningar lagras. Olika typer av klienter ska enkelt kunna komma åt denna data, vilka inom projektet är administrationsverktyget, truckdatorn och stationsklienten. Detta förhållandet illustreras i figur 3.1. Det är även aktuellt att i framtiden kunna ansluta externa logistiksystem till den centrala databasen.

Servern utgör systemets mittpunkt och det är där den mesta logik sker. Dess roll är att ansvara för databasen. Det är även servern som klienterna kommunicerar med när de efterfrågar data från databasen. Exempelvis kan det hända att en klient behöver veta vilka pallar som finns på lagret, och då är det serverns uppgift att tillhandahålla den informationen som den i sin tur hämtar från databasen.

Som inloggad användare i administrationsverktyget är tanken att det enkelt ska gå att få en översikt över lagrets status. Statusen ska innehålla vad som finns på lagret samt dess position. Det ska även gå att se vilka förflyttningar som har genomförts för en viss pall. Verktyget ska även ge användaren möjlighet att lägga till nya platser, pallar och taggar i systemet.

Truckklientens uppgift är att registrera när pallar flyttas och rapportera händelserna till servern. Detta är en enklare klient än administrationsverktyget, men den måste ha högre tillförlitlighet på grund av sina kritiska moment. Om kontakten till servern bryts ska händelserna köas och hanteras när kontakten återfås. För att minimera mängden data som behöver skickas mellan server och klient läggs logiken på serversidan, så att servern tar beslut baserat på vilka taggar trucken har skannat.



Figur 3.1: Översiktsbild som visar två klienter som kommunicerar med en central server. Dels är det en användare via administrationsverktyget på sin dator och dels är det lagertruckens dator.

Stationsklienten är en stationär klient vars uppgift är att ta bort och lägga till pallar i systemet. Ett scenario är att skanna in uppmärkta pallar då de kommer in på lagret. Datan som stationsklienten då skall skicka består av pallens två taggar, datum, tid när den skannades samt vilka artiklar som finns på pallen. Stationsklienten är inte lika kritisk, utan är till för att göra vissa rutinuppgifter enklare. Istället för att manuellt via administrationsgränssnittet behöva knappa in varje position och tagg till varje pallplats, då de ska läggas in i systemet, ska man med hjälp av stationsklienten kunna automatisera delar av ett sådant rutinuppdrag. I fallet med att lägga in nya pallplatser ska det, med stationsklienten, vara möjligt att kunna gå längs en rad pallplatser och automatiskt associera taggen på en pallplats med positionen i systemet.

3.1.2 Hårdvara

Ett beslut som togs inledningsvis var att montering av taggar skulle ske så att två taggar fästs på varje pall och en tagg på varje plats. För att pallar ska kunna plockas upp ifrån båda hållen ska en RFID-tagg sättas på varsin kortsida av pallen. Taggarna kan sedan tas bort från pallen när den skall skickas iväg, och på så vis kan taggar återanvändas. RFID-läsare sätts sedan på varje truck. Denna metod blir praktiskt implementerbar om truckens läsare någon gång kan läsa pallens respektive pallplatsens RFID-tagg under flyttningsprocessen. På så vis kommer systemet alltid uppdateras med rätt information och samtidigt ha få instanser som kommunicerar med servern.

Att använda RFID i närheten av metall orsakar störningar. Dessa störningar är mindre vid låga frekvenser och därför valdes ett lågfrekvent RFID-system. Störningar uppkommer dock även i låga frekvenser om taggen sitter i direkt kontakt med metall. En effektiv lösning till detta är att montera RFID-taggar på gummidistanser som i sin tur monteras på metallbalkarna. Med andra ord reduceras störningarna från miljön och taggar kan användas enligt detta koncept.

3.2 Avgränsningar

Det har i detta projekt gjorts en del avgränsningar vad gäller hård- och mjukvara. De flesta avgränsningar fanns i beaktande från början men en del har även uppkommit under projektets gång.

3.2.1 Mjukvara

Sedan projektets start har målet varit att skapa ett lagerhållningssystem snarare än ett fullskaligt logistiksystem. Idag använder de flesta företag dessutom redan etablerade logistiksystem som fungerar väl. Kompletta logistiksystem är något projektgruppen saknar domänkunskap om. Det kan även tänkas vara enklare att motivera ett företag att investera i en fristående lösning för lagerhållning än att köpa in ett nytt komplett system. Speciellt om denna fristående lösning dessutom är kompatibel med deras redan etablerade system. Detta är en av de främsta anledningar till att bygga en specialiserad lösning som sedermera kan integreras med existerande system.

3.2.2 Hårdvara

Det bestämdes tidigt att hårdvara i form av RFID-läsare och -taggar inte skulle byggas på egen hand. En anledning är att det hade krävts mycket tid för något som bara är en liten del av projektet. Samt att det har funnits RFID-läsare väldigt länge och det finns därmed många bra, vältestade och dokumenterade produkter som skulle passa till ändamålet. Dessutom skulle det vara opraktiskt om alla produkter som krävs av projektet skulle behöva tillverkas när existerande alternativ redan finns.

4

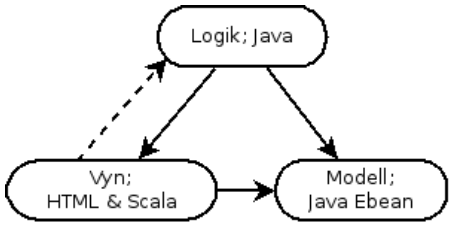
Implementation

Mjukvaruarkitekturen är implementerad med en central databas och ett flertal olika mjukvarukomponenter för att fungera enligt den översiktstbild som visas i figur 3.1. Mjukvarukomponenterna innefattar databasen, administrationsverktyget, klientmjukvaran samt hur kommunikationen mellan klient och server fungerar. Hårdvaran som är implementerad i prototypen består av en RFID-läsare samt av RFID-taggar.

4.1 Administrationsverktyget och Play Framework

I projektet ingår ett webbaserat administrationsverktyg, vilket valdes då det kan användas i alla standardwebbläsare. Det är utvecklat för att vara enkelt, användarvänligt samt smidigt att demonstrera.

För att underlätta utvecklingen av administrationsverktyget byggdes det med hjälp av ett webbapplikationsramverk. Det ramverk som valdes var Play Framework som är skrivet i Java och Scala och strukturerar byggandet av webbsidor enligt MVC-mönstret [20]. Det finns många ramverk som sköter samma uppgift, Play valdes dock för att projekgruppen redan var bekanta med Java. Ramverket sköter allt från att hantera sockets och vilken URL som leder till vilken kontroller, till att bygga HTML-svaren från mallar. Den gör det dessutom enkelt att skriva tester till HTML-vyerna. Det som är kvar för utvecklaren att implementera är det som är unikt för dess webbsida. Funktionalitet som implementerades för projektet var bland annat databasmodellerna med hjälp av Java-biblioteket Ebean, input-logik (controllers) i vanlig Java-kod samt vyerna med hjälp av HTML-mallar byggda med inbäddad Scala-kod. Detta förhållande illustreras i figur 4.1. Play sköter sedan sammanlänkandet av dessa olika applikationsdelar.



Figur 4.1: Relationen mellan komponenterna i en MVC-arkitektur. Modellen är helt fristående, vyn presenterar data från modellen och logiken, också kallad kontroller, hanterar användarens interaktion. I detta projektet är logiken skriven i Java, modellen är skriven i Java med Ebean-notation och vyerna är mallar skrivna i HTML med hjälp av Scala.

Play Framework stödjer även att logik och modeller kan skrivas i Scala, men Java valdes då alla i projektgruppen var bekanta med språket sedan tidigare. Logiken i HTML-vyerna måste däremot fortfarande skrivas i Scala, men det handlar då oftast om mindre avancerad kod. Projektet använder sig av Java-biblioteket Ebean som kommer inbyggt i Play, vilket beskrivs i avsnitt 4.2.

Grunden till alla vyer utgörs av HTML. CSS-stilmallen från Bootstrap-ramverket används som grund för designen på hemsidan då det är ett välanvänt ramverk som dessutom har öppen källkod. Vidare används Bootstraps JavaScript för det visuella. Filtreringsfunktionerna som har implementerats i loggvyn är däremot egenskrivna i JavaScript.

4.2 Modell av domänen och databasen

Lagersystemet har en klar struktur över de objekt som finns samt de handlingar som kan utföras. Dessa representeras med fördel som modeller av objekt. För att interaktion med databasen ska ske med enkelhet används, som tidigare nämnt, Java-biblioteket *Ebean* [21]. Ebean är en så kallad Object Relational Mapper vilket gör det möjligt att definiera databasmodellerna direkt i Java som klasser. Ebean abstraherar alltså bort den underliggande databasen, vilket gör det möjligt att läsa, skriva, uppdatera och radera databasobjekt i Java-kod som vanliga metodanrop. Detta utan att hänsyn behöver tas till om den underliggande databasen är MySQL, PostgreSQL, SQLite eller liknande relationsdatabashanterare. Ebean genererar automatiskt SQL-kod som fungerar till den underliggande databasenhanteraren för att skapa de tabeller som motsvarar de klasser man definerat i Java. Detta kan ses i figur 4.2.

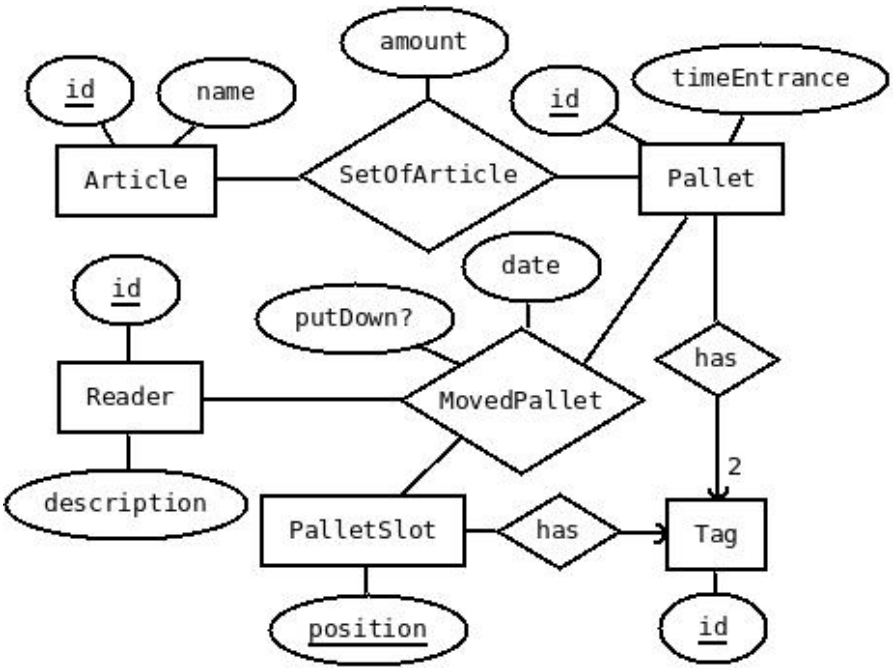
För lagring av data valdes relationsdatabashanteraren MySQL. Den valdes då det är en av de mest välanvända databashanterarna samt att den är licensierad som fri programvara.

```java
@Entity
public class Article extends Model {
    @Id
    @Column(length = 30)
    private String id;

    @Basic(optional = true)
    @Column(length = 50)
    private String name;

    @OneToMany(mappedBy = "article")
    private List<SetOfArticle> partOfSets;
}
```

```sql
CREATE TABLE article (
  id    VARCHAR(30) NOT NULL,
  name  VARCHAR(50),
  CONSTRAINT pk_article PRIMARY KEY (id)
);
```

article	
id	**name**
ART001	Shoes
ART002	Boots
ART003	Sandals

Figur 4.2: Figur som visar hur Java Ebean-kod översätts till en SQL-tabell. Till vänster ses Java-kod uppmärkt med Ebean-notation. Java-modellen av en artikel översätts därefter till SQL-kommandot som ses till höger för att skapa en motsvarande databastabell. Tabellen i figuren visar hur data sedan skulle kunna se ut i databasen.



Figur 4.3: Databasmodellen som har implementerats. Dels finns det pallar (*Pallet*) och pallplatser (*PalletSlot*) som båda har RFID-taggar (*Tag*). Vidare är det möjligt att ställa artiklar (*Article*) på en pall via relationen *SetOfArticle*. En RFID-läsare (*Reader*) kan registrera förflyttningar av pallar mellan pallplatser med hjälp av relationen *MovedPallet*.

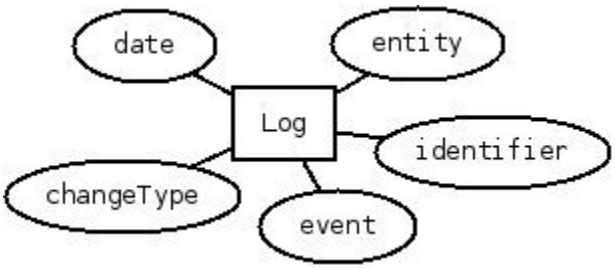
Databasmodellen av domänen kan ses i figur 4.3. *Article* lagrar information om artiklarna, vilket är artikelnumret och eventuellt dess namn. Tabellen *Pallet* lagrar information om varje pall som finns på lagret och har kolumner för dess två unika RFID-taggar, en tidsstämpel när pallen kom in på lagret och ett unikt ID. För att definiera vilka artiklar som står på en pall används relationen *SetOfArticle*. *SetOfArticle* binder ihop en pall och en artikel tillsammans med ett heltal som säger hur många enheter av artikeln som står på pallen. Detta upplägg skapar kompatibilitet för att kunna ha flera olika artiklar på samma pall, även om det kanske inte är situationen i de flesta fall. Vidare finns en tabell, *PalletSlot*, med alla pallplatsers unika plats-ID:n. För att göra det enklare att förhindra att en ny pall, eller pallplats, skapas med en tagg som redan existerar, används tabellen *Tag* där alla unika RFID-taggar som används av pallar eller pallplatser sparas. Tabellen *Reader* innehåller ID:n för truckklienterna. Slutligen finns relationen *MovedPallet* som berättar när en truck *Reader* lyfte upp eller satte ner en pall *Pallet* på en pallplats *PalletSlot*. Om pallen inte ställs ner på en pallplats, det vill säga att den med största sannolikhet ställts på golvet, skapas en relation till en *PalletSlot* med ID "*floor*".

Syftet med tabellen *Log*, som visas i figur 4.4, är att återberätta alla händelser som sker i systemet. Tabellen har inga relationer till domänmodellen hos lagerhållningssystemet för att förhindra att historiken ändras när entiteterna ändras. Historik skulle försvinna om entiteter tas bort från systemet, samt om till exempel en pallplats tilldelas samma positions-ID som en tidigare skulle historiken inte hänga med. Men i och med att loggtabellen nu är helt fristående medför det att alla händelser i lagret kommer att behållas och kan kontrolleras i efterhand. Tabellen är strukturerad på det viset att dess kolumner är *entity*, *identifier*, *event*, *changeType* och *date*. Kolumnen *entity* håller en sträng som förklarar vad händelsen berör, med andra ord om den berör en artikel, en pall, en förflyttning eller en pallplats. Kolumnen *identifier* håller variabeln för de olika typer av entiterna för att kunna finna all historik kring en specifik entitet; *event* innehåller en förklarande textsträng om händelsen; *changeType* består av ett heltal för att underlätta generaliseringen av olika händelser; *date* är tidpunkten för den loggade händelsen.

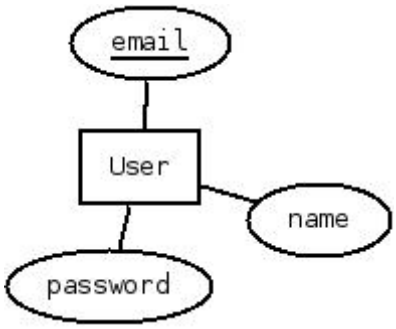
För att lagra information om användare, vilket innefattar användarens namn, e-postadress, och ett krypterat lösenord, används en tabell *User* som kan ses i figur 4.5.

Slutligen finns det en tabell som heter *Play_Evolutions* och som enbart används av Playramverket för interna sysslor. Läs mer om Play i avsnitt 4.1.

Utöver ovannämnda tio tabeller består databasen också av fyra vyer och fjorton triggers. På grund av bristande stöd i Ebean är dessa skrivna manuellt i SQL. Vyerna är till för att förenkla queries till databasen som berör relationen mellan pallar och pallplatser, som exempelvis att få en lista på vilka pallplatser pallar står på just nu. Dessa vyer syns i tabell 4.1. Tolv av de triggers som används är till för att skapa händelser i loggtabellen, när händelser såsom att artiklar, pallplatser eller pallar skapas, ändras eller tas bort etc. Att ha databas-triggers som automatiskt lägger data i loggtabellen innebär att det inte behöver göras manuellt i Java-koden. Någonting som skulle innebära en risk för att det



Figur 4.4: Databastabellen där logghändelserna sparas. Varje händelse har ett datum (*date*); *entity* och *identifier* för att identifiera ett objekt, exempelvis en pall eller en pallplats; *event* är en textsträng som beskriver händelsen och slutligen *changeType* som generaliserar händelser till kategorier, till exempel när en entitet lagts till, tagits bort eller modifierats.



Figur 4.5: Databastabellen för användarna i systemet. Varje användare har en unik e-postadress, ett krypterat lösenord och ett namn.

görs på fel sätt, blir jobbigare att uppdatera, underhålla och att det helt enkelt finns en risk att olika uppgifter glöms bort. Resterande två triggers undersöker om en tagg redan används på en pall eller pallplats. Alla triggers kan ses i tabell 4.2.

Tabell 4.1: Systemets olika vyer och deras respektive syften.

Vynamn	**Beskrivning**
pallet_on_slot	Listar vilka paller som finns på vilka pallplatser.
pallet_on_slot_helper1	Listar senaste tillfället för alla pallar som funnits på alla platser.
pallet_on_slot_helper2	Filtrerar bort alla tidigare platser om en pall ställts på en plats senare.
pallets_on_move	Undersöker om en pall är lyft, men inte nersatt på en ny plats.

Tabell 4.2: Systemets triggers, när när de aktiveras samt vad som sker när de aktiveras. Observera att de flesta enbart skriver till tabellen Log.

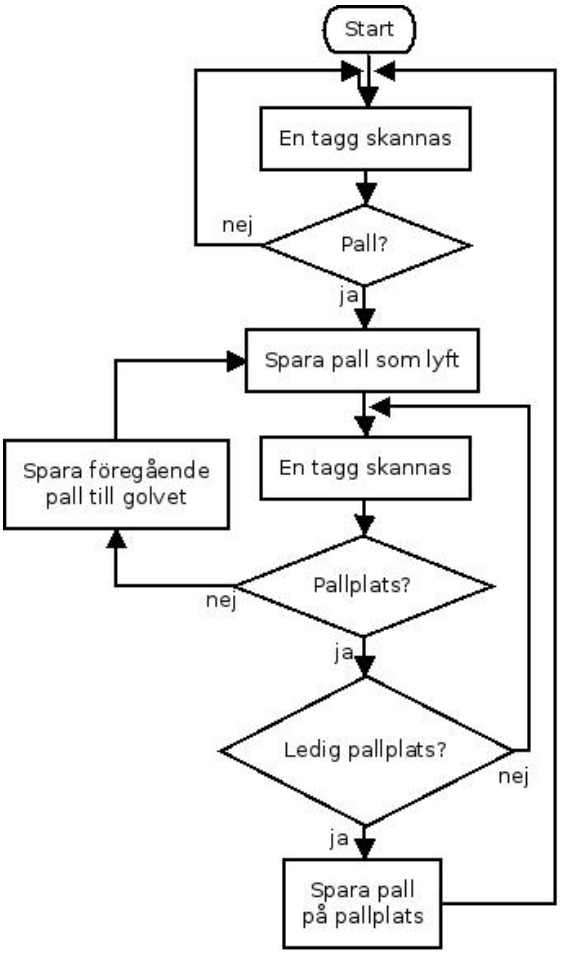
Trigger	**Lyssnar på**	**Händelse**
good_tags_pallet	Pallet	Säkerställer att taggarna inte används. Om det är fallet hindras tilläggning av den pallen.
good_tags_slot	Pallet_Slot	Säkerställer att taggen inte används. Om det är fallet hindras tilläggning av den pallen.
log_article_add	Article	Skriver till Log då en artikel läggs till i databasen.
log_article_remove	Article	Skriver till Log då en artikel tas bort från databasen.
log_article_update	Article	Skriver till Log då en artikel ändras i databasen.
log_move	Moved_Pallet	Skriver till Log då en pall flyttas och/eller sätts på en ny pallplats.
log_pallet_add	Pallet	Skriver till Log då en pall läggs till i databasen.
log_pallet_remove	Pallet	Skriver till Log då en pall tas bort från databasen.
log_slots_add	Pallet_Slot	Skriver till Log då en plats läggs till i databasen.
log_slot_remove	Pallet_Slot	Skriver till Log då en plats tas bort från databasen.
log_slot_update	Pallet_Slot	Skriver till Log då en plats ändras i databasen.
log_setOfA_add	Set_Of_Article	Skriver till Log då ett saldo för en viss artikel på en viss pall läggs till i databasen.
log_setOfA_remove	Set_Of_Article	Skriver till Log då ett saldo för en viss artikel på en viss pall tas bort från databasen.
log_setOfA_update	Set_Of_Article	skriver till Log då ett saldo för en viss artikel på en viss pall ändras i databasen.

4.3 Server

Serverns ansvar är att sköta anrop från klientera när de vill hämta eller skriva data till databasen. Den främsta logiken ligger i hur truckklientens skannade taggar hanteras, se avsnitt 4.4.1 för mer information om hur truckklienterna skickar taggars ID-nummer till servern. I figur 4.6 visas logiken som sker på servern. Oavsett om taggen som skannas sitter på en pall eller pallplats så skickas ID-numret till servern då klienten inte kan skilja dem åt. Om en platstagg skannas först ignoreras det av servern och den går tillbaka till sitt första tillstånd. Medan om en palltagg skannas registreras istället att en pall har lyfts från den plats, som det i databasen står att den befinner sig på. När en pall väl är registrerad som lyft väntar servern på ett ID-nummer som antingen hör till en ledig pallplats eller en pall. Om en ledig pallplats skannas efter att en pall har lyfts upp drar servern slutsatsen att pallen nu står på pallplatsen. Därefter går servern tillbaka till utgångsläget där den väntar på att en ny pall ska lyftas upp av trucken. Skannas däremot en andra pall, istället för en ledig pallplats, drar servern istället slutsatsen att den första pallen har ställts på golvet. Detta för att pallen inte ställts på en giltig lagerplats.

Kommunikation mellan olika komponenter i systemet sköts med hjälp av JSON. Det är ett kompakt, textbaserat format som är lätt att hantera vid hantering och kontroll av information. De anrop som görs med JSON från klienterna hanterar servern. Anropen kan bland annat vara för att skapa pallar och pallplatser, radera taggar, och få en lista på pallar, pallplatser och förflyttningar i systemet.

Ett exempel på ett anrop till API:et och dess JSON-svar kan ses i figur 4.7. Ett GET-anrop görs till `/api/moves` och som svar fås en lista över hur två olika pallar har flyttats. Forklift #1 har lyft en pall från golvet, ställt den på pallplats AB 12 403 och därefter lyft en ny pall från golvet.



Figur 4.6: Flödesschema som beskriver serverlogiken som hanterar skannade RFID-taggar. När en tagg som sitter på en pall skannas registreras pallen som lyft. Antingen ställs den därefter ner på en pallplats om en tagg tillhörande en ledig pallplats skannas eller ställs den på golvet om en ny pall blivit skannad.

```
GET /api/moves HTTP/1.1
Host: www.smartrfid.se:9000

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
[
  {
    "date": "Sat May 23 22:10:00 CET 1970",
    "put_down": "false",
    "reader": "Forklift #1",
    "slot": "floor",
    "tags": ["jFKfjl234a4", "mfaA4jfl611"]
  },
  {
    "date": "Sat May 23 22:20:00 CET 1970",
    "put_down": "true",
    "reader": "Forklift #1",
    "slot": "AB 12 403",
    "tags": ["jFKfjl234a4", "mfaA4jfl611"]
  },
  {
    "date": "Sat May 23 22:30:00 CET 1970",
    "put_down": "false",
    "reader": "Forklift #1",
    "slot": "floor",
    "tags": ["dladfaorej4", "sadfoji23zx"]
  }
]
```

Figur 4.7: Ett exempel på hur en JSON-sträng ser ut. En användare har bett om att få en lista på pallförflyttningar via HTTP-gränssnittet på `/api/moves`. Som svar fås en lista med tre objekt, där varje objekt är en förflyttning. En lista defineras med hakparenteser `[ ]` och objekt med klammerparenteser `{ }`. Ett objekt består i sin tur av attribut och dess värden.

4.4 Klientmjukvara

Två olika klienter har implementerats och för att underlätta utvecklingen delar de kodbas. Det ger fördelen att de respektive klienternas funktioner kan användas genom samma grafiska gränssnitt. Det grafiska gränssnitt är implementerat med Python-biblioteket TkInter [22]. TkInter valdes för att det är välanvänt samt har utförlig dokumentation vilket underlättar utvecklingprocessen. TkInter inkapslar i sin tur kommunikationen med TK som är ett så kallat *GUI widget toolkit* som ritar fönster på skärmen [23]. Klienterna är skrivna i Python för att ge en strukturerad kodbas som är lätt att vidareutveckla vid behov. Modulen som sköter kommunikation mellan RFID-läsaren och klienterna är skriven i C enligt den dokumentation som ingick med läsaren. Gemensamt för klienterna är att de får data, i form av ID-nummer, från skannade taggar från RFID-läsaren. Därefter initieras kommunikation med Play-servern via JSON-gränssnittet.

4.4.1 Truckklienten

Truckklienten, som är den mest avancerade av de två klienterna, avses användas tillsammans med en läsare på en truck. Det klienten gör är att den får in data från RFID-läsaren i form av det skannade ID-numret från en pall- eller platstagg. Dessa ID-nummer läggs i en kö som JSON-objekt, som sedan en separat tråd så snart som möjligt skickar upp till servern via ett HTTP-anrop. Denna design innebär att trucken kan fortsätta att flytta pallar även om truckklienten temporärt tappar anslutning till det lokala nätverket. När ID-numret väl når servern avgör den lagrets nya tillstånd. Att lägga logiken som avgör hur ID-numrena ska hanteras på servern leder till smidigare uppdateringar, enklare design av truckklientens kod samt att mindre data behöver skickas mellan klient och server. Mer om serverlogiken kan läsas i avsnitt 4.3.

4.4.2 Stationsklienten

Stationsklientens grafiska gränssnittet gör det enkelt att utföra rutinuppgifter som att registrera pallar och pallplatser och dess respektive RFID-tagg i systemet. Den är implementerad till stort på samma sätt som truckklienten då den också läser in RFID-taggar och gör anrop till servern. Skillnaden är att stationsklienten tillhandahåller fler formulär då dess verktyg inte går att utföra lika automatiskt. Till exempel måste användaren mata in vilken artikel som ska stå på en pall som skapas.

4.5 Kryptering av data

Kryptering används för att skydda från oönskad tillgång till databasen och administrationsverktyget. Användardata sparas i databasen i tabellen *Users*. Användarnamnet består av en e-postadress som sparas i klartext medan lösenordet krypteras på servern innan det skickas till databasen. För kryptering av lösenord används Java-biblioteket jBCrypt, som är en Java-implementation av Blowfish-algoritmen [24].

4.6 Hårdvara

Hårdvaran som används är en lågfrekvent GP60A (se bilaga B) från PROMAG och runda RFID-taggar med en diameter på 5 cm. Platstaggarna är i kolfiber och palltaggarna i epoxy. Läsaren är en kompaktläsare med en fempolsanslutning som omvandlas till en niopols-COM-anslutning. Läsaren drivs med hjälp av ett 15 V nätaggregat som kopplas till en adapterbox, vilken i sin tur levererar 12 V likström och som dessutom har ett dimmerrelä. Dimmerreläet används för att reglera läsavstånd, då det kan vara önskvärt att minska eller öka det. Läsaren läser endast den tagg som är starkast inom dess räckvidd. Läsaren är dessutom självkalibrerande och meddelar vid varje uppstart, på en fyrgradig skala, hur bra omgivningen är gentemot optimala förhållanden (se bilaga C).

Den ena typen av taggar som valts är passiva, runda och gjorda i kolfiber för att vara mer stryktåliga. Dessa taggar placeras i sin tur i gummidistanser som möjliggör läsning av tagg nära metall. RFID är överlag väldigt känsligt för metall då metall absorberar de elektromagnetiska vågor som annars skulle skapa induktans i RFID-taggen. Taggarna som används har en diameter på 5 cm vilket är tillräckligt för att kunna läsa på 50 cm avstånd med vald läsare, enligt L-G Johansson, VD för RFID Systems Sweden, (2014). Den andra typen är också passiv, men är gjord i epoxy och har bortsett från stöttålighet samma specifikationer som den första. Dessa två typer av taggar valdes för att de uppfyllde de krav som ställts om läsavstånd och hållbarhet i industriell miljö.

Läsaren måste placeras på trucken så att signalen från platstaggen alltid blir starkast vid något tillfälle vid flytt av en pall till lagerplats. Anledningen till att platstaggen måste bli starkast vid ett tillfälle är för att läsaren ska uppfatta vilken pallplats pallen har ställts på. Läsaren kan läsa taggar och skicka deras ID-nummer i form av bitströmmar till den anslutna truckklienten. Klienten kan köras på en enkortsdator som har stöd för Python. Det är möjligt att förse både läsare och klient med ström direkt från trucken och data kan skickas via den WiFi anslutning som enkortsdatorn innehar. På så vis blir det enkelt att skala upp systemet. Det vill säga en ny lagerplats innebär en ny tagg och en ny truck innebär en ny läsare.

Figur 4.8: En möjlig placering av RFID-läsaren på en gaffeltruck.

5

Slutprodukt

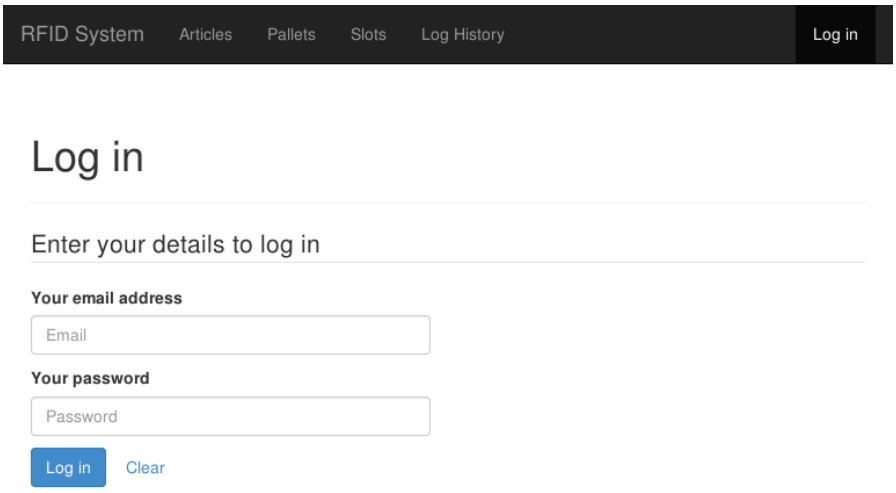
Slutprodukten är ett resultat av den systemdesign som beskrevs i kapitel 3 och implementation i kapitel 4. Kapitlets avsnitt demonstrerar administrationsverktyget och de olika klienterna, samt ger en kort resumé av de tester av systemet som gjorts. Kapitlet visar upp slutprodukten och dess slutgiltiga funktionalitet ur en användares perspektiv.

5.1 Administrationsverktyget

Admininstrationsverktyget har fyra primära funktioner i form av artikelvy, platsvy, pallvy, och loggvy. Dessa vyer ger en överblick över de entiteter som finns i databasen samt möjliggör administration av dem.

5.1.1 Inloggning

Varje del av verktyget har en auktoriseringskontroll som verifierar om användaren ska ha tillgång till funktionen, om inte behöver användaren logga in med giltiga användaruppgifter. Hur inloggningsformuläret ser ut kan ses i figur 5.1. Eftersom detta inte är något publikt verktyg finns det ingen öppen funktionalitet för att skapa nya konton, utan dessa skapas av administratören. När användaren har angett giltiga användaruppgifter och tryckt på *Log in* presenteras välkomstmeddelandet i figur 5.2 och därefter kan användaren fortsätta att utnyttja verktygets funktioner.

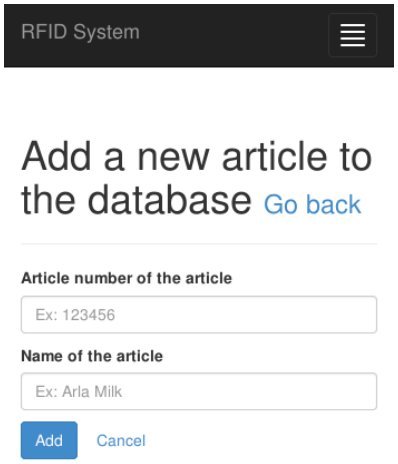


Figur 5.1: Användaren hamnar i denna vy när den inte är inloggad. Med användarnamn och lösenord får användaren tillgång till resterande vyer.

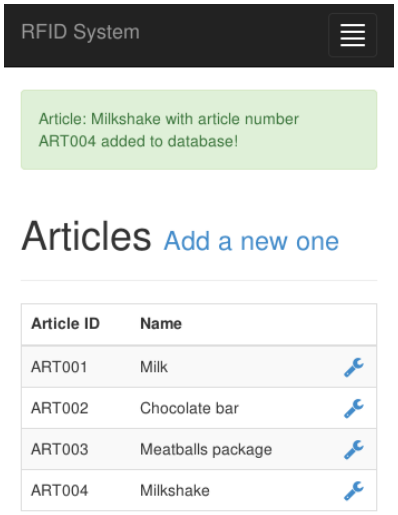
Figur 5.2: Vyn som visas när användaren har loggat in i systemet.

5.1.2 Artiklar

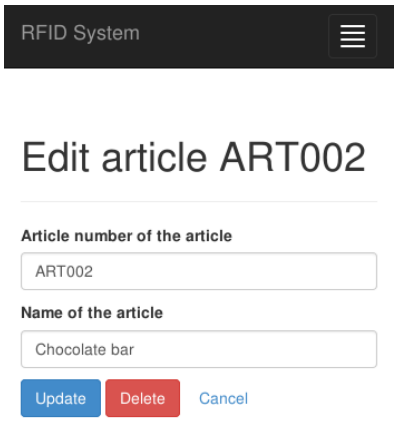
Genom att klicka på *Articles* i navigationsmenyn tas användaren till listan med artiklar i figur 5.4. Klickar användaren på *Add a new one* tas hon till formuläret för att lägga till en ny artikel, se figur 5.3. När användaren har fyllt i uppgifter och trycker på *Add* skickas hon tillbaka till artikelvyn i figur 5.4. När artiklar har skapats är det möjligt att redigera och uppdatera dem genom att trycka på skiftnyckeln till höger. Redigeringsformuläret i figur 5.5 påminner om formuläret för att lägga in nya artiklar, men det finns även möjlighet att radera artikeln såvida en pall inte innehåller artikeln, i sådana fall dyker ett felmeddelande upp. Om en ändring utförs och användaren sen klickar på *Update* skickas hon tillbaka till listvyn och meddelandet, i figur 5.6, om att en artikel uppdaterats visas.



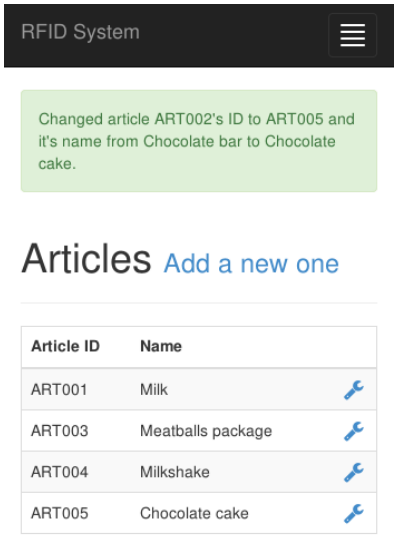
Figur 5.3: Formuläret för att skapa en ny artikel med ett artikelnummer och -namn.



Figur 5.4: Listan med artiklar inklusive ett meddelande som säger att en artikel lagts till. Efter att en artikel har skapats visas denna listan samt meddelandet i grönt.



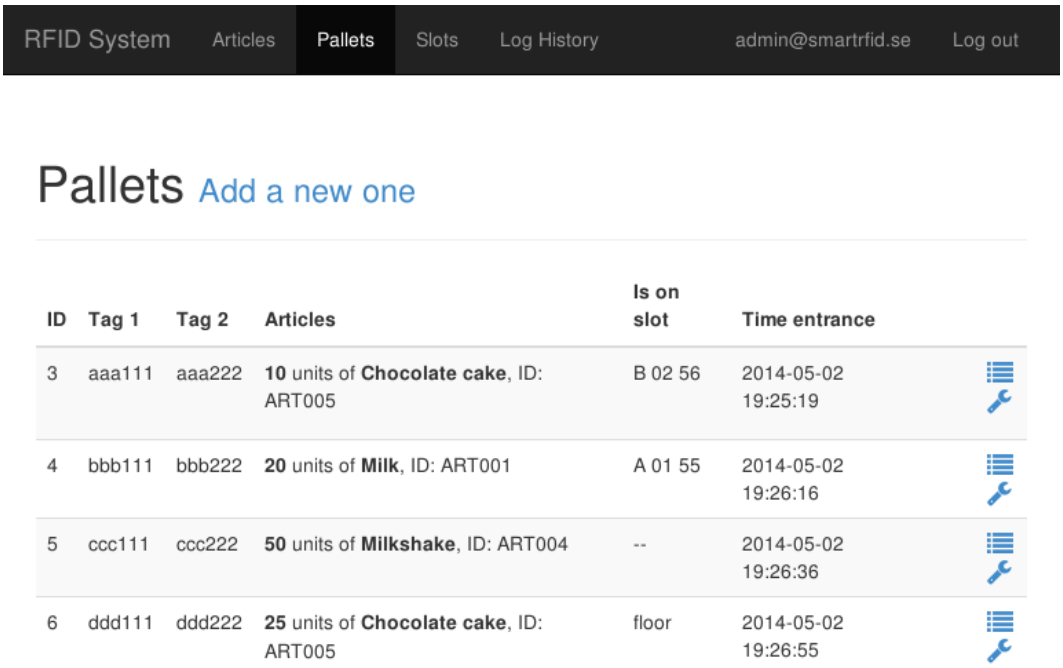
Figur 5.5: Formuläret för att redigera en artikel. Genom att trycka på skiftnyckeln i figur 5.4 visas detta formuläret. Här är det möjligt att redigera en artikel.



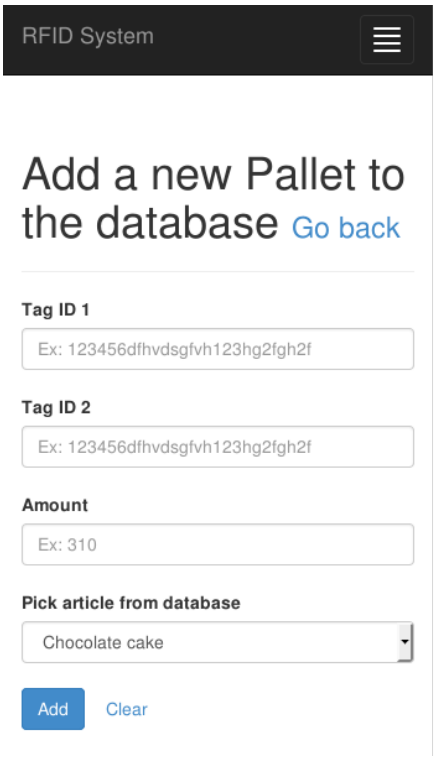
Figur 5.6: Listan med artiklar och ett meddelande om att en artikel har uppdaterats. Efter att en artikeln har blivit uppdaterad via formuläret i 5.5 presenteras återigen denna listvy.

5.1.3 Pallar

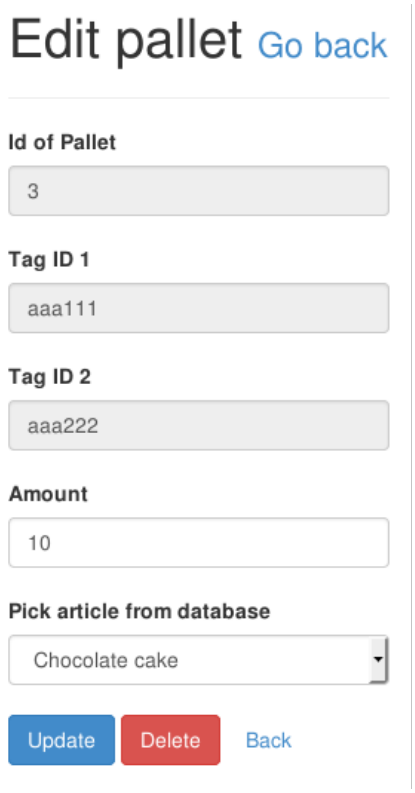
Gränssnittet för att redigera pallar är likt gränssnittet för artiklar. I figur 5.7 kan översiktslistan av pallarna i systemet ses. Där syns pallens två taggar, antalet artiklar som står på pallen, vilken plats pallen står på och tiden när pallen kom in på lagret. Genom att klicka på *Add a new one* tas användaren till formuläret i figur 5.8 där det är möjligt att lägga in en ny pall i systemet. Från listvyn är det också möjligt att redigera befintliga pallar genom att trycka på skiftnyckeln, vilket presenteras i formuläret i figur 5.9. Där går det att ändra vilken artikel och antal som står på pallen. Om användaren i listvyn klickar på listknappen på raden för pallen med mjölk visas loggvyn i figur 5.10 för just den specifika pallen. I detta fall är det möjligt att se vilka taggar och vilken artikel som pallen skapades med och att den har lyfts från golvet och ställts på pallplats `A 01 55`.



Figur 5.7: En lista med de pallar som existerar i systemet.



Figur 5.8: Formuläret för att lägga till en ny pall.

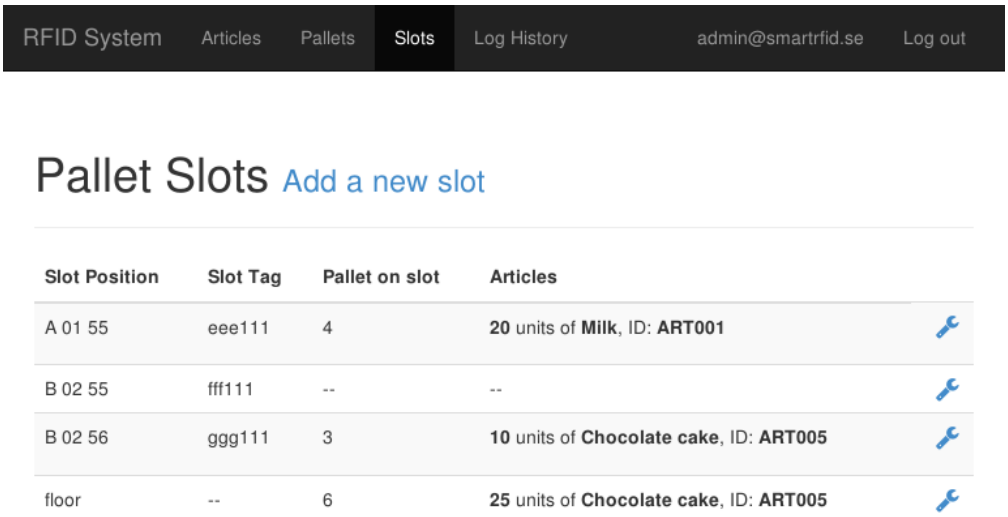


Figur 5.9: Formuläret för redigering av en pall.

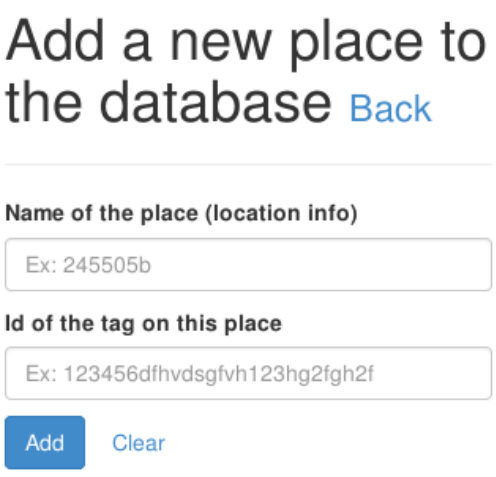
Figur 5.10: Listan med logghändelser för en specifik pall.

5.1.4 Pallplatser

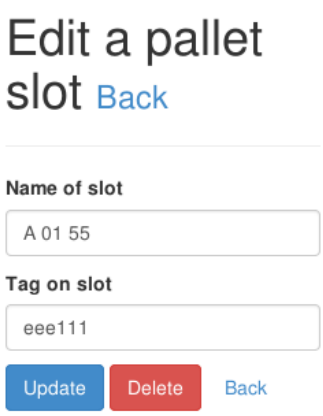
För att få en översikt över pallplatserna kan användaren klicka på *Slots* längst upp på navigationsmenyn, då vyn i figur 5.11 visas. Där platsens namn och tagg ID kan hittas tillsammans med information om pallen som står på den platsen. En ny pallplats kan skapas genom att trycka på *Add a new slot* och då kommer formuläret i figur 5.12 visas. Väljer användaren istället att från listvyn redigera en pallplats visas formuläret i figur 5.13.



Figur 5.11: Vyn med de pallplatser som existerar i systemet.



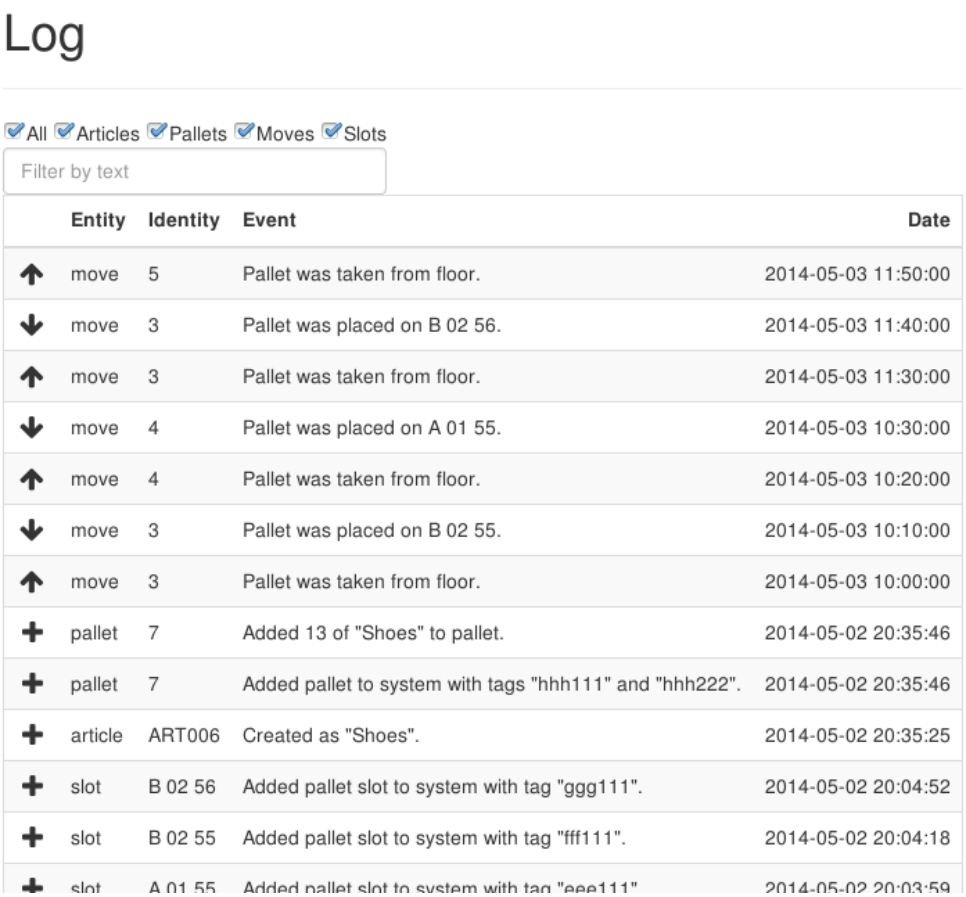
Figur 5.12: Formulär för att skapa en ny pallplats.



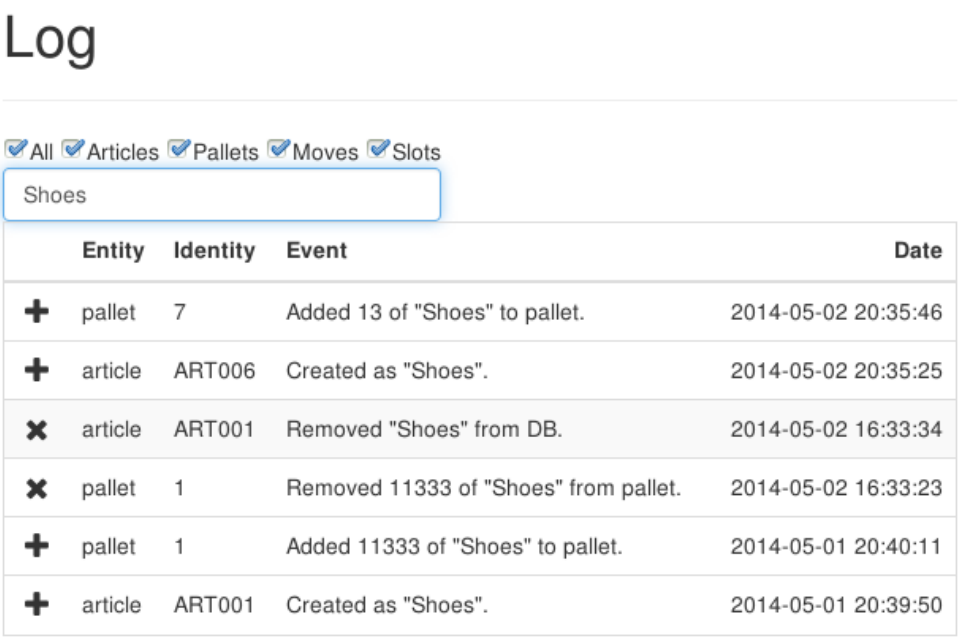
Figur 5.13: Formulär för att redigera en pallplats som redan finns i systemet.

5.1.5 Logghistorik

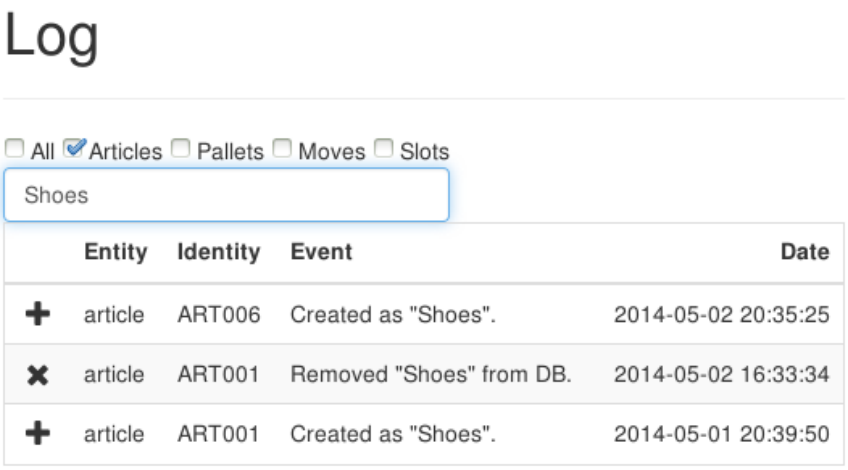
Genom att klicka på *Log History* i navigationsmenyn förflyttas användaren till historikvyn som syns i figur 5.14. Här visas en lista på flera olika typer av händelser; pallplatser, artiklar och pallar som skapats samt pallar som flyttats mellan golv och pallplatser. Om användaren matar in text i textfältet filtreras listan i realtid med hjälp av Javascript. Ett exempel på det kan ses i figur 5.15 där listan filtreras på ordet *Shoes*. Då visas bara händelser som förknippas med ordet *Shoes*. För att ytterligare filtrera historiklistan kan användaren, ovanför textfältet, välja mellan olika typer av händelser. Detta visas i figur 5.16 där enbart *Articles* är valt och detta resulterar i att användaren får se att artikeln *Shoes* skapades ena dagen, togs bort dagen därpå och återskapades igen fyra timmar senare men med ett nytt artikelnummer. Det går även att sortera efter kolumner om så önskas.



Figur 5.14: Vyn som visar alla händelser som skett i systemet. Bland annat är det artiklar som skapats, pallars som skapats och pallars som flyttats.



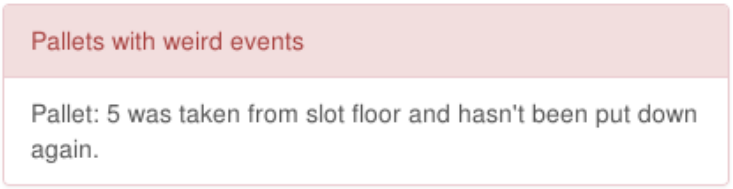
Figur 5.15: Vyn som visar alla händelser som skett i systemet filtrerat på ett ord. Användaren har fyllt i ett ord i textfältet, vilket får vyn att bara visa de händelser som har det ordet i sig.



Figur 5.16: Vyn som visar alla händelser som skett i systemet filtrerat på ett ord och kategori. Användaren har filtrerat listan ytterligare och har valt att bara visa händelser relaterat till artiklar med ordet *Shoes*.

5.1.6 Notiser

Om något oväntat inträffar i systemet som eventuellt kräver uppmärksamhet, så visas detta på förstasidan då en användare loggat in i systemet. Ett exempel på ett sådant meddelande kan ses i figur 5.17. Meddelandet säger att pallen med ID 5 har lyfts upp men ej satts ner igen, vilket kan indikera att den inte står på en giltig pallplats.



Figur 5.17: Notifikation om en händelse som behöver hanteras. Den som är inloggad får en notifikation om att pall med ID-nummer 5 har lyfts upp men ej satts ner på en plats.

I nuläget notifieras användare enbart om en pall lyfts från en plats och sedan aldrig satts ner på en annan giltig lagerplats.

5.2 Stations- och truckklienten

Funktionaliteten som hör ihop med stations- och truckklienten går att använda genom ett och samma program som kan ses i figur 5.18, då det är enklare att demonstrera. Dock är bara en av funktionerna relevant i en truckklient och tre av dem i en stationsklient.

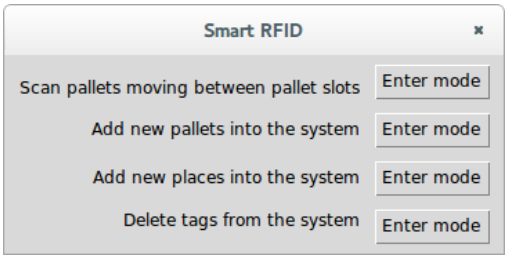
När mjukvaran implementeras på en truck kommer den automatiskt att gå in i *Scan pallets moving between pallet slots* och visa vyn enligt figur 5.19. Den vyn väntar sedan på att taggar ska skannas av RFID-läsaren och så fort som möjligt skickas de till servern i en separat tråd. I sin tur uppdaterar servern statusen på lagret. Det som visas i skärmdumpen är hur mjukvaran skickar två lästa tagg-ID:n till servern.

En funktion som hör till stationsklienten är möjligheten att sammankoppla nya pallar med RFID-taggar i systemet. Detta läge kan nås genom att klicka på *Add new pallets to system* via huvudmenyn figur 5.18, vilket för användaren till vyn i figur 5.20. När en pall kommer till lagret förses den med två taggar, varefter de skannas. Därefter kan vilken artikel som är på pallen väljas ur en lista, samt dess kvantitet kan sättas. Slutligen skickas ett JSON-objekt till servern och pallen blir registrerad i systemet med dess två taggar, artikelnamn och -nummer.

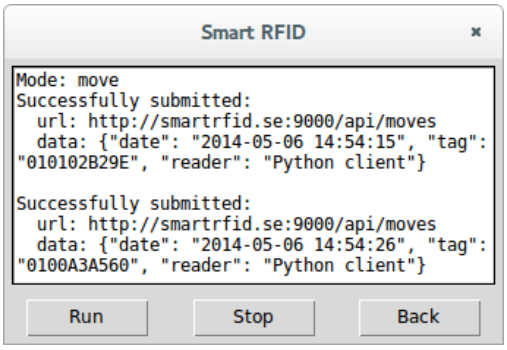
Användaren av stationsklienten kan också lägga till nya pallplatser i systemet genom att klicka på *Add new places to system*. Då visas vyn i figur 5.21. Där kan användaren skanna en RFID-tagg och fylla i det unika platsnamnet. När användaren trycker på *Send*

sparas platsen i systemet.

I stationsklienten finns möjligheten att radera en pall eller pallplats från systemet. Detta görs med hjälp av vyn som nås om användaren klickar på *Delete tags from the system* i huvudmenyn, vilket visar vyn i figur 5.22. När en tagg skannas raderas antingen pallen eller pallplatsen från systemet som är associerad med den, inklusive de förflyttningar som refererar till objektet. Detta är exempelvis användbart när en pall lämnar lagret, och dessa taggar kan med fördel skannas med stationsklienten för att de ska tas bort från systemet. När en tagg väl har blivit borttagen kan den återanvändas tillsammans med en ny pall eller plats.

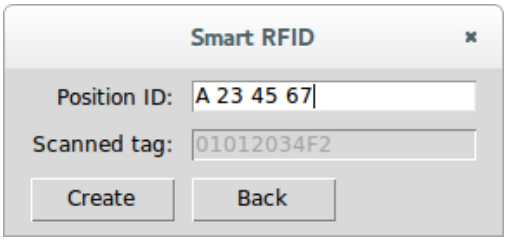


Figur 5.18: Menyn som användaren möts av när klienten startats. För att göra det enklare att demonstrera truck- och stationsklienterna finns bådas funktionalitet i samma program.

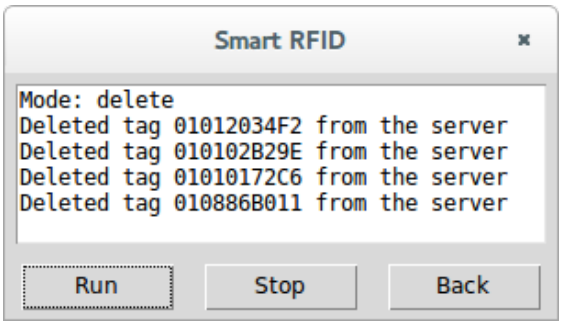


Figur 5.19: Truckklientsläget där klienten skickar till servern de RFID-taggar som trucken kommer i kontakt med. Därefter körs logiken på servern som uppdaterar statusen för pallarna och pallplatserna i databasen. Hur logiken är implementerad kan läsas under avsnitt 4.4 i kapitel 4.

Figur 5.20: Formulär för att skapa en ny pall. Först väljs ett av artikelnumrena som finns i systemet. Därefter fylls antalet enheter av artikeln in i fälten nedanför. Slutligen skannas två RFID-taggar och när användaren trycker på *Create* skapas pallen på servern.



Figur 5.21: Läget för att skapa en ny pallplats i systemet. För att skapa en pallplats krävs att ett ID för platsen fylls i och att dess tagg skannas. Därefter när användaren trycker på *Create* kommer platsen skapas på servern.



Figur 5.22: Läget för att radera RFID-taggar och sammankopplade entiteter från servern. Skannas exempelvis en tagg på en pall kommer både taggen och pallen raderas från databasen. Detta gör det möjligt att enkelt frigöra taggar för återanvändning.

5.3 Stresstestning

Dan Cargo har, genom platschef D. Leifsson (2014), uppskattat att lagret i Arendal kommer ha cirka 3-5 truckklienter och 1-2 stationsklienter aktiva samtidigt.

En simpel stresstestning utfördes därför genom att koppla upp tolv enheter samtidigt och sedan skicka förfrågningar till servern. Detta visade inte några tendenser till fördröjningar eller undermåligt beteende. Databasen i sig har stöd för många fler anslutningar än så, och kommer inte heller den utgöra ett problem.

6

Diskussion

De delar av projektet som hittills beskrivits har behandlat implementering och olika val som gjorts. Utöver dessa finns exempelvis en rad olika implementationer som gjorts i testsyfte och som valts bort av olika anledningar, samt tankar som behandlat projektet i stort. I detta kapitel sker en reflektion dels över vad som gjorts och dels vad som skulle kunna gjorts annorlunda.

6.1 Förbättringspotential

Tankegången kring hård- och mjukvara har sett väldigt annorlunda ut genom projektets gång. Här belyses alternativa lösningar som har beaktats och de förbättringar som existerar med den befintliga lösningen.

6.1.1 Hårdvara

Under projektets gång har ytterligare egenskaper hos RFID-läsaren identifierats, som hade kunnat vara önskvärda med syfte tt förbättra driftsäkerhet och förenkla implementation. En fördel hade varit om läsaren kunde läsa flera taggar samtidigt. Detta skulle innebära att hänsyn inte måste tas till att pall- och platstagg behöver vara närmast läsaren vid något tillfälle var, vid flytt av en pall. Vilket i sin tur leder till att RFID-läsarens placering på trucken skulle få mindre betydelse för driftsäkerheten.

En annan önskvärd specifikation på läsaren hade varit ifall en eller flera externa antenner skulle användas. Själva läsaren med kretskort och anslutningar skulle då kunna sitta var som helst på en truck och inte riskera att skadas av pallar eller lagerhyllor. För att läsaren ska kunna fungera på den typ av höglager som Dan Cargo använder är det till och med en nödvändighet att använda två antenner för att systemet skulle fungera. Detta på grund av den typ av truck som används, som kan ses i figur 6.1, där uppställning av pallar sker på tvären på grund av den smala gången mellan lagerhyllorna.

Aktiva taggar är något som projektgruppen funderat på att använda som platstaggar. Detta då dessa taggar sällan kommer att bytas och på grund av att den läsare som används i projektet endast läser en tagg. Detta har inte testats då detta ansågs ligga utanför prototypens utsträckning. Men i en fullskalig installation kan det vara ett alternativ att använda aktiva taggar tillsammans med en enklare läsare.

Figur 6.1: Höglagertruck som används på Dan Cargo. Notera att gafflarna förs ut åt sidorna istället för att vara statiskt placerade riktade framåt.

Valet att använda ett lågfrekvent RFID-system baserades på dess penetrationsförmåga i metallrika miljöer. Det hade varit intressant att se hur andra frekvenser presterar lagermiljöer eftersom de i övrigt har bättre egenskaper och billigare taggar än lågfrekventa system. Att genomföra ett sådant test ansågs däremot ligga utanför projektet syfte.

Det har även funnits designproblem på hårdvarusystemet som projektrgruppen löst under projektets gång. Grundtanken var att sätta en läsare på varje pallplats och en tagg på varje pall. Detta skulle resultera i att truckarna kan flytta runt pallar fritt mellan giltiga pallplatser utan att trucken behöver ha någon extra teknik. Att ha en läsare på varje plats skulle dock medföra flertalet andra problem. Under den inledande delen av projektet spenderades en del tid på att hitta läsare som skulle kunna köras på *Power over Ethernet*, det vill säga där en kabel kan förse enheten med både datakommunikation och strömförsörjning.

En annan potentiell lösning som diskuterades var att använda läsare som enbart behövde strömförsörjning för att sedan kommunicera direkt via WiFi. Efter ett tag insågs dock att dessa lösningar ej var varken ekonomiskt eller praktiskt genomförbara. För det första hade det inneburit att det hade behövts en switch i varje ände på varje balk och skarv i varje ställage, ställningar med pallplatser, på hela lagret. Detta för att det lätt ska gå att byta ut en balk om den går sönder eller om avståndet mellan balkar av en eller annan anledning skulle behöva ökas. Om man dessutom i detta scenario skulle använda sig av WiFi-läsare kom projektgruppen, efter samtal med Ali Salehson (Institutionen för Data- och Informationsteknik, Chalmers tekniska högskola), fram till att det skulle behövas väldigt många accesspunkter för att sammankoppla alla läsare och detta i en miljö med väldigt mycket störningar i form av metallställnigar. Något som i sin tur hade talat emot en lösning med en RFID-läsare på varje pallplats.

6.1.2 Mjukvaran

Då klienten skulle vara webbaserad samt agera som API-server bestämdes det att den skulle byggas på på ett webbapplikationsramverk för att underlätta utvecklingen. Flera varianter av ramverk togs i beaktning och de främsta tre var Play Framework skrivet i Java, Django skrivet i Python [25] samt Symfony skrivet i PHP [26]. Dessa tre ramverk bygger på samma funktionalitet i form av användning av MVC-konceptet. Första fungerande utkastet på administrationsgränssnittet byggdes med Django. Django fungerade bra men kändes väldigt komplext, särskilt när det var en minoritet av gruppen som var insatta i Python. Det andra alternativet som tänktes på var Symfony men även det valdes bort av liknande orsak; få var insatta i PHP som är primärspråket i det ramverket. Slutligen valdes Play Framework som bygger på Java, ett språk som majoriteten i projektgruppen är insatta i. Mer om ramverket kan läsas i avsnitt 4.1. Andra fördelar som gavs var att det var väldigt enkelt att hantera både testning av kod och databas i samband med ramverket.

Inledningsvis fanns en förhoppning om att klientmjukvara skulle kunna köras på en truckdator, vilket ett program skrivet i programmeringsspråket C med stor sannolikhet skulle ge möjlighet till. Därför skrevs den första versionen av klientmjukvaran i C. Då syftet under projektet gick från att implementera mjukvaran på truckdatorer till att istället köra mjukvaran på enkortsdatorer, skrevs klienten om i Python. Att skriva koden i Python istället för i C snabbade upp och förenklade utvecklingen då Python bland annat är objektorienterat, har en *garbage collector* och har fler funktioner i sitt standardbibliotek. Den kod som sköter kommunikation mellan läsare och klient är dock skriven i C.

6.2 Säkerhet

Enligt projektets implementation är det endast genom Play Framework som interaktion sker med databasen. Genom att servern agerar som en koppling mellan databasen och externa klienter skyddas den data som ligger där från misstag i utveckling av klienter samt försök till sabotage.

För ett ytterligare skydd av otillåten access till databasen genom Play Framework har auktorisering av användare implementerats. Utan ett korrekt konto förhindras tillgång till administrationsverktyget liksom databasen.

Inom projektet fanns önskemål om att kryptera kommunikationen mellan klienterna och servern. Framförallt för att hindra att en person skulle ha möjlighet att avlyssna trafiken och därefter skicka egna anrop till servern. En attack som skulle kunna resultera i att lagerstatusen inte längre stämmer överens med verkligheten. Detta är något som legat utanför prototypens omfång.

Dock kommer det största säkerhetshotet internt, från användarna av systemet. En illasinnad användare är inget som går att skydda systemet ifrån. En användare kan lägga

in felaktiga uppgifter i systemet eller radera data som bör vara kvar. Det är administratörens uppgift att inte ge en illasinnad användare tillgång till systemet. Projektgruppen tror ändå att lösningen som har tagits fram undviker att ofrivilliga misstag sker. Genom att datan som läggs in kontrolleras om den stämmer överens med den modell av lagret som finns i databas och server. Samt att dubbelkontroller utförs vid radering av data via administrationsgränssnittet. Problemet med illasinnade användare existerar även om streckkoder används, då användaren med flit skulle kunna skanna fel pall eller pallplats, alternativt förstöra streckkoderna.

6.3 Produktkravsanalys

De krav som ställts på projektet vid dess uppstart har under projektets gång förändrats för att anpassas efter nya förutsättningar. De gamla kraven benämns här då det fortfarande är intressant att se projektets gång. Nedan följer kraven som finns i bilaga A.
Skallkrav:

- En pall ska kunna märkas med en RFID-tagg som lagrar data om vilken artikel, samt kvantiteten av denna artikel, som finns på pallen. Denna data ska på ett enkelt sätt kunna skrivas till taggen.

Detta stämmer inte längre då det som ansågs vara den smidigaste lösning är att lagra all data centralt. Denna information kopplas sedan till ett förprogrammerat ID-nummer på RFID-taggen [27].

- Med hjälp av en RFID-läsare ska det gå att läsa RFID-taggarna och identifiera den data som finns på dem.

Detta stämmer, men som nämndes under tidigare punkt lagras nu ingen data på taggarna mer än det förprogrammerade ID-numret. ID-numret är sedan kopplat till en pall i databasen. Därifrån måste en klient istället hämta all information om pallen.

- Om en pall märkt med en RFID-tagg sätts på en "giltig" lagerplats ska datan på pallens tagg, tillsammans med platsinformationen automatiskt lagras i en databas.

- Om en pall med en RFID-tagg flyttas från en lagerplats till en annan ska denna ändring ske automatiskt även i databasen.

Dessa krav uppfylls helt och hållet enligt de specifikationer som angivits.

- Lagersystemet på Dan Cargo använder i nuläget $(2014 - 01 - 27)$ platser med benämningar som exempelvis 19 05 35A för att identifiera en unik lagerplats. Dessa benämningar, samt nya benämningar på lagerplatser i framtiden, ska enkelt gå att lägga in i databasen.

Detta uppfylls eftersom lagerplatser kan både läggas till i systemet genom administrationsgränssnittet och stationsklienten.

- Dan Cargos egna lagerhållningssystem ska kunna hämta data från produktens databas och på så vis uppdatera sig självt. Detta innebär att databasen behöver ha ett API (Application Programming Interface) för att andra program både kan hämta och skriva data till den.

Systemet har byggts med tanken att det ska kunna användas både som en fristående produkt och som ett externt API. Som ett API kan en implementation till externa system skapas.

Dan Cargo vill se en funktionell prototyp då projektet är slutfört innan de börjar inleda diskussioner med Consafe Logistics, som är leverantören av deras logistiksystem Astro WMS [28]. Det har därför inte inletts någon implementation hos Dan Cargo.

Nedan följer börkraven:

- Det ska gå att spåra var en pall från första gången den ställdes på en "giltig" lagerplats till dess aktuella position.

Det går att spåra hela processen från tilläggning till borttagning. Detta innefattar ändring av artiklar, pallar, pallplatser samt hur pallar flyttats runt på lagret med hjälp av loggvyn i administrationsverktyget.

- Det ska gå att spara ytterligare data på taggarna, på varje pall såsom exempelvis utgångsdatum.

- Det ska gå att i framtiden spara ytterligare data i databasen tillhörande taggarna. Exempelvis om man i senare skede vill ha ett anteckningsfält associerat med varje pall, ska det vara genomförbart.

I den nuvarande lösningen går det inte att spara någon data på taggarna, däremot går det att lagra data i samband med en pall i databasen.

6.4 Ekonomisk analys

En viktig aspekt av projektet är att produkten som tas fram inte bara effektiviserar produktionen utan att den totala summan som sparas in vid tillämpning, på sikt, överstiger kostnaden för systemet. Tillsammans med platschef D. Leifsson (2014) gjordes en preliminär kalkyl över vad en fullskalig installation av en RFID-lösning skulle kosta på deras lager vid Godsgatan, Göteborg. Detta vid användande av tekniska lösningar ur detta projekt, med undantag från att den hårdvara som används i nuläget mer fungerar som en prototyp och skulle vid full implementation bytas ut.

Dan Cargo har i dagsläget 20 000 lagerplatser, i snitt 300 pallar som går in/ut varje dag och 8 truckar som kan lyfta upp eller ned pallar från pallställen. Samt en in-bana i form av ett rullband där alla pallar passerar.

Vid undersökning av hårdvara fanns önskemål om att inkludera extrafunktionalitet såsom externa antenner och stöd för läsning av flera taggar samtidigt. Med dessa förutsättningar skulle priset öka till cirka det dubbla för en läsare. Detta är funktionalitet som inte var motiverbar för införskaffning till prototypen men som räknas med i den ekonomiska analysen för att få ett mer realistiskt resultat.

Som det ser ut nu med den läsare som används, behövs det dessutom någon form av dator till varje läsare. Det är inte helt säkert att det kommer behövas om det köps in en bättre läsare, då vissa redan har inbyggda mikroprocessorer och WiFi. Oavsett är det en förhållandevis liten summa i sammanhanget. Om en enkortsdator skulle användas, till exempel en Raspberry Pi, resulterar det i en extrakostnad på cirka 200 SEK per läsare [29]. För att göra den ekonomiska beräkningen nedan mer realistisk har detta tagits i beaktning.

Inköpen av taggar är, i stort sett, den enda rörliga kostnaden. Det finns indikationer som tyder på att lågfrekventa taggar kostar 1 SEK/styck vid massbeställning. Till de 20 000 lagerplatserna kommer det dessutom tillkomma cirka 4 SEK för gummiadaptern som behövs för att fästa taggar på metallbalkar. Man kan dessutom räkna med att endast cirka 75 % av pallplatserna används och dessutom behövs 2 taggar till varje pall.

Den totala installationskostnaden för systemet som sådant skulle då bli:

$$\begin{aligned}\text{Kostnad} = (\text{antalet truckar} \cdot (\text{kostnad för en läsare} + \text{enkortsdator})) + \\ (\text{antalet pallplatser} \cdot (\text{kostnad för en tagg} + \text{en gummidistans})) + \\ (\text{Antalet pallar som finns i lager just nu} \cdot \text{Kostnad för en tagg} \cdot 2) = \\ (8+1) \cdot 6\,200 + 20\,000 \cdot 5 + 20\,000 \cdot 0.75 \cdot 1 \cdot 2 = \\ 185\,800 \text{ SEK}\end{aligned}$$

Där den extra läsaren är till för *in-banan*.

Den enda löpande kostnad som man sedan har, är inköp av nya taggar till pallar och nya pallplatser. Men eftersom att taggarna går att återanvända och det ungefär går lika många pallar in och ut per dag, räcker det med att ha ett mindre reservlager, om en tagg skulle behöva bytas.

Dan Cargo anser sig i dagsläget ha en välfungerande arbetsgång efter många år i branchen. De håller däremot med om att projektets lösning hade kunnat effektivisera deras arbete med ökad lönsamhet i längden.

6.5 Etiska aspekter

Etiska aspekter är något som präglar de flesta projekt inom ny teknik och nya tekniska lösningar. Det är därför värt att spegla lite av den etiska problematik som projektgruppen tagit ställning till.

6.5.1 Automatiserad produktion

Ur flera perspektiv finns fördelar med en mer automatiserad produktion, som i exempelvis Dan Cargos fall skulle kunna innebära besparingar på lång sikt. Om det däremot är sett ur ett strikt etiskt perspektiv finns det en del frågor som skulle kunna ställas. Till exempel innebär en automatiserad produktion att en maskin eller dator tar över det jobb som tidigare gjorts utav människor och därigenom drar ner deras arbetstimmar eller gör dem överflödiga. I fallet med Dan Cargo har projektgruppen, efter diskussioner med platschef D. Leifsson (2014), kommit fram till att produkten skulle på sikt kunna spara in minst en heltidstjänst. Alltså har lösningen, i värsta fall, gjort att en medmänniska blivit arbetslös.

Det går istället se det som att antalet moment som gemene anställd måste gå igenom minskas, och därmed antalet ställen som fel kan begås på, vilket i någon utsträckning kan stärka den persons självkänsla då färre fel begås. Mer konkret blir arbetsplatsen säkrare genom att en automatiserad lösning bidrar till minskat spring och minskade repetitiva arbetsuppgifter. Detta kan dessutom innebära att de resurser som tidigare lades på detta nu kan omfördelas till andra projekt inom företaget. Det vill säga idéer som man inte kunnat genomföra tidigare, exempelvis av kostnadsskäl, kan nu gå att realisera med de resurser som precis frigjorts och därmed bidra till nya arbetstillfällen.

De aspekter som belyses ovan är helt och hållet spekulativa. Trots att det inte finns några belägg för att det som är beskrivet ovan skulle vara fallet så har gruppen tagit det i beaktning.

6.5.2 Ansvar och skyldigheter mot kunder/konsumenter

Det kan verka drastiskt att tänka i de banor som beskrivits inledningsvis. Om det ses till vad logistikföretagens syfte är gentemot samhället bidrar en mer automatiserad produktion, i enlighet med den lösning som beskrivits i detta projekt, till mycket mer positivt för alla konsumenter. En aspekt som kunder och/eller konsumenter eftertraktar är snabba och säkra leveranser, samt fulla butikshyllor. Utöver det gäller att paket eller brev inte ska slarvas bort eller rentav försvinna på vägen till deras mottagare.

6.5.3 Miljöaspekter

Miljön är ett högaktuellt diskussionsämne i dagens samhälle och något som präglar alla företag, produkter och system. Med utgångspunkt i Dan Cargos nuvarande logistiksystem, finns det inget som tyder på att vår lösning skulle innebära ytterligare miljöpåverkan. Den hårdvara som nu används i form av RFID-läsare kräver endast marginellt mer energi i med streckkodsläsare. Det skulle däremot totalt sett snarare kunna bli så att miljöpåverkan minskar då antalet arbetsmoment minskar i form av exempelvis färre inventeringar och då mindre truckkörning osv. Det är även ett tänkbart scenario att antalet transporter till och från lagret blir färre till följd av att lagersaldot blir mer pålitligt.

6.5.4 Personlig integritet

Det är enkelt att dra paralleller till övervakning när man talar om RFID-teknik, inom det användningsområde som vi beskrivit i denna rapport, då alla händelser i systemet loggas i realtid. Det skulle alltså, rent teoretisk, gå att se exakt vad en viss anställd gjorde vid en viss tid punkt, hur mycket han gjorde, och vilka eventuella fel han gjorde. Detta kan kännas som en inskränkning av personlig integritet, men det går även att använda till sin fördel exempelvis vid lönesamtal. Faktum är att det är exakt såhär det går till idag. På Dan Cargo så har alla anställda en egen inloggning till streckkodsläsarna och deras aktivitet loggas kontinuerligt. Något som D. Leifsson visar oss (2014). Det blir således ingen skillnad vid användning av projektets lösning kontra hur det fungerar på lager idag.

6.6 Vidareutveckling

Forskning kring RFID har funnits sedan länge och marknaden för RFID-lösningar inom logistikbranchen är väldigt stor. Utifrån den feedback som givits av Dan Cargo dras slutsatsen att även om det finns återförsäljare med liknande lösningar på marknaden idag så är det inget som är vida spritt.

Möjligheterna och förutsättningarna för produkten att vidareutvecklas och nå en fullskalig implementation är goda och en framtida lösning som kan vara redo att tas i bruk är ett realistiskt mål.

Som ett av målen uttryckte ville projektgruppen gärna visa på att det slutgiltiga systemet faktiskt gick att integrera med andra system och därmed kunna fungera som en insticksmodul till nuvarande storskaliga logistiksystem. Möjligheterna diskuterades med tjänstemän på Dan Cargo och det visade sig vara en aning problematiskt att tillåtas göra ändringar i deras nuvarande system.

Då databasen är en vanlig MySQL-relationsdatabas kan alla system med stöd för SQL enkelt använda sig av den genom att skicka SQL queries. Det skulle dessutom kunna göras på databasnivå, förutsatt att det befintliga logistiksystemet redan använder sig av en SQL-databas. Företrädelsevis genom att en trigger ligger och väntar på att tabeller ska uppdateras för att då skriva datan vidare till ytterligare en databas.

7

Slutsats

En välfungerande prototyp utav ett lagerhållningssystem har implementerats. Prototypen består av ett antal RFID-taggar, RFID-läsare, truckklient, stationsklient, server, databas och ett webbaserat administrationsverktyg. Prototypen visar på att automatiserad lagerhållning inte bara fungerar i teorin utan även i praktiken, om än i liten skala.

Projektets lösning visar på en effektivisering av arbetet på ett lager. Antalet manuella moment minskas vilket leder till färre moment där fel kan begås. Projektgruppen anser att lösningen går att använda även för en person utan större teknisk kunskap inom IT, då gränssnitten anses vara självförklarande. I sin helhet är projektgruppen nöjda med det resultat som uppnåtts, även om det finns rum för förbättringar inom flertalet områden.

Litteratur

[1] P. J. Sweeney II, *RFID For Dummies*, Second. Hoboken: Wiley Publishing, Inc, 2005.

[2] *Vilant*, 2014. [Online]. Tillgänglig: http://www.vilant.com (Hämtad: 2014-05-15).

[3] J. Thörnqvist, "SweClockers besöker Kompletts huvudlager och PC-produktion", *SweClockers*, 2014. [Online]. Tillgänglig: http://www.sweclockers.com/artikel/18330-sweclockers-besoker-kompletts-huvudlager-och-pc-produktion (Hämtad: 2014-05-15).

[4] *Git*, 2014. [Online]. Tillgänglig: http://git-scm.com/ (Hämtad: 2014-06-03).

[5] "Free source code hosting for Git and Mercurial by Bitbucket", *Bitbucket*, 2014. [Online]. Tillgänglig: https://bitbucket.org/ (Hämtad: 2014-06-03).

[6] V. D. Hunt, et al., *RFID: A Guide to Radio Frequency Identification*. Hoboken: John Wiley & Sons, 2007.

[7] M. Roberti, "The History of RFID Technology", *RFID Journal*, 2005. [Online]. Tillgänglig: http://www.rfidjournal.com/articles/view?1338 (Hämtad: 2014-04-05).

[8] M. Cardullo, "Genesis of the Versatile RFID Tag", *RFID Journal*, 2003. [Online]. Tillgänglig: http://www.rfidjournal.com/articles/view?392 (Hämtad: 2014-05-15).

[9] R. Das och D. P. Harrop, "RFID Forecasts, Players and Opportunities 2014-2024", 2013. [Online]. Tillgänglig: http://www.idtechex.com/research/reports/rfid-forecasts-players-and-opportunities-2014-2024-000368.asp?viewopt=showall (Hämtad: 2014-05-15).

[10] "Intelligent ID Card USB Reader", *DealExtreme*, 2014. [Online]. Tillgänglig: http://www.dx.com/p/intelligent-id-card-usb-reader-174455#.U3R-XygUxkN (Hämtad: 2014-05-15).

[11] "RFID Frequently Asked Question > How much does an RFID tag cost today?", *RFID Journal*, 2014. [Online]. Tillgänglig: http://www.rfidjournal.com/faq/show?85 (Hämtad: 2014-04-05).

[12] *Streckkodsinformation*. [Online]. Tillgänglig: `http://www.streckkod.se/category/452-streckkodsinformation.aspx` (Hämtad: 2014-04-19).

[13] M. Roberti, *What is the read range of a passive rfid tag?*, 2013. [Online]. Tillgänglig: `http://www.rfidjournal.com/blogs/experts/entry?10684`.

[14] *Atlas RFID solutions*, 2014. [Online]. Tillgänglig: `http://atlasrfid.com/auto-id-education/active-vs-passive-rfid/` (Hämtad: 2014-05-16).

[15] "Den svenska Frekvensplanen", *Post- och telestyrelsen*, 2014. [Online]. Tillgänglig: `http://e-tjanster.pts.se/radio/frekvensplanen/Service.aspx` (Hämtad: 2014-05-15).

[16] R. Want, "An Introduction to RFID Technology", *Pervasive Computing*, vol. 5, nr. 1, s. 25–33, januari 2006.

[17] M. Heró, S. Ohlsson och P. Wolf, "RFID - en teknologi för förbättrad lagerhantering", diss., Handelshögskolan vid Göteborgs Universitet, 2004.

[18] *The RFID Shop*, 2014. [Online]. Tillgänglig: `http://www.therfidshop.com` (Hämtad: 2014-04-15).

[19] "RFID Frequencies", *Brooks Automation*, [Online]. Tillgänglig: `http://www.brooks.com/applications-by-industry/rfid/rfid-basics/rfid-frequencies` (Hämtad: 2014-05-15).

[20] "Build Modern & Scalable Web Apps with Java and Scala", *Play Framework*, 2014. [Online]. Tillgänglig: `http://www.playframework.com/` (Hämtad: 2014-05-16).

[21] "Using the Ebean ORM", *Play Framework*, 2014. [Online]. Tillgänglig: `http://www.playframework.com/documentation/2.3.x/JavaEbean` (Hämtad: 2014-05-16).

[22] *TkInter*, 2014-04-24. [Online]. Tillgänglig: `https://wiki.python.org/moin/TkInter` (Hämtad: 2014-05-15).

[23] *TkDocs*, 2014. [Online]. Tillgänglig: `http://www.tkdocs.com/` (Hämtad: 2014-05-15).

[24] *jBCrypt*, 2010. [Online]. Tillgänglig: `http://www.mindrot.org/projects/jBCrypt/` (Hämtad: 2014-05-15).

[25] "The Web framework for perfectionists with deadlines", *Django Software Foundation*, 2014. [Online]. Tillgänglig: `https://www.djangoproject.com/` (Hämtad: 2014-05-16).

[26] "High Performance PHP Framework for Web Development", *Symfony*, 2014. [Online]. Tillgänglig: `http://symfony.com/` (Hämtad: 2014-05-16).

[27] "RFID Frequently Asked Question > How much information can an RFID tag store?", *RFID Journal*, 2014. [Online]. Tillgänglig: `http://www.rfidjournal.com/faq/show?66` (Hämtad: 2014-04-15).

[28] "Efficient warehouse management in real-time - Astro WMS", *Consafe Logistics*, 2014. [Online]. Tillgänglig: `http://www.consafelogistics.com/our-offer/warehouse-management/astrowms` (Hämtad: 2014-05-15).

[29] "Raspberry Pi, Model B, 512 MB", *Farnell*, 2014. [Online]. Tillgänglig: `http://se.farnell.com/jsp/displayProduct.jsp?sku=2191863`.

[30] D. Leifsson, Intervju, Platsschef på Dan Cargo Göteborg, 2014.

[31] E. Laago, Intervju, Lagerchef på IKEA Torsviksterminalen, 2014.

[32] A. Salehson, Intervju, Lecturer, Division of Networks and Systems, Department of Computer Science and Engineering, Chalmers University of Technology, 2014.

[33] L.-G. Johansson, Intervju, CEO of RFID System Sweden, 2014.

A

Kravspecifikation

Kravspecifikation
Här nedan följer de konkreta krav som vi räknar med ska vara uppfyllda vid projektets slut. Det finns två typer av krav; skallkrav och börkrav. Detta innebär att skallkraven kommer vara primära medan börkraven sekundära.
Här nedan följer en lista över våra skallkrav:

- En pall ska kunna märkas med en RFID-tagg som lagrar data om vilken artikel, samt kvantiteten av denna artikel, som finns på pallen. Denna data ska på ett enkelt sätt kunna skrivas till taggen.
- Med hjälp av en RFID-läsare ska det gå att läsa RFID-taggarna från föregående punkt och identifiera den data som finns på dem.
- Om en pall märkt med en RFID-tagg sätts på en “giltig” lagerplats ska datan på pallens tagg, tillsammans med platsinformationen automatiskt lagras i en databas.
- Om en pall med en RFID-tagg flyttas från en lagerplats till en annan ska denna ändring ske automatiskt även i databasen.
- Lagersystemet på Dan Cargo använder i nuläget platser med benämningar som exempelvis 19 05 35A för att identifiera en unik lagerplats. Dessa benämningar, samt nya benämningar på lagerplatser i framtiden, ska enkelt gå att lägga in i databasen.
- Det finns ytterligare ett skallkrav som vi enbart kan fullfölja om Dan Cargo tillåter det. Det handlar om att när alla de ovan beskrivna skallkraven är uppnådda, så ska hela lösningen vara kompatibel och gå att testa, i liten skala, med Dan Cargos egna system. Det vill säga deras lagerhållningssystem ska kunna hämta data från vår databas och på så vis uppdatera sig självt. Vilket innebär att databasen behöver ha ett API (Application Programming Interface) så att andra program både kan hämta och skriva data till den. Vi inser att detta inte ligger så mycket på vår bit, utan det blir då upp till Dan Cargos IT-avdelning om de vill låta oss fullfölja detta.

Här nedan följer en lista över våra börkrav:

- Det ska gå att tillbakaspåra var en pall befunnits från första gången den ställdes på en “giltig” lagerplats. Detta kanske genom att databasen har ett grafiskt gränssnitt som icke programmeringskunniga kan använda i form av ett excelark eller liknande. Detta utan att behöva vara rädda för att databasdesignen förstörs av ett par knapptryck.
- Det ska gå att spara ytterligare data på taggarna, på varje pall såsom exempelvis utgångsdatum.
- Det ska gå att i framtiden spara ytterligare data i databasen tillhörande taggarna. Exempelvis om man i senare skede vill ha ett anteckningsfält associerat med varje pall, ska det vara genomförbart.
- Databasen ska ha teoretisk implementerbar redundans, så att om systemet skulle implementeras på ett lager ska det enkelt gå att skapa kopior av datan. Annars riskerar man mardrömsscenariot att data för tusentals pallar försvinner med att en hårddisk går sönder.

B

Produktblad: PROMAG GP60A

PROMAG™

GP60A
Powerful Extended Range Proximity Reader

Introduction:

GP60A is a powerful extended range proximity reader featuring small dimensions and a read range of up to 60 cm. The unit will run from any voltage from 5 to 13.5 Vdc and making it particularly suited to access control, car parking and through-wall reading applications.

GP60A is available in most of the common interface formats, including Wiegand, Mag-stripe, RS-232 and RS-485 (optional version) serial ASCII output.

Features:

- ☆ **Relay control**
- ☆ **Auto-tuning on power up**
- ☆ **Weather resistant**
- Wide voltage range
- Externally programmable interface
- Potted for environmental protection

Applications:

- Access Control System
- Car Parking System
- Through-wall Reading Application
- Any Long Range Reading Application

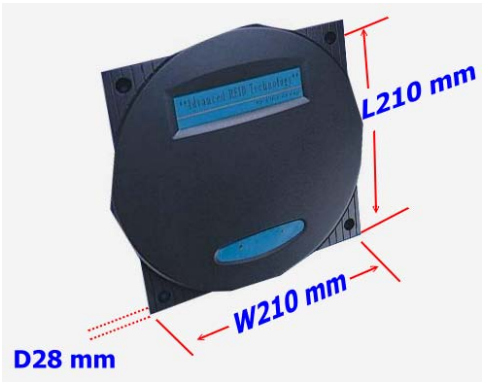
Specifications:

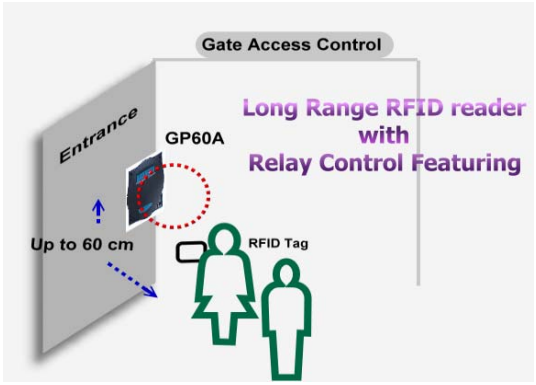
Power Requirements	DC 12V, 350mA, linear power regulator recommended
Interfaces	Wiegand 26 bits MSR ABA Track2 RS232 RS485 (optional version)
Read Range	Up to 60 cm with ISO card in ideal conditions
RFID cards accepted	125KHz, 64 bits, Manchester encoding
Maximum power switched by internal relay	Up to 24V / 2A
Dimensions	210 x 210 x 28 mm
Nominal weight	~ 900 g
Interface cable length	~ 90 cm
Operating temperature range	-10 ~ +60° C

Ordering information:

- GP60A –00 without power supply
- PS60A-00E – AC 230V optional power supply for GP60A
- PS60A-00U – AC 120V optional power supply for GP60A

※ **Specification is subject to change without notice.**





We welcome OEM inquiries

- **Custom design manufacturing is available**
- **Custom device programming is available**
- **Call factory for other configuration**

集佳股份有限公司
GIGA-TMS INC.

8F.,NO.31, LANE 169, KANG-NING STREET,HSI-CHIH,TAIPEI,TAIWAN

TEL: 886-2-26954214

FAX: 886-2-26954213

e-mail: promag@ms24.hinet.net

gigatms@ms3.hinet.net

http://www.gigatms.com.tw

C

Manual: PROMAG GP60A

GP60A

Long-range RFID (proximity) card reader

(1) Features:

- Extended reading range of up to 60 cm with ISO-size cards
- High-precision auto tuning compensates for environmental changes
- Four user-selectable interfaces: Magstripe, Wiegand, RS232, RS485
- Firmware Download Mode allows for firmware upgrades in the field
- Internal relay and serial port commands to control it
- Six function LEDs, one of them externally controlled
- Wide input power range (6.5V-15V, 12V nominal)
- Compact and stylish design
- Electronic circuit completely sealed (potted)

(2) Cable wire assignment

	Magstripe I/F	Wiegand I/F
Green	Data	Data0
White	Clock	Data1
Orange	Card present	*<unused>*
Dark Blue	RS232, RX	
Brown	RS232, TX	
Light Blue	+RS485	
Light Green	-RS485	
Yellow	I/F selection, see (6)	
White/Black	Enable Firmware Download Mode, see (11)	
Pink	External LED control, see (9)	
Black	Relay, common	
Purple	Relay, normally opened	
Gray	Relay, normally closed	
Red	Power (6.5V-15V, 12V nominal), see (5)	
Shield with black wire	Ground	

(3) LED signaling

	Normal mode	Firmware Download Mode
Power (green)	Always on when power is applied	
Relay (Green)	On when internal relay is activated	
Comms/Dload (Green)	Blinks when command is received	Displays download status
Tuning/Dload (Red)	Displays tuning result	Displays download errors
Ext (Red)	Externally controlled LED	
Red (Red)	Blinks once when good read occurs	

(4) Installation requirements

For best reading results the GP60A should be installed on a metal-free surfaces and away from large metal objects. Despite high-precision auto-tuning circuitry the presence of metal objects decreases the reading range of the GP60A. Presence of metal and the degree to which it affects the operation of the GP60A can be verified by checking the results of auto-tuning. See (7) for more details.

(5) Power supply

Best reading results are achieved when the GP60A is used with a high-quality linear power supply (12VDC, min 500mA). Using low-cost power adaptors severely affects the reading distance of the GP60A.

(6) Interface selection & lines

Interface lines for Magstripe, Wiegand, and RS232 interfaces, as well as interface selection are 100% compatible with the original GP60:

Magstripe	Connect Yellow to Orange
Wiegand	Connect Yellow to White
RS232 (or RS485)	Leave Yellow unconnected

The GP60A features both the RS232 and the RS485 interfaces. Only one of them can be used at a time. That is, if you are using RS485 then leave RS232 lines unconnected, and vise versa! Serial communications parameters for both interfaces are fixed at 9600,N,8,1.

(7) Auto-tuning

The GP60A is equipped with an advanced 8-bit auto-tuning circuitry that compensates for the presence of metal objects in the vicinity of the GP60A and for other environmental factors that can affect the tuning of the GP60A's internal antenna and decrease the reading distance. Although it is best to install the GP60A away from metal objects and on a metal-free surface this is not always possible. Auto-tuning recovers significant portion of the reading distance that otherwise would have been lost due to the presence of metal.

The auto-tuning is performed once each time the GP-60 is powered up. Moving the GP60A after the auto-tuning voids the tuning results. Always switch the GP60A off and on to repeat the auto-tuning after relocating the GP60A or changing the environment around it.

The GP60A displays the results of auto-tuning (and expected reading performance) by blinking its Tuning LED and beeping a certain number of times:

Number of blinks/beeps	Tuning condition
3 times	Good
2 or 4 times	Acceptable
5 times	Poor
1 time or 6 times	Unacceptable

(8) Internal relay

The GP60A features an internal relay that can be controlled by sending commands via the RS232 or RS485 interface. Relay can be used to control an electric lock. Commands are described in (10).

(9) Externally controlled LED

The "Ext" LED on the front panel can be controlled externally. The LED is lighted up by applying 3V-15V to the Pink Wire (there is a 1.5K resistor connected in series with this LED).

(10) Serial commands

All commands are sent to the GP60A via its RS232 or RS485 port. For each command sent the GP60A will issue a reply. Serial port settings are 9600,N,8,1. Commands and replies have the following format:

STX(02h)	Command or reply code	Check field	CR(0Dh)

Currently supported commands (with their check field already calculated and shown in gray) are shown below (without STX and CR):

Switch the relay on (relay enable)	"RE68"
Switch the relay off (relay disable)	"RD69"

The GP60A can issue two possible replies:

Command acknowledged	"ABE"
Invalid command	"CBC"

(11) Firmware Download Mode

Internal firmware of the GP60A can be upgraded in the field using a new Firmware Download Mode. The download is effected through the serial port (communications parameters are 9600,N,8,1). To download a firmware file you'll need any PC software that supports an XMODEM communications protocol (checksum version). Procedures below assume the use of *HyperTerminal* for *Windows*:

- Power the GP60 off
- Connect the RS232 port of the GP60A to the PC
- Ground the Download wire of the GP60A
- Launch the *HyperTerminal* and configure it for an appropriate COM port with communications parameters set to 9600,8,N,1.
- Choose *Transfer→ Send file* from the *Main* menu- the *Send file* dialog will appear
- In the *Send file* dialog, select the firmware file that you want to download into the GP60A and choose the *Xmodem protocol* from the *Protocol* drop-down box. Click *OK* when finished. The *Xmodem file send for* a dialog will be displayed
- Power the GP60A up- the download will begin
- The Green "Dload" LED of the GP60A will blink during the download

- When the download is finished, untie the Download wire from the ground and switch the GP60A off and back on- the reader will resume normal operation
- If, when you power the GP60A up after the download, both Green and Red "Dload" LEDs start to blink this means that you have downloaded a wrong file or that the download process was not completed.

A number of errors can occur during the download. These errors are displayed by the Red "Dload" LED:

1 long "blink"	Communications timeout (check baudrate, protocol)
1 long + 1 short "blink"	Communications error (check baudrate, protocol)
1 long + 2 short "blinks"	Invalid data file (check which file you are downloading)
1 long + 3 short "blinks"	Hardware failure (this GP60A is malfunctioning)

(12) Data output formats

Magstripe interface, simulated to 38 IPS (inches per second)

10 leading zeros	SS	Data (14 digits)	ES	LRC	10 trailing zeroes

Wiegand Interface (26-bit format):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
P	S	S	S	S	S	S	S	S	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	P
P	E	E	E	E	E	E	E	E	E	E	E	E													
													O	O	O	O	O	O	O	O	O	O	O	O	P
Summed for even parity (E)													Summed for odd parity (O)												

P- parity (Even/Odd), S- site bits, C- card data

Serial interface (9600,N,8,1)

STX (02h)	Data (10x "hex" ASCII chars)	CR (0Dh)	LF (0Ah)	ETX (03h)

(13) Specifications

Power Requirements	12V, ~0.25A, linear power regulator recommended!
Interfaces	Magstripe, Wiegand (24 bits), RS232 & RS485 (9600,N,8,1)
Maximum reading range (with ISO-size card in a noise-free environment, when installed on a metal-free surface and powered by a linear low-noise power supply)	~60cm
RFID (proximity) cards accepted	125KHz, 64 bits, Manchester encoding
Maximum power switched by internal relay	Up to 24V/2A
Dimensions	210x210x28mm
Nominal weight	~900g
Interface cable length	~90cm
Operating temperature range	-10 ~ +60 C°