YRKESHÖGSKOLAN
NOVIA

Utveckling av en webbapplikation för lagerhantering

Jonathan Svahn

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2017

EXAMENSARBETE

Författare:	Jonathan Svahn
Utbildning och ort:	Informationsteknik, Vasa
Handledare:	Susanne Österholm

Titel: Utveckling av en webbapplikation för lagerhantering

Datum: 10.4.2018 Sidantal: 31

Abstrakt

Detta examensarbete utfördes på uppdrag av eCraft Ab. Arbetets syfte var att skapa ett skräddarsytt system för att underlätta planering och utförande av lagerarbete för kunden Jeppo Potatis Ab. Systemet skulle implementeras som en del av projektet JEPOStore, en webbapplikation under utveckling.

Systemet utvecklades i ASP.NET MVC med SQL Server som databashanterare. Dessa tekniker var givna, eftersom de redan användes i JEPOStore. Resultatet av arbetet blev ett integrerat verktyg för planering och utförande av arbete som används i dagligt bruk av Jeppo Potatis lagerarbetare och planerare.

Språk: svenska Nyckelord: ASP.NET MVC, Full-Stack Web Development

---

OPINNÄYTETYÖ

Tekijä:	Jonathan Svahn
Koulutus ja paikkakunta:	Tietotekniikka, Vaasa
Ohjaaja:	Susanne Österholm

Nimike: Varastohallinnan verkkosovelluksen kehittäminen

Päivämäärä: 10.4.2018 Sivumäärä: 31

Tiivistelmä

Tämä opinnäytetyö tehtiin eCraft Oy:n toimeksiannosta. Opinnäytetyön tavoitteena oli luoda räätälöity järjestelmä asiakkaan Jeppo Potatis Oy:n varastotoiminnan suunnittelun ja toteuttamisen helpottamiseksi. Järjestelmä toteutettiin osana JEPOStore-projektia, joka on kehitteillä oleva verkkosovellus.

Järjestelmä kehitettiin hyödyntäen ASP.NET MVC:tä ja SQL Server tietokantanahallintajärjestelmää. Koska nämä tekniikat olivat jo käytössä JEPOStoressa, niiden käyttö tässä työssä oli ilmiselvää. Työn tuloksena syntyi integroitu työkalu varastotoiminnan suunnitteluun ja toteutukseen, jota Jeppo Potatis Oy:n varastotoimittajat ja suunnittelijat käyttävät päivittäin.

Kieli: ruotsi Avainsanat: ASP.NET MVC, Full-Stack Web Development

---

BACHELOR'S THESIS

Author:	Jonathan Svahn
Degree Programme:	Information Technology
Supervisor:	Susanne Österholm

Title: Development of a Web Application for Warehouse Management

---

Date: April 14, 2018 Number of pages: 31

Abstract

This Bachelor's thesis was performed as a task given by eCraft Oy. The goal of the thesis was to create a tailor-made system to make planning and performing warehouse work for the customer Jeppo Potatis Oy easier. The system was to be implemented as a part of the project JEPOStore, a web application under development.

The system was developed in ASP.NET MVC with SQL Server as database management system. Since these technologies were already in use in JEPOStore their use in this thesis was a given. The work resulted in an integrated tool for planning and performing warehouse work, used by the warehouse workers and planners of Jeppo Potatis Oy.

Language: Swedish Key words: ASP.NET MVC, Full-Stack Web Development

---

Innehållsförteckning

1 Inledning

Det här examensarbetet beskriver utvecklingen av en webbapplikation gjord för Jeppo Potatis Ab, ett österbottniskt företag som tillverkar potatisprodukter av potatisar köpta direkt från odlare. Jeppo Potatis är en kund till mjukvaruföretaget eCraft Ab, som är uppdragsgivare för det här arbetet. (JEPO, 2018)

Webbapplikationen är ett stort projekt med flera utvecklare. Arbetet kommer att kort presentera helheten, men fokusen kommer att ligga på två utvalda centrala moduler. Beskrivningar av annan funktionalitet fungerar som stöd för förståelsen av dessa moduler.

1.1 Verksamhet

Jeppo Potatis verksamhet består av att köpa in, lagra och förädla potatis. Råvaruinköp sker i regel direkt från producenter. Alla inkommande potatislass vägs och bokförs, och förs sedan in till lagret av en truckförare.

Potatisarna i varje lass blir analyserade och storlekssorterade, och i sinom tid flyttade till produktionsavdelningen för att förädlas till olika potatisprodukter. De färdiga produkterna säljs i sin tur vidare till återförsäljare.

1.2 Utmaningar och tidigare lösningar

Det här underkaptitlet behandlar de relevanta utmaningarna inom Jeppo Potatis verksamhet samt deras tidigare lösningar. Kapitlet är uppdelat i två underkapitel, vart relevant till en av modulerna som behandlas i det här examensarbetet.

1.2.1 Lagring

Potatislådorna som lagras är stora och utrymmet i lagerlokalerna begränsat. Det här betyder att placeringen i lagret blir viktig. Lådorna placeras i rader av staplar, så att för varje rad kan en truckförare endast komma åt den yttersta stapelns högsta låda. Det här, kombinerat med den höga råvaruvolymen (tusentals ton per verksamhetsplats), innebär att bokföringen av lagrets innehåll och lådornas specifika placering är viktig. Som ett förberedande steg till digitalisering har den här bokföringen utförts genom att under dagens lopp skriva ner lådplaceringar på papper och sedan mata in informationen i Microsoft Excel-filer. Tidigare

har det inte funnits ett explicit system för det här, utan man har förlitat sig på en erfaren personal.

1.2.2 Produktionsplanering

Planerarna måste först avgöra vilka produkter som ska tillverkas och i vilka mängder. I de beräkningarna ser man bland annat på öppna försäljningsordrar, förutspådda försäljningsordrar och tidsbegränsningar.

Sedan avgörs det vilken råvara som ska användas per produkt. Vilken råvara som är optimal för en produkt avgörs med hjälp av analysresultat och storlekssorteringar.

Tidigare har det här gjorts med hjälp av Microsoft Excel, JEPONET (ett omfattande mjukvarusystem som bland annat bokför analysresultat), och en erfaren personal.

1.3 Vision för mjukvarulösning

Ända från projektets planeringsskede har man pratat om mjukvarulösningen som två projekt: JEPOStore och JEPOSteer. Idag är de två projekten så tätt sammanbundna att det i det här examensarbetet är mer praktiskt att prata om dem som en applikation, men för att beskriva visionen för applikationen är det enklare och mer sanningsenligt att skilja på projekten.

1.3.1 JEPOStore

JEPOStore skulle bli applikationen som skulle användas av främst truckpersonalen. Varje truck skulle utrustas med en dator med pekskärm som truckförarna skulle använda för att utföra lassinvägningar, bokföra lådflyttar och inventera lagret. Allt som förut hade gjorts med papper, penna och Excel skulle gå att göra direkt i truckdatorn, med så få klick som möjligt. Applikationen skulle också kunna ge rekommendationer om lådplaceringar samt informera användaren om jobb som behövde göras. Eftersom användningen av applikationen skulle ske parallellt med det fysiska arbetet i kyliga lager var det viktigt att användargränssnittet skulle gå att använda även med arbetshandskar.

1.3.2 JEPOSteer

JEPOSteer skulle användas av produktionsplanerarna. Applikationen skulle hämta öppna försäljningsordrar från det befintliga orderhanteringssystemet och användarna skulle planera morgondagens produktionsdygn mot dessa. I planeringen skulle bland annat ingå

råvarurekommendationer, produktionssekvens, samt automatiska uträkningar av produktions- och råvarubehov. Det planerade produktionsdygnet skulle visas i JEPOStore, så att truckförarna kunde utföra det arbete som behövdes.

För övrigt bör det nämnas att stor tyngd lades på att applikationen skulle vara dynamisk. Användaren skulle kunna skapa och modifiera saker som lagerlokaler och produktstrukturer, och applikationen skulle gå att tas i bruk på flera verksamhetsplatser.

2 Uppgift

Examensarbetets fokus låg på två huvudmoduler: *arbetslistan*, en interaktiv lista över arbetsuppdrag som truckförarna skulle använda i majoriteten av sina arbetsmoment; och *produktionsorderplaneringen*, en vy som planerarna skulle använda för att detaljplanera ett produktionsdygns produktionssekvenser och råvaruanvändning. Det här kapitlet introducerar dessa moduler och beskriver de arbetsprocesser de skulle skapas för, men lämnar beskrivningen av modulernas implementation till senare kapitel.

2.1 Arbetslistan

Det här underkapitlet introducerar arbetslistan genom att diskutera dess syfte och bakgrund, samt planen för en mjukvarulösning.

2.1.1 Syfte och bakgrund

En truckförares arbete består i huvudsak av att uföra lådflyttar. Lådflyttar kan ske av många skäl: det kan röra sig om en ny inleverans, en storlekssortering, en produktionsorderfyllning eller bara ett behov av att komma åt en låda som finns längre bak i raden. För varje skäl skiljer sig arbetsprocessen och informationsbehovet.

Ett exempel är de två typerna av inleveranser: ett potatislass anländer antingen med potatisarna lösa på ett släp eller färdigt satta i lådor. I det senare fallet består arbetet av att flytta lådor från släpet till en lagerplats. För ett löslass behöver potatisarna först sättas i lådor, och man tar då tillfället i akt för att göra en grov storlekssortering, vid vilken etiketter där potatisstorleken framgår skrivs ut och fästs på lådorna. Efter detta tas lådorna till olika lagerplatser. I båda inleveransfallen behövs information om vilken lagerplats lådorna ska tas till, men till skillnad från till exempel en produktionsorder-fyllning behövs ingen information om vilken lagerplats lådorna ska tas *från*.

Funktionalitet för de flesta av dessa arbetsprocesser fanns redan i ett tidigt skede av JEPOStores utveckling. Varje unik arbetsprocess hade sin egen vy, där användaren berättade för applikationen vilket lass eller vilken låda arbetet gällde, fick information och bokförde det utförda arbetet.

Den här lösningen hade dock brister. Användaren var tvungen att navigera till en ny vy varje gång hen skulle utföra en annan arbetsprocess. En annan brist var att applikationen saknade överskådlighet över arbetsläget – applikationen hade till exempel information om anlända inleveranser, men detta presenterades inte åt användaren, utan användaren var tvungen att själv mata in lassnumret för att utföra arbetet.

2.1.2 Lösning

Lösningen som slutligen valdes var att ge varje arbetsprocess sitt eget minimalistiska användargränssnitt, och samla varje instans av en arbetsprocess i en gemensam interaktiv lista. Med andra ord skulle truckföraren meddelas om jobb som behövde göras genom att en rad visades i en lista, och raden kunna expanderas för att öppna ett gränssnitt för att utföra arbetet. En rad skulle till exempel kunna motsvara en specifik inleverans som behövde flyttas in i lagret. Användaren skulle inte längre behöva berätta för applikationen vilket arbete hen vill göra, utan applikationen skulle visa åt användaren vad som behövde göras och förse denne med det nödvändiga gränssnittet. Hädanefter kommer listan att kallas för *arbetslistan*, och en rad för ett *arbetsuppdrag*.

Med arbetslistan kunde användaren med ett klick navigera från en typ av arbetsuppdrag till ett annat, och arbetsuppdragen kunde strömlinjeformas efter sin unika process. Med arbetslistan kom också möjligheten att automatiskt skapa arbetsuppdrag vid vissa händelser och på så sätt kunna meddela användaren om det jobb som behövde göras, eller ge förmän möjlighet att styra arbetet i lagret genom att manuellt skapa arbetsuppdrag.

2.2 Produktionsorderplanering

Det här underkapitlet introducerar produktionsorderplaneringen genom att diskutera planerarnas arbete, vilka behov de har av en mjukvarulösning samt planen för lösningen.

2.2.1 Planerarnas arbete

Genom den andra modulen, Produktionsorderplaneringen, utförs råmaterialreservering och sekvensstyrning för dygnets produktion. Vilka produkter som ska produceras i vilka mängder har i det här skedet redan blivit fastslaget.

För att bestämma vilka potatislådor som ska användas till en slutprodukt finns det flera saker man tar i beaktande. Främst ser man på potatisens egenskaper – saker som konsistens, form och storlek avgör om potatisen passar till en viss slutprodukt eller ej. När en slutprodukt produceras används i regel en enda potatissort, och en enda potatisstorlek.

På grund av lagrets arrangemang (man når endast den yttersta lådan i varje rad) måste man också ta lådornas tillgänglighet i beaktande. I takt med att lådor förs till produktionsavdelningen blir nya lådor tillgängliga, och även detta måste tas i beaktande. Ordningen som lådor kommer att föras bort i kan förutspås genom att se på dygnets produktionssekvens.

Efter att planeringen är gjord för truckförare lådorna till produktionsavdelningen, vilket är ett temporärt lager med begränsat utrymme som ordnas på samma sätt som övriga lager – det här innebär att truckförarna måste veta var de bör placera lådorna så att de finns tillgängliga när den slutprodukt de är reserverade för produceras.

2.2.2 Behov

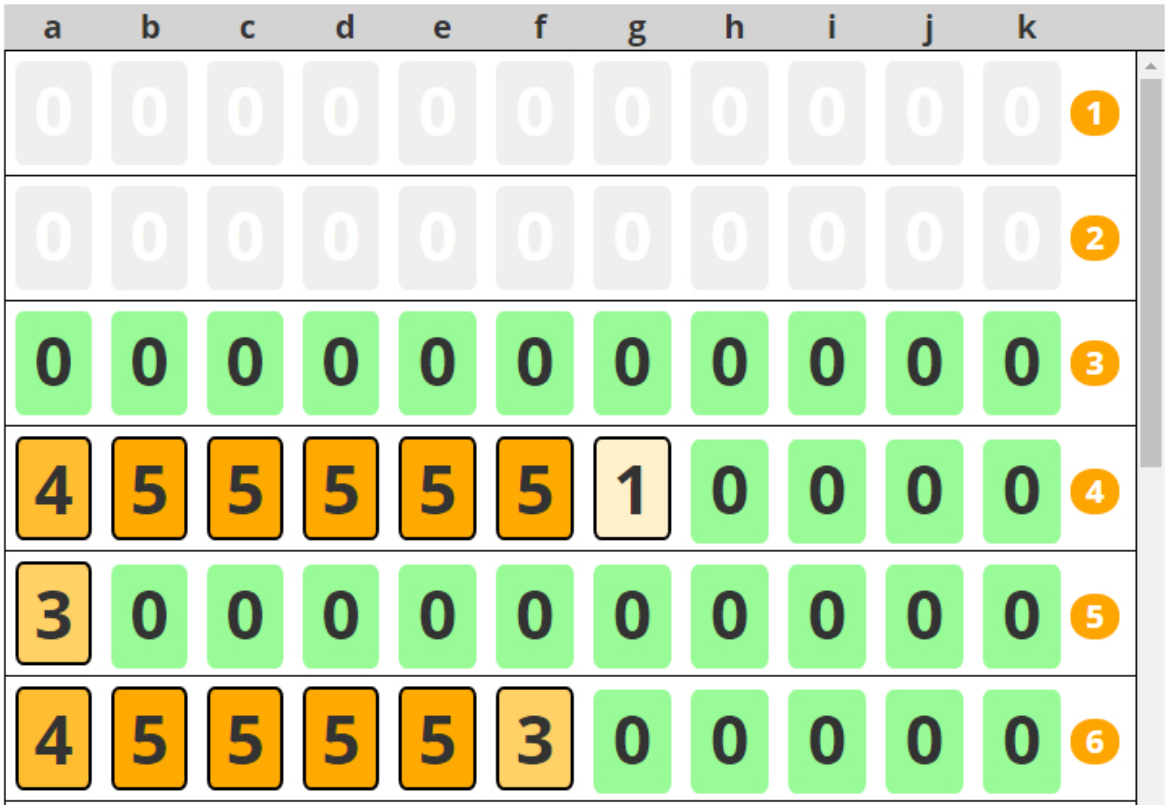
Användaren behövde få veta vilka lådor i lagret som passar till de olika slutprodukterna (utgående från definierade egenskapsprioriteringar), och möjlighet att reservera dessa till en produktionsorder. Applikationen måste också förse användaren med information om lådornas tillgänglighet beroende på den valda produktionsorderns produktionssekvens. Den här tillgängligheten påverkas av reservationer samt produktionssekvensförändringar. För att förenkla arbetet behövde applikationen också kunna automatiskt reservera de bäst lämpade lådorna för varje produkt, samt föreslå en produktionssekvens.

Slutligen måste applikationen kunna förmedla planen till truckförarna, så att planen kan bli verklighet.

2.2.3 Lösning

Lösningen som valdes var en interaktiv lagerkarta. Lagerkartor hade tidigare blivit använda på andra platser i applikationen, och hade visat sig vara både lättanvända och bra på att delge mycket information i ett kompakt format.

En lagerkarta i det här sammanhanget innebär en grafisk representation av lådors position i lagret, i tre dimensioner. Rader och kolumner ritas upp och resulterar i celler som representerar lådstaplar. En skärmdump av en lagerkarta visas i figur 1. Siffran i varje cell representerar lådstapelns höjd (t. ex. en trea innebär att stapeln innehåller tre lådor). Cellernas bakgrundsfärg innebär olika saker: grå bakgrundsfärg innebär en inaktiverad cell, grön innebär ledig, och de olika nyanserna av gulbrun motsvarar de olika stapelhöjderna. Genom att klicka på en cell fås information om den cellen.



Figur 1. Lagerkarta.

För att snabbt förmedla all relevant information om lådorna i en cell behövdes nya grafiska symboler (i likhet med siffran och bakgrundsfärgen) för att representera reservationsstatus, projicerad tillgänglighet (baserat på reservationens produktionssekvens relativt till vald sekvens), samt lämplighet (baserat på analysresultat jämfört med definierade egenskapspreferenser). Något som komplicerade det hela är att en cell, eller stapel, kunde

innehålla lådor av olika typer – det här innebar att applikationen måste kunna summera flera lådors innehåll i cellens symbolik.

Eftersom den ovan nämnda informationen inte skulle vara densamma för olika slutprodukter måste kartans innehåll uppdateras vartefter användaren valde produktionsorder.

Användaren behövde ett enkelt sätt att reservera lådor. I regel används många lådor från samma odlingsskifte till en slutprodukt. För att underlätta det här skulle applikationen visa en interaktiv lista över lämpliga lådor grupperade enligt skifte.

Ett sätt att reservera individuella lådor behövdes också. Ett enkelt sätt hade varit att klicka på en cell för att reservera den, men eftersom en cell representerar en stapel av lådor är det här inte möjligt. Lösningen som valdes var att öppna en enkel dialog precis bredvid pekaren när användaren klickar på en cell. Dialogen skulle visa de individuella lådorna i stapeln och låta användaren reservera dem.

Som tidigare nämnt strävar man till att endast använda en sort och en storlek per slutprodukt. Här skulle applikationen hjälpa till genom att erbjuda filtreringsalternativ. Filtreringen skulle påverka lagerkartan genom att visa filtrerade lådor som olämpliga och dölja dem i listan över lämpliga lådor grupperade enligt skifte.

Slutligen behövdes ett sätt att förmedla den färdiga planen till truckförarna. Det här skulle applikationen göra genom att skapa arbetsuppdrag i arbetslistan. Arbetsuppdragen skulle berätta för truckföraren vilka lådor de ska hämta, vart de ska föra dem, och i vilken ordning det helst borde ske.

Eftersom ett planerat produktionsdygn förverkligas genom att skapa arbetsuppdrag för truckförarna i arbetslistan är dessa två moduler logiskt sammanlänkade, och genom att presentera dem tillsammans bildas en naturlig helhet.

3 Teori och teknik

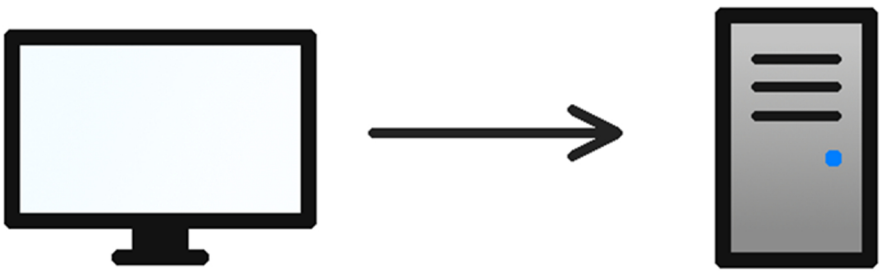
I det här kapitlet introduceras de tekniker och verktyg som har använts i utförandet av uppdraget. En överblick av hela systemet byggs stegvis upp, sedan förklaras de olika delarna var och en för sig.

3.1 Systemöverblick

Applikationen är implementerad som en webbapplikation. Den mest grundläggande abstraktionen man kan göra av en webbapplikation är att dela upp systemet i en klient och en server, eller en webbläsare och en webbserver.

3.1.1 Webbläsaren och webbservern

Webbserverns uppgift är att hantera förfrågningar från webbläsaren och svara med vyer och information. Webbläsaren presenterar vyerna för användaren, hanterar användarens input och skickar nya förfrågningar till webbservern. Figur 2 illustrerar kommunikationen mellan webbläsaren och servern, representerade av datorskärmen till vänster respektive datorn till höger. Pilen däremellan representerar kommunikationens riktning – webbläsaren skickar förfrågningar till webbservern, och webbservern svarar.



Figur 2. Webbläsaren och webbservern.

3.1.2 Webbläsarens presentation

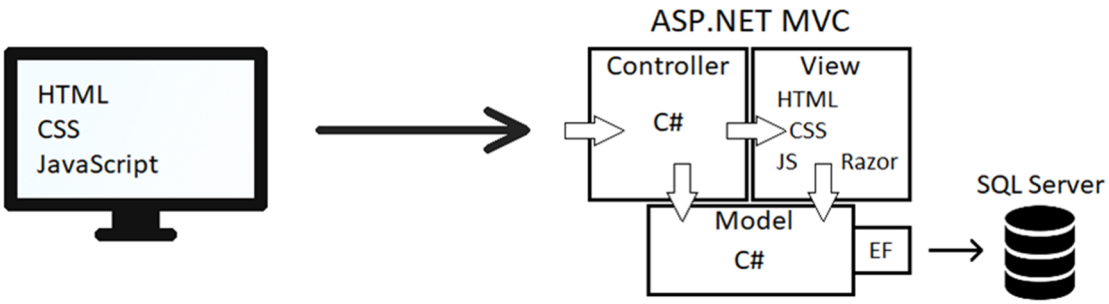
Servern förser webbläsaren med vyer i form av märkspråket *HTML*, stilmallen *CSS* och skriptspråket *JS*. Webbläsaren presenterar informationen i ett användarvänligt format.

3.1.3 Webbserverns arkitektur

På webbservern körs en *ASP.NET MVC*-applikation. MVC står för Model-View-Controller och är ett arkitekturmönster som delar upp applikationen i ett presentationslager, ett lager för data- och affärslogik, och ett lager för interaktionen mellan de förstnämnda. I ASP.NET MVC används HTML, CSS, JavaScript, programmeringsspråket *C#* och mallsyntaxen *Razor*. (ASP.NET MVC, 2017) (ASP.NET Razor, 2017).

3.1.4 Databas

För att lagra information använder applikationen databasen *SQL Server*. För att underlätta kommunikationen mellan webbapplikationen och databasen används *Entity Framework* (EF). I figur 3 har serverillustrationen ersatts med ett diagram över ASP.NET MVC-systemet, databasen och kommunikationen därmellan. Datorskärmen visar nu webbläsarens tekniker.



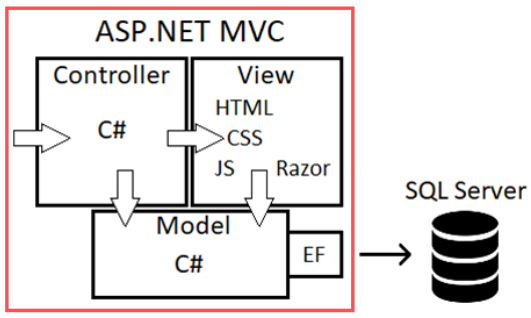
Figur 3. Webbläsaren och webbserverns tekniker.

3.2 Systemets delar

Det här underkapitlet diskuterar delarna i den systemöverblick som presenterades i föregående kapitel i högre detalj, var och en för sig.

3.2.1 ASP.NET MVC

ASP.NET MVC är ett webbapplikations-ramverk utvecklat av Microsoft. MVC är en förkortning för *Model-View-Controller* och är som tidigare nämnt ett arkitekturmönster som separerar applikationens funktionalitet i olika lager. Figur 4 identifierar ASP.NET MVC i systemöverblicksbilden från figur 3.



Figur 4. ASP.NET MVC i en systemöverblick.

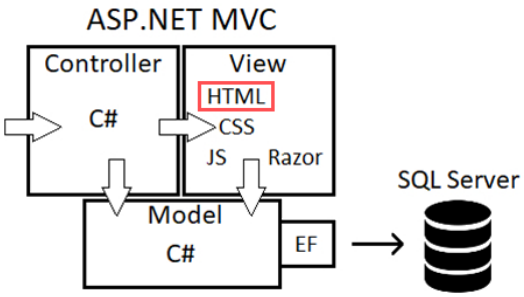
Controller kan sägas vara kontaktytan med webbläsaren. Den är ett tunt lager som tar emot förfrågningar, anropar andra delar av applikationen, och sänder tillbaka vyer eller information till webbläsaren.

I *View* konstrueras vyn som skickas till webbläsaren. Vyn är alltså webbsidan, som består av HTML, CSS och JS. I ASP.NET MVC används Razor för att skapa vyn dynamiskt. Controller anropar View för att få en vy att skicka som svar på en förfrågan från en webbläsare.

Model hanterar applikationens data, logik och regler. Controller och View kan anropa Model för att bland annat hämta, manipulera eller beräkna data. (ASP.NET MVC, 2017)

3.2.2 HTML

HTML, eller Hypertext Markup Language, är den mest grundläggande delen av en webbsida. HTML är ett språk som beskriver strukturen av webbsidor i form av en trädstruktur innehållandes bland annat tabeller, rubriker och paragrafer. Figur 5 visar HTML i system-överblicken. Ett exempel på HTML visas i kodexempel 1. (HTML, 2017)



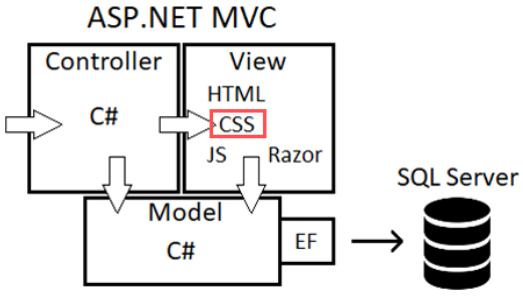
Figur 5. HTML i en systemöverblick.

Kodexempel 1. HTML-exempel.

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <link rel="stylesheet" type="text/css" href="button-styles.css">
    <title>Exempelsida</title>
</head>
<body>
    <button class="uncool-css">Click</button>
</body>
</html>
```

3.2.3 CSS

CSS, eller ”Cascading Style Sheets”, är ett stilmallsspråk som beskriver presentationen av bland annat webbsidor. CSS kan kombineras med HTML för att styra bland annat färg, upplägg och teckensnitt. Figur 6 visar CSS i systemöverblicken. Kodexempel 2 och figur 7 illustrerar två CSS-klasser och hur de påverkar utseendet på knappar. (Cascading Style Sheets, 2017)



Figur 6. CSS i en systemöverblick.

Kodexempel 2. Två CSS-klasser.

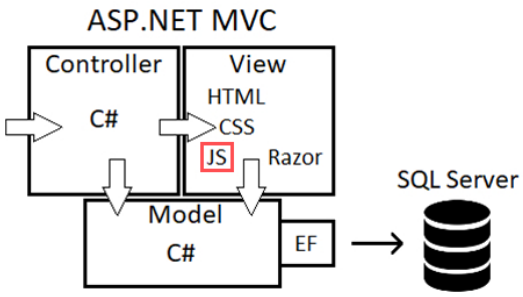
```
.uncool-css{
    background-color:dimgray;
    border:double;
    border-color:black;
}

.cool-css {
    background-color: crimson;
    border: none;
    padding: 15px 32px;
    color: white;
    font-size: 16px;
}
```

Figur 7. Två knappar av olika CSS-klasser.

3.2.4 JavaScript

JavaScript, eller JS, är ett skriptspråk som tillsammans med HTML och CSS utgör webbens tre kärnteknologier. HTML beskriver en webbsidas struktur, CSS dess presentation och JS dess beteende och interaktivitet.



Figur 8. JavaScript i en systemöverblick.

JavaScript möjliggör till exempel förändringar i sidans innehåll, animationer och uträkningar. JavaScript-funktioner kan knytas till, eller *prenumerera på*, olika händelser – till exempel kan man skapa en funktion som anropas när användaren för musen över ett visst område på sidan. Eftersom JavaScript kan köras lokalt i användarens webbläsare möjliggörs snabba reaktioner på användarhandlingar som inte kräver en rundtur via webbservern. Figur 8 visar JavaScript i systemöverblicken. Kodexempel 3 visar ett exempel på en JavaScript-funktion som är ämnad att ta emot ett HTML-element och ändra dess utseende genom CSS-klasser. Kodexempel 4 visar HTML som definierar en knapp som anropar metoden i kodexempel 3 när användaren klickar på den. (JavaScript, 2017)

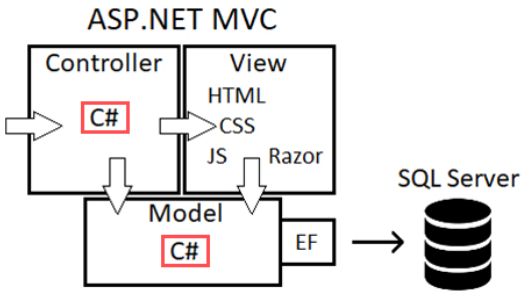
Kodexempel 3. En JavaScript-funktion.

```
function blingify(element) {
    $(element).removeClass('uncool-css');
    $(element).addClass('cool-css');
}
```

Kodexempel 4. Ett HTML-element som kan anropa en JavaScript-funktion.

```
<button class="uncool-css" onclick="blingify(this)">Click</button>
```

3.2.5 C#



Figur 9. C# i en systemöverblick.

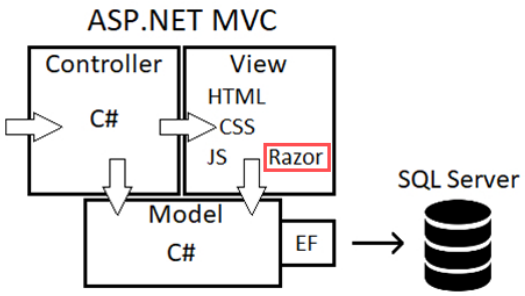
C# (uttalat C-Sharp) är ett objektorienterat programmeringsspråk utvecklat av Microsoft. Objektorientering innebär att koden struktureras som en uppsättning objekt som interagerar med varandra. Ett objekt är en instans eller realisering av en klass, som kan sägas vara objektets typ och definierar både egenskaper och funktionalitet. Det här speglar vår intuition om hur den fysiska världen består av objekt av olika typer – en god liknelse skulle vara hur *klassen* ”Bok” kan sägas kunna ha ett antal sidor, text skrivet på ett visst språk, en pärmbild, och så vidare. *Objektet* ”The Hitchhiker's Guide to the Galaxy” är ett objekt av klassen ”Bok” och har 193 sidor, text på engelska, en kaffefläck på sida 15, och så vidare. Figur 9 visar C# i systemöverblicken. Kodexempel 5 visar ett exempel på en klass skriven i C#. (C# (programming language), 2017) (Objektorienterad programmering, 2017) (Goodreads, 2017)

Kodexempel 5. Klassen “Book” skriven i C#.

```
class Book
{
    string Title;
    IEnumerable<Page> Pages;
    int NumberOfPages
    {
        get
        {
            return Pages.Count();
        }
    }
    string ReadPage(int pageNumber)
    {
        Page page =  Pages.ElementAt(pageNumber);
        return page.Text;
    }
}
```

3.2.6 ASP.NET Razor

Razor, som används i ASP.NET MVC:s View-lager, är ett programmeringssyntax som används för att skapa webbsidor dynamiskt. Att skapa webbsidor dynamiskt innebär att samma sidas innehåll och struktur automatiskt anpassar sig till variabler. Figur 10 visar Razor i systemöverblicken. I kodexempel 6 visas ett exempel på hur man med Razor-syntax kan bygga upp en lista i HTML med ett variabelt antal rader.



Figur 10. Razor i en systemöverblick.

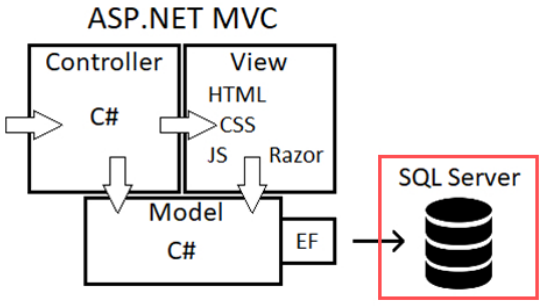
Kodexempel 6. Razor.

```
<ul>
    @foreach (var measurement in Model)
    {
        <li>
            @measurement
        </li>
    }
</ul>
```

I kodexempel 6 instrueras razor att för varje (for each) mätning (measurement) i samlingen Model skriva ut innehållet mellan klamrarna till HTML-filen. Ifall det finns 10 mätningar i samlingen Model kommer alltså innehållet mellan klamrarna att upprepas 10 gånger. ”
” och ”” är HTML-taggar som definierar början och slutet på ett listobjekt, och raden ”@measurement” instruerar Razor att skriva ut mätvärdet innanför dem. (ASP.NET Razor, 2017)

3.2.7 SQL Server

SQL Server är ett relationsdatabashanterings-system utvecklat av Microsoft. En databas är en organiserad informationssamling. I en relationsdatabas är information organiserad i en eller flera tabeller, och en tabell har möjlighet att ha relationer till andra tabeller. Figur 11 visar SQL Server i system-överblicken. (Microsoft SQL Server, 2017) (Relational database management system, 2017)



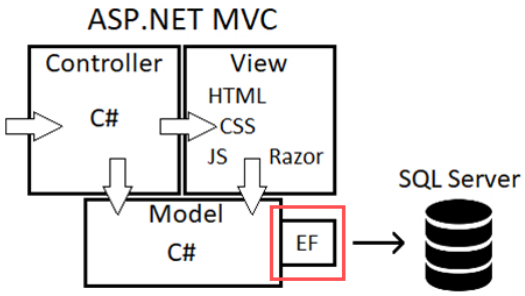
Figur 11. SQL Server i en systemöverblick.

En tabell i SQL Server består av kolumner som är definierade att lagra en viss typ av data. Datatypen kan till exempel vara heltal, datum eller text. En tabell i SQL Server kan relatera till en annan tabell genom s.k. främmande nycklar. En främmande nyckel är en kolumn i en tabell som innehåller värden som kan identifiera unika rader i samma eller en annan tabell. (Relationsdatabas, 2017)

Informationen i relationsdatabasen hanteras med hjälp av det standardiserade programmeringsspråket SQL, eller *Structured Query Language*.

3.2.8 Entity Framework

Entity Framework är ett s.k. ORM-system, eller *object-relational mapping*-system, utvecklat av Microsoft. Ett ORM-system är ett objektorienterat system som gör databashantering i applikationskod smidigare genom att konvertera databastabeller till klasser i kodbasen, eller klasser i kodbasen till databastabeller. Figur 12 visar Entity Framework i systemöverblicken.



Figur 12. Entity Framework i en systemöverblick.

Entity Framework underlättar också hämtningen av data från databasen genom att låta programmeraren skriva förfrågningar i LINQ. LINQ, eller *Language Integrated Query*, är en .NET-komponent som integrerar .NET-språken med databaser genom att automatiskt konvertera en förfrågan skriven i ett .NET-språk (såsom C#) till SQL-kod, och automatiskt instansiera objekt av resultatet som databasen returnerar.

3.2.9 AJAX

AJAX, eller Asynchronous JavaScript and XML, är en grupp teknologier som använder bland annat JavaScript för att skicka förfrågningar till webbservern och uppdatera webbsidans innehåll utan att ladda om hela sidan. (Ajax (programming), 2017)

3.2.10 Extensioner av DevExpress

DevExpress utvecklar, bland annat, UI-komponenter för ASP.NET MVC. Dessa komponenter bygger ut funktionaliteten på existerande HTML-komponenter, till exempel textboxar, listor och dropdown-menyer. Det finns också UI-komponenter som inte är extensioner av redan existerande HTML-komponenter. Ett exempel på en sådan är GridView, en komponent som visar data i tabellform med inbyggt stöd för att sortera, filtrera, gruppera och manipulera data. De flesta av dessa komponenter har inbyggt stöd för *callbacks* via AJAX, vilket innebär att de kan skicka en förfrågan till servern och uppdatera sitt innehåll utan att ladda om hela sidan. (ASP.NET MVC Extensions, 2017)

4 Arbetslistans implementation

Det här kapitlet beskriver förverkligandet av arbetslistan. Arbetslistans funktion och problemområde beskrevs i kapitel 2.

4.1 Teknisk infrastruktur

Det här underkapitlet beskriver arbetslistans implementation från en teknisk synvinkel.

4.1.1 Listan

Arbetslistan implementerades som en DevExpress GridView. GridView är en mycket anpassningsbar komponent vars grundfunktionalitet är att visa data i tabellform. Utöver grundfunktionaliteten finns funktionalitet för till exempel datamanipulering, -sortering, -gruppering, och -filtrering. GridView har också funktionalitet för att expandera rader och ladda in vyer. Denna expansionsfunktionalitet, kombinerat med komponentens anpassningsbara presentation, gjorde GridView till en ypperlig grund för arbetslistan. (Grid View Overview, 2017)

För att visa arbetsuppdrag i listan måste GridView:n ges data att visa. Datan ges i formen av en samling objekt, i det här fallet en samling av arbetsuppdragsobjekt av klassen *TaskViewModel* som har skapats från data i databasen. Klassen definierar textsträngar som är bundna till GridView:ns kolumner. Genom att lagra information om uppdraget i textsträngarna kan applikationen visa information relevant för en översiktsbild direkt i listan.

Som tidigare nämndes används GridView:ns inbyggda expansionsfunktionalitet för att expandera uppdrag. Expansionen fungerar genom att använda AJAX för att asynkront hämta en ny vy från Controller och visa den under den expanderade raden. För att ladda in rätt uppdragsvy skickas två argument med i callbacken: ett för uppdragstyp, som Controller använder för att anropa rätt metod i Model, och en unik uppdrags-id, som används i Model för att hitta rätt uppdrag i databasen.

4.1.2 Arbetsuppdrag

Eftersom varje uppdragstyp är unik och kräver olika data och olika uträkningar har varje uppdrag sin egen klass. Ett arbetsuppdragsobjekt fås från en statisk metod specifik till den arbetsuppdragstypen. I de fall där flera uppdragsklasser har egenskaper gemensamt används komposition, vilket innebär att egenskaper som hör ihop abstraheras till en egen klass, varav

arbetsuppdragen definierar medlemmar av. Kodexempel 7 visar klassen ”Common-TaskInfo”, vilken nästan alla uppdragsklasser innehåller.

Kodexempel 7. Klassen “CommonTaskInfo”.

```
public class CommonTaskInfo
{
    public History.ReceiptChangeReason ChangeReason { get; set; }
    public List<CommentViewModel> Comments { get; set; }
    public List<WorkerViewModel> Workers { get; set; }
    public SelectedCrateInfoViewModel SelectedCrate { get; set; }
    public CrateGroupInfoViewModel CrateGroupInfo { get; set; }
    public bool HighPriority { get; set; }
    public bool ShowNoPlacementFoundWarning { get; set; }
    public Guid ParentId { get; set; }
    public Guid TaskId { get; set; }
    public ProductionProgressInfoViewModel ProductionProgressInfo;
    public List<string> Loadnumbers { get; set; }
    public List<string> Messages = new List<string>();
    public bool HasSettings { get; set; }
}
```

I en förfrågan till Controller om en arbetsuppdragsvy ingår alltid en id för en viss uppdragstyp, som används för att hitta rätt metod. Varje uppdrags metod tar emot en unik uppdrags-id som argument och använder den för att hitta uppdraget i databasen.

Utifrån datan från databasen kan metoden ta flera vägar. Ta till exempel ett flyttuppdrag: när en användare för första gången öppnar uppdraget måste applikationen visa var lådorna som ska flyttas finns, och när användaren redan har lyft en låda måste applikationen visa var i lagret lådan ska placeras. För de här olika scenariorna behöver metoden kalla på andra metoder. I det första fallet måste metoden anropa en annan metod som hittar lagerplatser med relevanta lådor, och i det andra fallet måste metoden anropa en metod som ger en placeringsrekommendation.

Efter att Controller har fått ett arbetsuppdragsobjekt från Model skickas det vidare till View för att användas till att konstruera en ny vy. Även här används uppdragstypen, nu för att hitta rätt vy. Kodexempel 8 visar en del av Controller-metoden som har diskuterats.

Kodexempel 8. En metod i Controller.

```
public ActionResult TaskListExpand(Guid parentTaskId, TaskType taskType)
{
    switch (taskType)
    {
        case TaskType.Receipt:
            return PartialView("_TasklistReceiptExpanded", ReceiptTaskDataProvider.ExpandReceiptTask(parentTaskId));
        case TaskType.RoughSort:
            return PartialView("_TasklistRoughSortExpanded", RoughSortTaskDataProvider.ExpandRoughSortTask(parentTaskId));
        case TaskType.Move:
            return PartialView("_TasklistMoveExpanded", MoveTaskDataProvider.ExpandMoveTask(parentTaskId));
        case TaskType.MoveFromTransit:
```

I View konstrueras den vy som användaren kommer att se. Vyn är uppdelad i delvyer som kan användas i flera arbetsuppdrag, likt kompositionen i uppdragsklasserna. Kodexempel 9 visar koden för ett flyttuppdrags vy, där applikationen presenterar olika delvyer beroende på värdet av variabeln ”Model.UserHasCrateOnTruck”. På det här sättet kan vyn anpassas till var i arbetsprocessen användaren befinner sig. När användaren utför handlingar uppdateras informationen i databasen och vyn laddas in igen, uppdaterad i enlighet med det nya informationstillståndet.

Kodexempel 9. Ett flyttuppdrags View-fil.

```
@model Jepo.WarehouseManagement.Models.Tasklist.MoveTaskViewModel

<input hidden="hidden" id="detailRowExpanded" />
<div style="border:thin" id="expandedContainer">
    @Html.Partial("_TasklistExpandedCommon", Model.CommonInfo)
    <h4>@Tasklist.Work</h4>
    <div class="row">
        <div class="col-md-3">
            @Html.Partial("_TasklistMessages", Model.CommonInfo.Messages)
        </div>
    </div>
    <div class="row">
        @if (!Model.UserHasCrateOnTruck)
            {
                <div class="col-md-5">
                    @Html.Partial("_CbFromCell", Model.FromLocations)
                    <span id="errorSpan"></span>
                </div>
                <div class="col-md-2">
                    <button class="btn btn-lg mainScanAction" style="width:150px; height:60px"
                            onclick="StartMoveChildTask('@Model.CommonInfo.TaskId')">@Tasklist.MoveCrate</button>
                </div>
        }
        else
        {
            <div class="col-md-10">
                @Html.Partial("_CratePlacement", Model.CratePlacement)
            </div>
                    <div class="col-md-2 receiptBlink">
                        <div style="padding-top:34px; float:right">
                            <button style="width:150px; height:60px" class="btn btn-lg"
                                    onclick="MoveTaskCancelChild('@Model.CommonInfo.TaskId')">@Receipt.Cancel</button>
                        </div>
                    </div>

        }
    </div>
</div>
```

4.1.3 Databastabeller

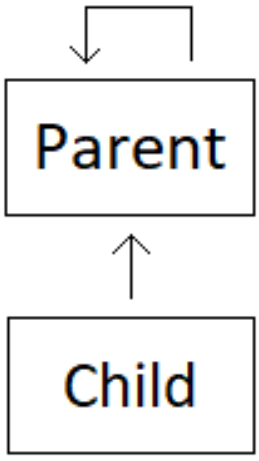
För att lagra information om arbetsuppdrag behövs databastabeller. För att förklara och samtidigt motivera databastabellernas struktur kommer det här kapitlet att diskutera hur arbetsprocesser grupperas som arbetsuppdrag, börjandes med lådflyttar.

För att kunna spara information om en enskild lådflytt behövs en databastabell. Man kan tänka sig att varje rad innehåller information om vilken låda som ska hämtas och till vilken avdelningen den ska föras.

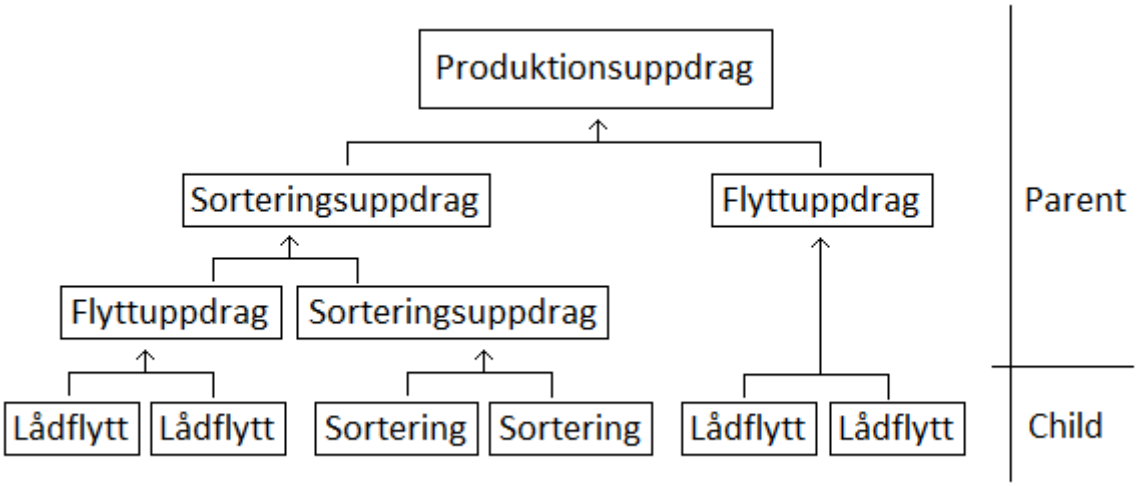
Det är sällsynt att endast en låda ska flyttas, oftast är det en samling lådor som berörs. Det innebär att man kan abstrahera och istället för att se arbetet som ett antal individuella lådflyttar, se det som en grupp lådflyttar. I tabellskapning speglas det här av att skapa en tabell till, där informationen som alla lådflyttar i en grupp har gemensamt sparas. Till exempel kan flyttens destination sparas på gruppnivå.

Det är på det här sättet arbetsuppdragens tabeller är uppbyggda. Ett arbetsuppdrag är en rad i en ”Parent”-tabell som innehåller gruppinformation, och de repeterande processerna i uppdraget, till exempel lådflyttar, existerar som rader i en ”Child”-tabell.

För vissa arbetsuppdrag är det användbart att definiera fler nivåer av gruppering. Till exempel när en grupp lådor ska storlekssorteras måste de först flyttas till sorteringsavdelningen. Sorteringen och flytten är då två individuella grupper med sina egna repeterande processer, men de kan också själva grupperas i en högre grupp. För att skapa en högre grupp i databasformat, och samtidigt tillåta för oändlig gruppering, har ”Parent”-tabellen en främmande nyckel till sig själv. Det här innebär att flera rader i ”Parent”-tabellen, till exempel ett flyttuppdrag och ett sorteringsuppdrag, kan peka på en gemensam ”Parent”-rad, som i sin tur kan peka på ännu en ”Parent”-rad, och så vidare. Figur 13 och figur 14 illustrerar den diskuterade strukturen.



Figur 13. En child-tabell pekar på en rekursiv Parent-tabell.



Figur 14- Ett exempel på hur informationen för ett produktionsuppdrag struktureras i mindre deluppdrag, som i sig kan innehålla deluppdrag.

4.2 Användargränssnitt

Det här underkapitlet beskriver arbetslistans implementation från en design-synvinkel.

4.2.1 Listans utseende och användarvänlighet

En GridViews utseende går att modifiera genom att använda GridView:ns egna inställningar. I kodexempel 10 visas hur en GridViews utseende modifieras.

Kodexempel 10. Koden för att modifiera en GridViews utseende.

```
@Html.DevExpress().GridView(settings =>
{
    settings.Images.DetailCollapsedButton.Url = "/Images/expand-icon.png";
    settings.Images.DetailCollapsedButton.Height = 45;
    settings.Images.DetailExpandedButton.Url = "/Images/collapse-icon.png";
    settings.Images.DetailExpandedButton.Height = 45;
    settings.Styles.Row.Font.Size = System.Web.UI.WebControls.FontUnit.Medium;
    settings.Width = System.Web.UI.WebControls.Unit.Percentage(100);
    settings.Styles.Cell.BorderRight.BorderStyle = System.Web.UI.WebControls.BorderStyle.None;
```

Eftersom användarna av arbetslistan använder pekskärm och ibland handskar är det viktigt att alla klickbara objekt är tillräckligt stora. För det här ändamålet är listans rader ungefär tre gånger större än hos en omodifierad GridView. Radtexten är uppdelad i tydliga kolumner med korta uppdragsbeskrivningar. När ett uppdrag expanderas används JavaScript för att kontrollera om hela det expanderade uppdraget är synligt på skärmen, och ifall det inte är det automatiskt skrolla fram uppdraget. Figur 15 visar arbetslistans färdiga utseende.

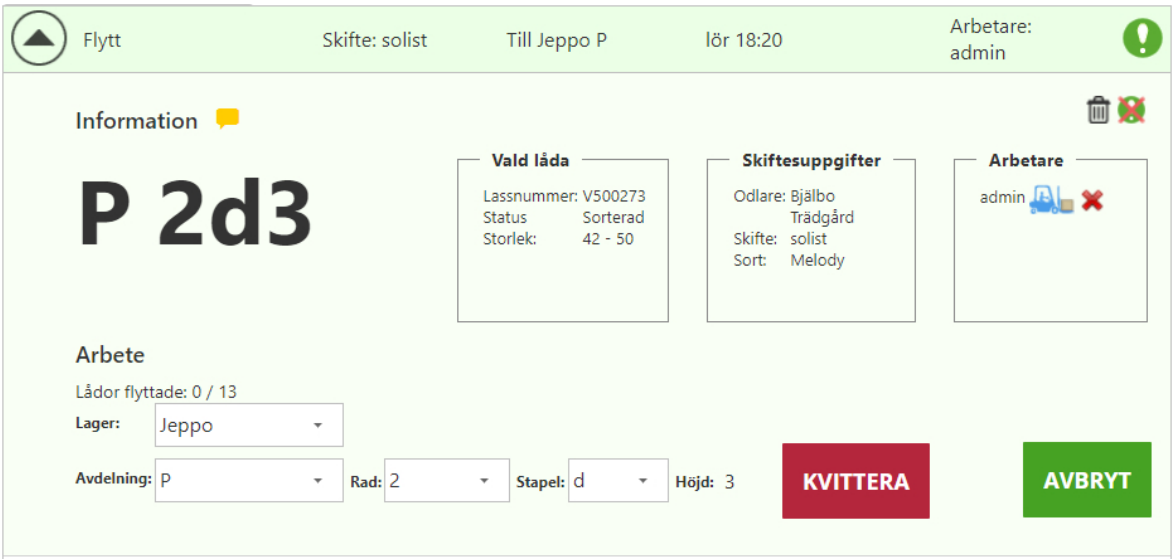
Sortering	Sort: Solist	Skifte:	fre 09:20	Arbetare: Ingen
Grovsortering JE23462	Dörr: Jeppo A, D1	Förare: koukku släp	fre 09:45	Arbetare: kalle
Produktionsflytt Jeppo G	3 Lådor kvar	Stormossen 1 A	4. Strimlor 6mm 6.0 (sön)	Arbetare: Ingen
Produktionsflytt Jeppo G	9 Lådor kvar	Löti a	4. Strimlor 6mm 6.0 (sön)	Arbetare: Ingen
Produktionsflytt Jeppo G	3 Lådor kvar		4. Strimlor 6mm 6.0 (sön)	Arbetare: Ingen

Figur 15. Arbetslistans utseende.

4.2.2 Arbetsuppdragets upplägg

Ett arbetsuppdrag är uppdelat i två huvuddelar: en informationsdel och en arbetsdel. Informationsdelens innehåll skiljer något från uppdrag till uppdrag, men allmänt innehåller den information om den valda lådan, gruppen som lådan tillhör, användare som jobbar på uppdraget, samt en kod som identifierar en cell. Cellkodens synlighet är viktig – den berättar snabbt åt användaren varifrån en låda ska hämtas eller vart den ska föras – därför visas cellkoden i ett stort teckensnitt.

Arbetsdelen av uppdraget skiljer sig också från uppdrag till uppdrag, men det som är gemensamt hos dem alla är att knapparna är stora och valen få och tydliga. Figur 16 visar ett exempel på ett flyttuppdrag. I uppdraget har användaren lyft en låda på trucken och är nu i placeringssteget.



Figur 16. Ett expanderat flyttuppdrag.

5 Produktionsordervyns implementation

Det här kapitlet beskriver förverkligandet av produktionsordervyn. Produktionsordervyns funktion och problemområde beskrevs i kapitel 2.

5.1 Teknisk lösning

Det här underkapitlet beskriver implementationen av produktionsordervyn ur en teknisk synvinkel.

5.1.1 Information om råmaterialets lämplighet

För att fylla en lagerkarta med information om vilka lådor är lämpliga för en produktionsorder krävs det först att informationen hämtas från databasen och struktureras på ett intuitivt sätt för syftet.

Exakta beskrivningar av hur lådors lämplighet för en produkt bedöms är utanför det här examensarbetets omfattning, men en kort beskrivning kan vara på sin plats: varje potatislass som tas in i lagret undergår en stickprovsanalys, där ett antal egenskaper hos potatisen (såsom storlek, form och sjukdomar) kvantifieras. Applikationen låter användaren definiera vilka egenskaper som är lämpliga eller olämpliga för en slutprodukt. Applikationen avgör lämplighetsgraden av ett lass för en slutprodukt genom att jämföra analysresultatet med produktens valda egenskaper.

Applikationen hämtar nämnda lämplighetsdata, tillsammans med information om varje låda i lagret och dess position (i rader, kolumner, och höjd). Utifrån den här informationen skapas objekt i klasserna ”CrateViewModel”, ”CellViewModel”, ”DepartmentViewModel”, och ”StoreMapsViewModel”. Som namnen antyder bildar de en sorts hierarki – ett Department-objekt innehåller en samling Cell-objekt, som alla innehåller en samling Crate-objekt. Genom att strukturera informationen på det här sättet (istället för att, till exempel, skapa en tre-dimensionell räcka av Crate-objekt) erhålls möjligheten att enkelt implementera funktionalitet på de olika nivåerna.

Den här samlingen objekt sparas i en sessionsvariabel. En webbserver hanterar varje förfrågan som en individuell förfrågan och sparar ingen information om variabler använda vid en tidigare förfrågan. Sessionsvariabler är bestående variabler knutna till en användarsession, som identifierar förfrågningar från samma webbläsare under en begränsad tid. Utan en sessionsvariabel hade applikationen varit tvungen att hämta lämplighetsdata från

databasen vid varje förfrågan. Genom att använda en sessionsvariabel minskas antalet databasanrop och på så vis också hanteringstiden för varje förfrågan, vilket leder till en snabbare upplevelse för användaren. (ASP.NET Session State Overview, 2017)

5.1.2 Användarinput

Applikationen använder AJAX för att sända information om användarens handlingar till servern. Controller tar emot förfrågan och anropar Model som gör förändringar i den sparade sessionsvariabeln. Controller svarar sedan på förfrågan med relevant information och AJAX uppdaterar vyn i användarens webbläsare. Användaren kan välja att spara sina ändringar, och då uppdateras informationen i databasen med informationen som har sparats i sessionsvariabeln.

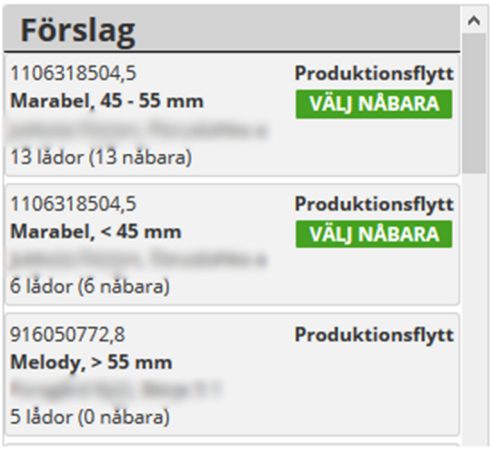
5.1.3 Lagerkartan

Lagerkartans utseende styrs med CSS-klasser. Rader och kolumner visas som en tabell. En cell på kartan består av ett antal div-taggar med varsin CSS-klass som styr allt dess utseende, bland annat storlek, teckensnitt och färg.

När lagerkartans vy ska konstrueras skickar Controllern cellobjekten från sessionsvariabeln till View. Klassen ”CellViewModel” definierar metoder som låter View veta hur många lådor cellen innehåller innehåller, lådornas lämplighet och lådornas reservationsstatus. Med den här informationen avgör View utseendet på cellen, vilket görs genom att ge cellerna olika CSS-klasser beroende på informationen.

5.1.4 Listan på reservationsförslag

De lämpliga lådorna grupperas på servern enligt odlingsskifte, och presenteras i en lista i användargränssnittet. Varje rad i listan visar gruppinformation (bland annat skifte, sort, storlek och lämplighetspoäng), innehåller en knapp för att reservera gruppen, och kan klickas på för att visa gruppen på lagerkartan. Gruppen visas på lagerkartan genom att, med hjälp av JavaScript, skrolla fram cellerna på kartan och ge dem en



Figur 17. Listans utseende

CSS-klass som färglägger deras bakgrund. Figur 17 visar listans utseende.

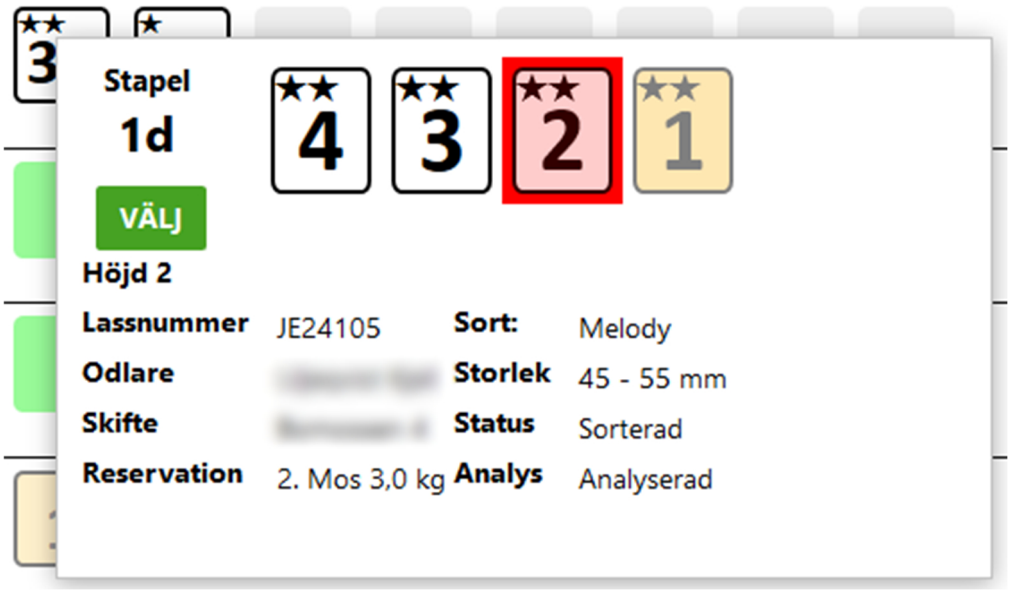
5.1.5 Individuell lådreservering och stapelinformation

För att reservera individuella lådor används en skild dialog som visas när användaren klickar på en cell. Dialogen är implementerad som en komponent av typen DevExpress Popup, ett popupfönster som enkelt kan visas och gömmas via JavaScript, och vars innehåll kan laddas och uppdateras via AJAX callbacks. För att göra användningen smidig öppnas dialogen precis vid användarens pekare och stängs när pekaren förs utanför dialogen. (Popup Control Overview, 2017)

En callback används när användaren först klickar på en cell för att hämta information om lådorna i cellen. I gränssnittet visas stapelns lådor grafiskt med samma symbolik som cellerna i lagerkartan. HTML-elementen för lådorna prenumererar på händelserna ”onmouseover” och ”onmouseout” (då pekaren förs över respektive lämnar lådans område) för att visa och gömma en tabell över den lådans egenskaper.

Reserveringen sker genom att användaren väljer de lådor som ska reserveras genom att klicka på dem, sedan klicka på en knapp som orsakar ett AJAX-anrop till servern, som uppdaterar sessionsvariabeln med det nya reservationsläget samt ritar om lagerkartan för att reflektera det nya läget. För att vidare underlätta reserveringen markeras nåbara lämpliga lådor automatiskt när användaren öppnar dialogen.

Figur 18 visar en stapeldialog där användaren håller pekaren över den andra lådan i stapeln.



Figur 18. Stapeldialogens utseende.

5.1.6 Automatisk reservering

Användaren kan klicka på en knapp för att automatiskt reservera lämpligt material för en order. Automatisk reservering sker genom att applikationen kontrollerar lämpligheten på alla raders yttersta låda och reserverar lådan med högst lämplighet. Det här repeteras tills råmaterialbehovet är uppfyllt.

5.1.7 Filtrering

Användaren presenteras med dropdown-menyer som har blivit fyllda med filtreringsalternativen. Filtreringsalternativen fås dynamiskt från samlingen lämpliga lådor, så att användaren endast kan välja bland existerande egenskaper. En filtrerad lådas lämplighet döljs för resten av applikationen, vilket orsakar en kedje-effekt så att all annan funktionalitet och presentation behandlar lådan som en olämplig låda.

5.1.8 Planens förverkligande

Efter att användaren har planerat produktionsdygnets råmaterialanvändning måste planen förverkligas. Det här sker genom arbetslistan. Applikationen grupperar de reserverade lådorna enligt slutprodukt och odlingsskifte, och skapar flyttuppdrag i arbetslistan för dessa. Flyttuppdragen berättar för truckförarna vilka lådor som ska tas till produktionslagret, och i vilken ordning det bör ske enligt den planerade produktionssekvensen.

5.2 Cellsymbolik

Det här underkapitlet beskriver symbolsystemet som används i cellerna på lagerkartan för att visa information relevant till lådreservering. Utvecklingen av symboliken skedde med hjälp av papper, penna och Microsoft Paint.

Bara genom att titta på en cell måste användaren snabbt kunna utläsa information om lådans lämplighet för vald produktionsorder, lådans tillgänglighet relaterad till vald orders sekvens i produktionsordningen samt reservationsstatus. Cellsymboliken måste också kunna meddela om det finns lådor av andra typer i stapeln.

5.2.1 Bakgrundsfärg visar reservationsstatus

Reservationsstatus visas med bakgrundsfärg. En olämplig och oreserverad låda är ointressant för användaren, och får då ingen speciell bakgrundsfärg, utan behåller den gulbruna lådfärgen som är bekant för användaren från andra lagerkartor.

En lämplig och oreserverad låda är intressant för användaren, och visas med en blå bakrundsfärg. Den blåa bakgrundsfärgen står i tydlig kontrast mot de gulbruna cellerna. När användaren reserverar lådan blir bakgrundsfärgen vit.

En låda som är reserverad till en slutprodukt som produceras tidigare än användarens valda slutprodukt visas med en genomskinlig gulbrun bakgrund. Genomskinligheten symboliserar att lådan inte kommer att finnas kvar vid tillfället då arbetet utförs, och att planeraren kan bortse från den. En låda som är reserverad till en slutprodukt som produceras senare än användarens slutprodukt visas med röd bakgrund.

5.2.2 Stjärnor visar lämplighet

En lådas lämplighet visas med stjärnor. En stjärna innebär en låg lämplighet och tre stjärnor en hög lämplighet. I figur 19 visas ett exempel på en cell med två stjärnor.

Figur 19. En lämplig cell.

5.2.3 Tre typer av siffror visar lådantal

I de fall där varje låda i stapeln kan representas av en bakgrundsfärg och ett stjärnantal visas endast en siffra för stapelns lådhöjd, kallad ”primärsiffra”.

I fall där det finns lämpliga lådor som inte kan representeras av cellens bakgrundsfärg och stjärnantal introduceras en ”sekundärsiffra”, som visas tillsammans med ett plustecken och antalet lämpliga dolda lådor i ett mindre teckensnitt än primärsiffran.

När det finns fler lådor i en stapel än vad primärsiffran visar introduceras också en ”totalsiffra”, som visas tillsammans med ett snedstreck-tecken (se figur 20). Figur 21 visar hur en lagerkarta i produktionsordervyn kan se ut.

Figur 20. En cell med total- och sekundärsiffra.

Figur 21. Lagerkartans utseende.

6 Resultat

Det här kapitlet beskriver slutresultatet av de båda modulerna, samt deras ibruktagning.

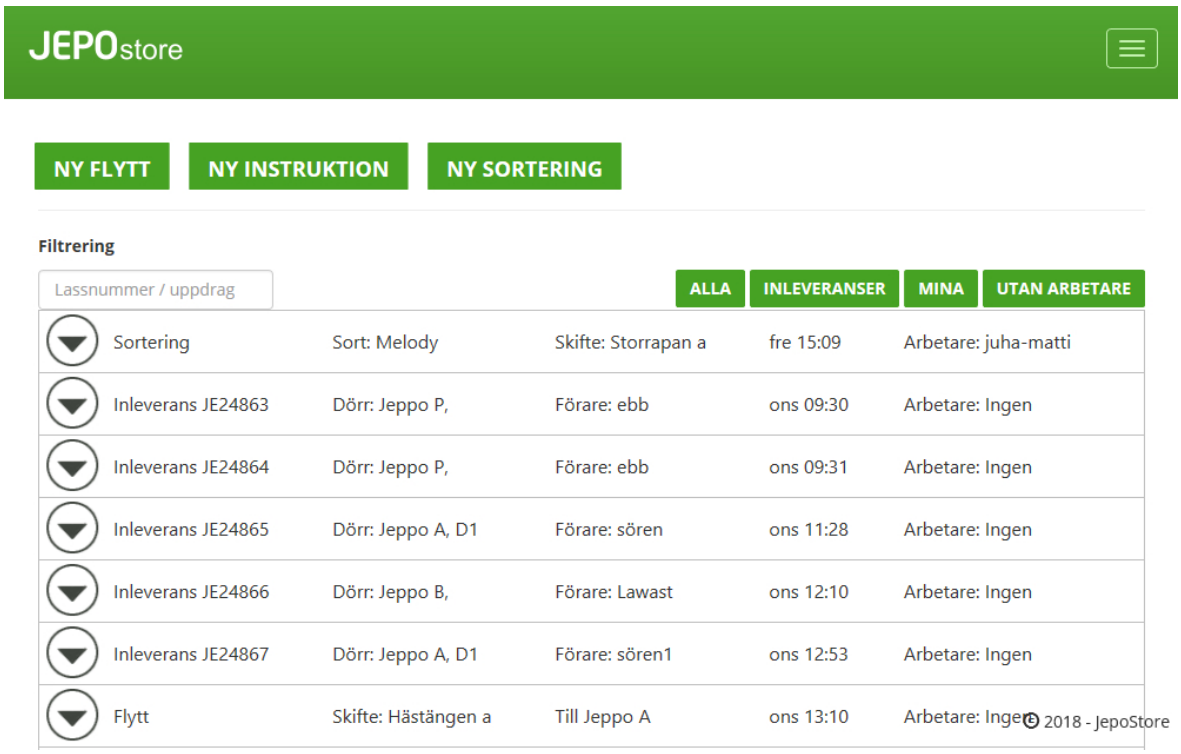
6.1 Arbetslistan

Målet med arbetslistan var att skapa ett system som samlade många olika arbetsprocessers gränssnitt på en plats. Syftet med detta var att förenkla navigationen i applikationen och samtidigt förse användarna med en översiktsbild över allt pågående arbete.

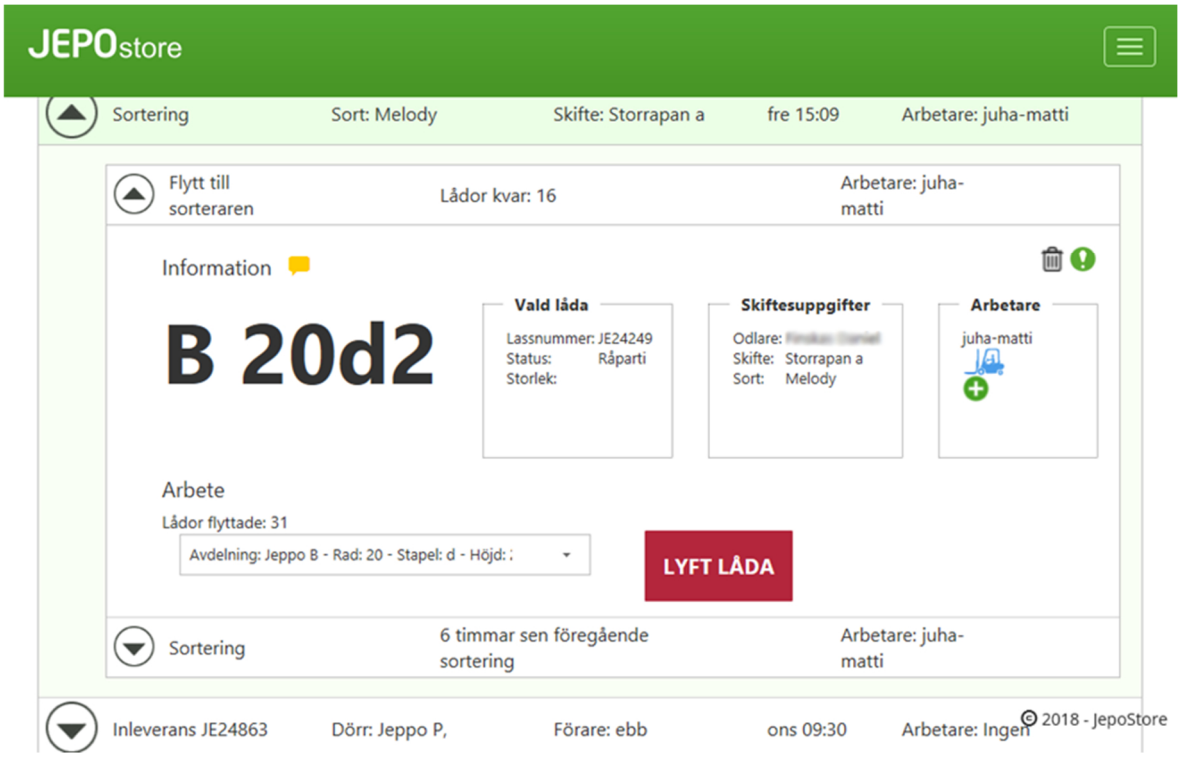
Arbetslistan implementerades i ASP.NET MVC-applikationen som en DevExpress GridView. Arbetslistans och arbetsuppdragens utseende anpassades för att kunna användas med pekskärm och arbetshandskar. GridView:ns rad-expanderingsfunktionalitet användes för att låta användaren expandera ett arbetsuppdrag i listan och presenteras med vyer specifika för ett visst arbetsskede i en viss arbetsprocess.

Resultatet blev en interaktiv lista, där användaren kan se allt pågående arbete i form av enskilda arbetsuppdrag, klicka på ett uppdrag, och genast få fram ett gränssnitt för arbetet. Det går också att styra arbetet i lagret genom arbetslistan. Det här sker både automatiskt, genom att applikationen skapar uppdrag vid olika händelser (till exempel vid en invägning av ett inkommande potatislass), och manuellt, genom att användare skapar nya arbetsuppdrag.

Figur 22 visar arbetslistan. Figur 23 visar ett expanderat sorteringsuppdrag.



Figur 22. Arbetslistan.



Figur 23. Ett expanderat sorteringsuppdrag.

6.2 Produktionsordervyn

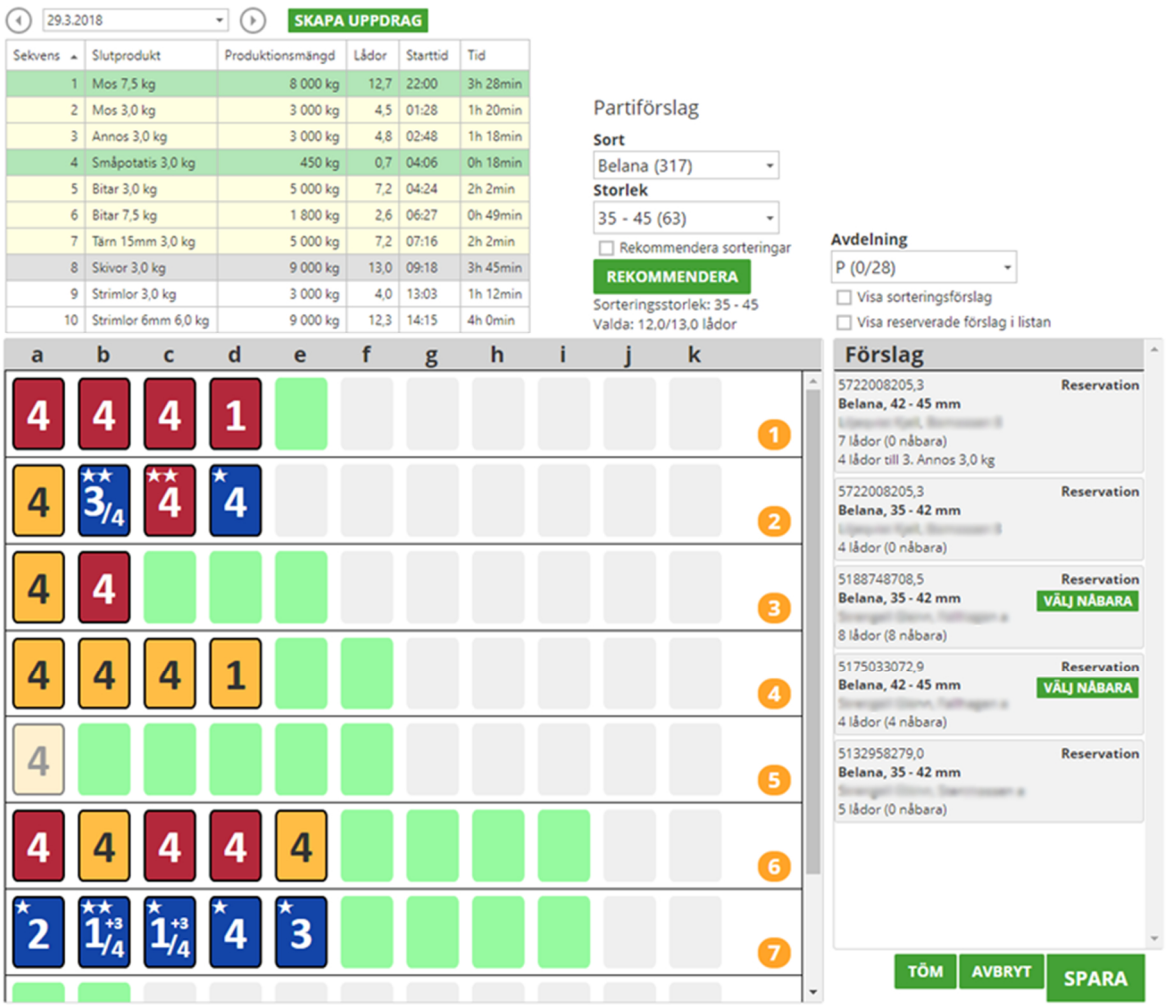
Målet med produktionsordervyn var att skapa ett hjälpmedel för planeringen av produktionens råmaterialskonsumtion. Systemets huvudsyfte skulle vara att visa

råmaterialets lämplighet och tillgänglighet, låta användaren reservera materialet och slutligen förmedla planen till truckförarna.

Produktionsordervyn implementerades med en interaktiv grafisk lagerkarta i center. Lagerkartan visar potatislådornas lämplighet och tillgänglighet med hjälp av färger och symboler, och användaren kan klicka på en lådstapel för att se detaljerad information och enkelt reservera individuella lådor för en produktionsorder.

För att vidare underlätta planerarens arbete implementerades gruppreservering i form av en interaktiv lista över lämpliga lådor grupperade enligt odlingsskifte, olika filtrerings-alternativ, och möjlighet att låta applikationen automatiskt reservera de lämpligaste tillgängliga lådorna.

När reserveringen är klar skapar applikationen de uppdrag i arbetslistan som behövs för att förverkliga planen. Tillsammans bildar då arbetslistan och produktionsordervyn ett helhetligt verktyg för planeringen och utförandet av arbetet kring en produktionsorders råmaterialskonsumtion. Figur 24 visar produktionsordervyn.



Figur 24. Produktionsordervyn.

6.3 Ibruktagning

Arbetslistan togs i bruk sommaren 2016, och produktionsorderplaneringsvyn vintern 2016-2017. Sedan dess har utvecklingen fortsatt med små förbättringar och buggfixar. Applikationen är nu i en utrullningsfas till fler verksamhetsplatser, vilket kommer att innebära mer utveckling i form av anpassningar för avvikande arbetsprocesser.

7 Diskussion

Projektet har varit utmanande och fyllt av möjligheter för nya kreativa lösningar. Jeppo Potatis har varit öppna för nya förslag och riktningar, och deras engagemang genom hela planeringen och utvecklingen har gjort arbetet till ett sant nöje.

Arbetet har varit mångsidigt och lärorikt. Jag har fått chansen att arbeta både med planering och implementering. I planeringen har ingått mycket kreativ problemlösning – vi har,

tillsammans med kunden, identifierat problem och behov, skapat mockups och diskuterat olika lösningar, och tagit beslut och satt upp mål. På implementationssidan har jag arbetat med ”hela stacken”, det vill säga med databasen, affärslogiken och presentationen.

Källförteckning

JEPO, 2018. [Online]
https://www.jepo.fi
[Använd 27 3 2018].

ASP.NET MVC, 2017. [Online]
https://en.wikipedia.org/wiki/ASP.NET_MVC
[Använd 16 8 2017].

ASP.NET Razor, 2017. [Online]
https://en.wikipedia.org/wiki/ASP.NET_Razor
[Använd 16 8 2017].

HTML, 2017. [Online]
https://en.wikipedia.org/wiki/HTML
[Använd 29 10 2017].

Cascading Style Sheets, 2017. [Online]
https://en.wikipedia.org/wiki/Cascading_Style_Sheets
[Använd 19 8 2017].

JavaScript, 2017. [Online]
https://en.wikipedia.org/wiki/JavaScript
[Använd 19 8 2017].

Ajax (programming), 2017. [Online]
https://en.wikipedia.org/wiki/Ajax_(programming)
[Använd 19 8 2017].

C# (programming language), 2017. [Online]
https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
[Använd 24 8 2017].

JavaScript, 2017. [Online]
https://en.wikipedia.org/wiki/JavaScript
[Använd 24 8 2017].

Objektorienterad programmering, 2017. [Online]
https://sv.wikipedia.org/wiki/Objektorienterad_programmering
[Använd 30 8 2017].

Goodreads, 2017. [Online]
https://www.goodreads.com/book/show/386162.The_Hitchhiker_s_Guide_to_the_Galaxy
[Använd 30 8 2017].

ASP.NET MVC Extensions. [Online]
https://documentation.devexpress.com/AspNet/7896/ASP-NET-MVC-Extensions
[Använd 30 8 2017].

ASP.NET Session State Overview. [Online]
https://msdn.microsoft.com/en-us/library/ms178581(v=vs.100).aspx
[Använd 16 9 2017].

Microsoft SQL Server. [Online]
https://en.wikipedia.org/wiki/Microsoft_SQL_Server
[Använd 17 9 2017].

Relational database management system. [Online]
https://en.wikipedia.org/wiki/Relational_database_management_system
[Använd 18 9 2017].

Relationsdatabas. [Online]
https://sv.wikipedia.org/wiki/Relationsdatabas
[Använd 20 9 2017].

SQL-injektion. [Online]
https://sv.wikipedia.org/wiki/SQL-injektion
[Använd 21 9 2017].

Grid View Overview. [Online]
https://demos.devexpress.com/MVCxGridViewDemos/Overview
[Använd 22 9 2017].

Popup Control Overview. [Online]
https://demos.devexpress.com/MVCxDockAndPopupsDemos/PopupControl/Overview
[Använd 30 9 2017].