

Product Information Management (PIM) System Rapport



cphbusiness

COPENHAGEN BUSINESS ACADEMY

Frederik Braagaard

Claus Findinge

Malthe Woschek

Gruppe 5

December 15, 2019

Indhold

1	Indledning	1
1.1	Baggrund	1
1.2	Teknologi valg	2
2	Krav	3
3	SCRUM Userstories	6
4	Domænemodel & ER Diagram	9
5	Navigationsdiagram	11
6	Særlige Forhold	12
6.1	Hvordan vi håndterer brugerinput validering.	12
6.2	Hvordan vi håndterer exceptions.	13
6.3	Hvordan vores GoToCommands fungerer.	15
6.4	Upload & Download.	15
7	Udvalgte Kodeeksempler	16
8	Status på Implementation	19
9	Tests	21
10	Process	24
10.1	Arbejdsprocessen Faktuelt	24
10.2	Arbejdsprocessen Reflekteret	25
11	Konklusion	26
12	Bilag	27
12.1	Sekvens Diagram	27
12.2	Klasse Diagram	27

Nyttige Links & Info

GitHub Repository: <https://github.com/Bringordie/PIM>

Projekt: <http://206.81.17.88:8080/PIM-1.0-SNAPSHOT/>

Excel ark med produkter: <https://github.com/Bringordie/PIM/blob/master/upload%20product%20files/newversion.xlsx>

JSON fil med produkter: <https://github.com/Bringordie/PIM/blob/master/upload%20product%20files/programlanguage.json>

User Story Backlog via Taiga.io:

<https://tree.taiga.io/project/bringordie-product-information-management-pim/backlog>

Daglige SCRUM-møde noter:

<https://docs.google.com/document/d/17stbRMmR3RXcKN1R8iMehUy0eP7UDkvr307m9x0E4HE/edit>

SCRUM-Planning noter:

https://docs.google.com/document/d/1x0YDSYMPvRlHiaLzg3gS6r4YRtIY2zd5_9I2P5CBOHg/edit

Frederik Braagaard: cph-fb87@cphbusiness.dk

Claus Findinge: cph-cf124@cphbusiness.dk

Malthe Woschek: cph-mw202@cphbusiness.dk

1 Indledning

1.1 Baggrund

- Claus Findinge

Fødevarergrossist- og leverandør Inco Cash & Carry ønsker, at vi udvikler et PIM-system (Product Information System), der kan håndtere kompleksiteten i produkt-data og hjælpe virksomheden med at finde de rigtige informationer og få dem ud på de rigtige markedsføringskanaler. Virksomheden ønsker med andre ord et system, der kan spare dem tid og ressourcer i forhold til at finde og arbejde med relevant data, samtidig med at de kan være sikre på, at den pågældende data også er den ønskede data.

Et PIM-system er i bredere forstand et system, der strukturerer og visualiserer et firmas produkter. Fordelen ved et PIM-system er bl. a., at det kan hjælpe et firma internt i forhold til at arbejde effektivt med firmaets produkter.

I det gældende projekt har vi sigtet efter at lave et system, der på let og overskuelig vis hjælper brugeren (evt. Incos ansatte) med at finde informationer på specifikke produkter fra databasen samt kunne tilføje og slette produkter. Derudover skal det også være muligt at ændre i informationerne på et bestemt produkt (evt. et produkts pris) samt kunne tilgå en komplet liste af informationer på udvalgte produkter. Da vi vælger at inddele hvert produkt i både under- og overkategorier, skal det også her være muligt at tilføje, ændre og slette disse kategorier.

Meningen med systemet er altså at hjælpe Inco til en bedre intern arbejdsgang, hvor det er lettere og mere sikkert at tilgå data på virksomhedens produkter.

1.2 Teknologi valg

- Frederik Braagaard

Vi bruger en Droplet fra <https://cloud.digitalocean.com>

Netbeans version 8.2 (Java EE) <https://netbeans.org/downloads/8.2>

Da vi bruger 8.2 Netbeans bruger vi den nyeste opdatering af Tomcat 8.5:
<https://tomcat.apache.org/download-80.cgi>

Github projektet ligger på <https://github.com/Bringordie/>

Vi bruger JUnit version 1.4.1, gson 2.8.5 til at konvertere en .json fil til et array, commons-fileupload 1.4 og commons-io som bliver tilføjet igennem fileupload til at uploade/downloade filer til vores server og Jacoco 0.8.1 til at se vores codecoverage for alle vores tests i Netbeans.

Til at se vores SQL data lokalt bruger vi den nyeste version af MySQL Workbench <https://www.mysql.com/downloads/>

For at vise charts på vores Tomcat server har vi valgt at bruge <https://canvasjs.com/> da de havde kode eksempler og det var nemt at sætte op og bruge.

Til at sortere vores tabeller har vi brugt <https://kryogenix.org/code/browser/sorttable/> da det også var nemt og lige til at få til at virke.

Til formatering af hele rapporten har vi valgt at bruge L^AT_EX via TeX Maker 5.0.3:
<https://www.xmlmath.net/texmaker/>

2 Krav

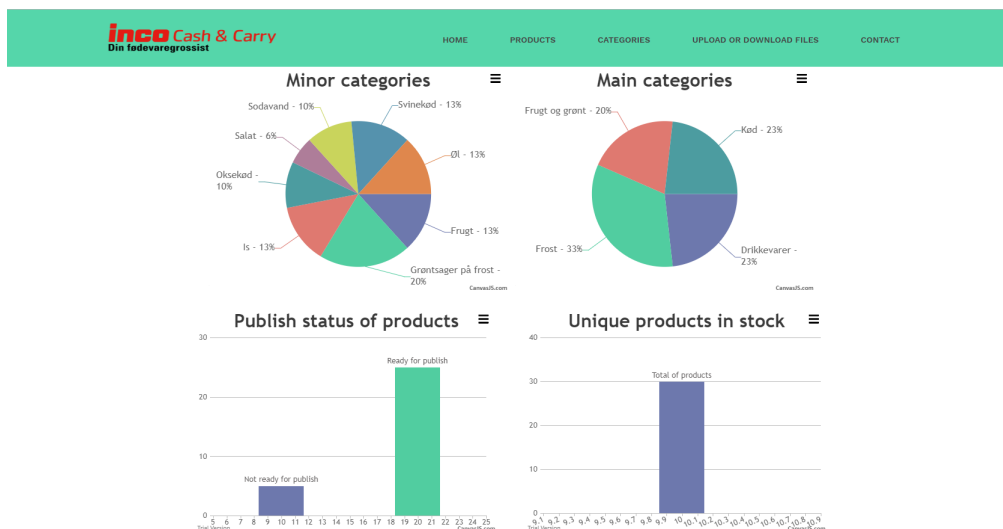
- Claus Findinge

Vi har med afsæt i de fire sprintmøder, vi har haft med Product Owner løbende udarbejdet 20 userstories, der dækker de ønsker og forventninger, Product Owner har haft til det færdige produkt. Product Owner er i den forbindelse kommet med ønsker til systemet og samtidig har udvalgt de userstories, vi har arbejdet med. Vores userstories ses i Screen shottet nedenfor:

Votes	User Stories	Status	Points ▾
<input type="checkbox"/>	▲ 0 #34 CSS and Design	Done ▾	?
<input type="checkbox"/>	▲ 0 #35 Add to droplet	Done ▾	?
<input type="checkbox"/>	▲ 0 #29 Add link between a minor and a main category on all pages.	Done ▾	?
<input type="checkbox"/>	▲ 0 #33 Charts on the home page	Done ▾	?
<input type="checkbox"/>	▲ 0 #22 Logger	Archived ▾	?
<input type="checkbox"/>	▲ 0 #1 Login page	Archived ▾	?
<input type="checkbox"/>	▲ 0 #11 Creation of user	Archived ▾	?
<input type="checkbox"/>	▲ 0 #18 Kategorier i alfabetisk og fejlhåndtering	Done ▾	?
<input type="checkbox"/>	▲ 0 #14 Search function	Done ▾	?
<input type="checkbox"/>	▲ 0 #13 Admin: Upload pictures	Done ▾	?
<input type="checkbox"/>	▲ 0 #21 Error handling	Done ▾	?
<input type="checkbox"/>	▲ 0 #23 Bulk editing of attributes	Done ▾	?
<input type="checkbox"/>	▲ 0 #20 View all products	Done ▾	?
<input type="checkbox"/>	▲ 0 #24 Export of database	Done ▾	?
<input type="checkbox"/>	▲ 0 #19 CSS and Design Mockup	Done ▾	?
<input type="checkbox"/>	▲ 0 #9 Admin: Add/Edit/Delete product	Done ▾	?
<input type="checkbox"/>	▲ 0 #2 Admin: Upload JSON file	Done ▾	?
<input type="checkbox"/>	▲ 0 #3 Admin: Upload Excel sheet	Done ▾	?
<input type="checkbox"/>	▲ 0 #7 JSP Header	Done ▾	?
<input type="checkbox"/>	▲ 0 #16 Admin: Add/Edit/Delete categories	Done ▾	?

Af de 20 userstories er vi nået i mål med de 17 af dem. Det gælder først og fremmest ønsket om at kunne tilføje, slette og ændre i produkter i databasen samt at kunne tilføje og slette kategorier til produkterne. Kategorierne er den forbindelse blevet organiseret i alfabetisk rækkefølge, så de er lettere at holde styr på. Derudover har vi fået lavet en søgeknop, så det er muligt at søge på bestemte produkter ved at indtaste søgeord (eller dele af søgeord) og tilpasse søgeordene bestemte attributter (f. eks. id eller navn). Det er i den forbindelse muligt at sortere listen af produkter enten i alfabetisk eller numerisk rækkefølge på de givne attributter (trykker man

eksempelvis på name, sorteres navnene alfabetisk ud fra deres navn, og hvis man trykker på id-knappen, sorteres produkterne numerisk ud fra id-nummer). Vi har i forlængelse heraf fået lavet en bulk edit funktion, så det er muligt at ændre i flere produkter ad gangen (f. eks. hvis tre produkter med forskellige priser skal have samme pris). Derudover har vi gjort det muligt for brugeren at tilføje JSON-filer og Excel-filer, så vedkommende kan tilføje flere produkter ad gangen, ligesom vi har gjort det muligt at tilføje billeder til produkterne. Det er også gjort muligt at eksportere produktinformation fra databasen, så denne kan deles. I forhold til forsiden har vi fået lavet en header, der repræsenterer Inco Cash & Carrys logo, ligesom der vises noget status på kategorierne og produkterne i databasen (bl. a. hvor mange produkter der er gjort offentlige og hvor mange, der ikke er). Vi har i tilgift forsøgt at lave et enkelt og brugervenligt CSS-design, så brugeren kan føle sig komfortabel med siden og ikke mister overblikket. Sidst men ikke mindst er vi nået i mål med at lave fejlhåndtering på flere af funktionerne, så brugeren får at vide, hvis noget er mislykket.



Der er også blevet lavet et mock-up, så Product Owner har haft mulighed for at vurdere, hvordan slutproduktet er kommet til at se ud, allerede inden vi færdiggjorde systemet. De tre userstories, vi ikke er nået i hus med, gælder ønsket om en logger, en loginside og i forlængelse heraf muligheden for at oprette en bruger, der kan logge ind. Tiden (og vores mandefald fra fire til tre i gruppen) taget i betragtning, er det hen mod slutningen af projektet blevet aftalt med Product Owner, at vi ikke skulle starte på disse userstories.

Alt i alt kan man sige, at det PIM-system vi har udarbejdet, hjælper Inco Cash & Carry i forhold til at kunne danne sig et bedre overblik over de produkter, de har på lager, ligesom det letter arbejdet i forhold til at tilføje og slette produkter. Med vores opdeling i søgeord og søgekriterier bliver virksomheden i stand til let at indhente informationer på specifikke produkter selv i store databaser. Ligeledes hjælper vores hensyn til sortering (bl. a. i alfabetisk rækkefølge) virksomheden i forhold til at kunne danne sig et generelt overblik over produkterne i databasen. Virksomheden vil i øvrigt været godt selvhjulpet i forhold til at tilføje, slette og redigere i informationer samt at uploade billeder. Det bedre overblik vil i sidste ende kunne hjælpe virksomheden i forhold til at undgå fejl i data (og hurtigt rette op på produkter med fejl i data) og dermed fremstå troværdig og professionel overfor kunder og samarbejdspartnere. Tilsvarende vil det lette arbejdet i forhold til leverancer, da man let og sikkert vil kunne tjekke lageret på specifikke varer. En bedre og mere sikker tilgængelighed til informationer på virksomhedens produkter, vil også betyde en mere effektiv og hurtig arbejdsgang, der i sidste ende vil gavne virksomhedens produktivitet og salg. Alt i alt vil PIM-systemet hjælpe Inco Cash & Carry i forhold til værdier som troværdighed, effektiv arbejdsgang, overblik, tilgængelighed, vedligeholdelse af database og i sidste ende også mersalg.

3 SCRUM Userstories

- Malthe Woschek

I samarbejde med 'Product Owner' (PO), er der i alt blevet skrevet 20 userstories. I dette afsnit har vi valgt tre userstories fra vores backlog, som vi mener var de mest interessante at skrive om, og har derved uddybet dem nedenfor. Hele vores produkt backlog kan ses via følgende link: <https://tree.taiga.io/project/bringordie-product-information-management-pim/backlog>

En af de første userstories der blev skrevet og lavet, er funktionaliteten til at tilføje, redigere og slette kategorier fra en database. Selve userstory'en er beskrevet på følgende måde:

User Story - "Add/Edit/Delete Categories"

As a user, I want to be able to add, edit and delete categories of products, so that products are more organized and can be filtered.

Estimate: XL

Tasks:

- Create SQL db
- Create SQL main and minor category tables with unique IDs + category names
- Function to add a new categories to DB.
- Function to delete an existing categories from DB.
- Function to edit an existing categories from DB.
- Tests

Requirements:

- When all tasks are completed.
- When it is possible to show an html site, where it is possible to add edit or delete a categories.

Det skal nævnes, at vi for "Tasks" i nogle userstories kun har valgt at skrive de største og vigtigste opgaver ned, for at undgå redundans og overflødig tekst. Kun ved enkelte userstories som indeholdte opgaver vi aldrig havde lavet før uddybede vi og skrev nogle flere trin under "Tasks".

Dét der gør denne userstory særlig, udover dens åbenlyse formål, er at der ved to ud af de seks tasks skal der oprettes en MySQL Database for hele projektet samt en tabel der indeholder både et navn og et unikt ID for hver over-kategori (main category) og under-kategori (minor category). Den oprettede database kommer vi også til at bruge for mange andre userstories, så dermed er fundamentet for resten af projektet også sat op. Da det er PO'en som skal vurdere, udvælge og prioritere

de enkelte userstories, mente vi ikke at selve oprettelsen af en database kunne være en userstory for sig selv, da en PO højst sandsynligvis ikke ville kende noget til databaser og samtidigt undervurdere vigtigheden af en database.

Med hensyn til vores estimer har vi valgt at lave en blanding mellem en relativ og absolut form for estimering. Dette vil sige, at vores estimer står i S, M, L, XL osv. men har stadig et tal af minutter og timer knyttet til sig. F.eks. er $S = 30$ minutter, $M = 2.5$ timer, $L = 5$ timer og hvert X er 5 timer ekstra. Dog er selv de tilknyttede tal stadig relative og kan variere afhængigt af de enkelte opgaver.

Én af de større userstories vi havde var funktionen til at kunne se eksisterende produkter i databasen fra en brugerplatform, og samtidigt sortere dem f.eks. fra A-Z eller fra laveste pris til højeste. Dette har vi beskrevet på følgende måde:

User Story - “View All Products”

As a user, I want to be able to see all products and their attributes with a picture, either by searching for a product ID or browse through categories, so that finding products can be done in multiple ways and so that the list of products is filtered to be more concise.

Additionally, I want to be able to sort viewed products by clicking on product names/product ID/etc., so that they are shown in descending or ascending order.

Estimate: XXL

Task:

- Function to show all product from DB.
- Function to match image string to a picture.
- Necessary JSP pages containing the list of products.
- Basic CSS and JS for the table of products on the HTML page.
- Tests

Requirements:

- When all tasks are completed.
- To show an html site, where it is possible to see all product.

Denne userstory (kaldet ”View All Products”) er særlig, da den bl.a. fungerer som en bro for to andre JSP sider og fordi at den indeholder det største mængde af information for brugeren. Resultatet på denne userstory er en JSP side der viser alle oprettede produkter inde i en tabel, samt hver attribut for hvert produkt. Herfra har brugeren mulighed til at trykke på hvert enkelt produkt, for at se det tilknyttede produktbillede i større format og læse mere omkring produktbeskrivelsen. Man har

også mulighed for at vælge flere forskellige produkter og derefter gå ind på en side der hedder 'Bulk Edit Products', der tillader brugeren til at redigerer alle produktinformationer, bortset fra produktets ID, for alle udvalgte produkter på én gang. Her opdagede vi, at man tit er nødt til at tænke forud i forhold til hvordan brugeren skal have adgang til de forskellige funktionaliteter. I eksemplet nævnt ovenfor, kan brugeren netop kun komme ind til 'Bulk Edit Products' hvis brugeren går ind på 'View All Products' først og vælger mindst ét produkt at redigere. Derudover gik det op for os, at man bare i én userstory kan komme ud for at skulle arbejde både med backend dele som opretter og henter en liste af vores eksisterende produkter, men også frontend dele såsom generel CSS og JavaScript, bare for at få produkterne vist på en acceptabel måde.

En tredje vigtig userstory, der også blev lavet som noget af det første, var adgang til en form for header i toppen af alle JSP sider. Dette skulle indeholder de vigtigste links/knapper til de forskellige funktioner og derved tillader brugeren at navigere nemmere rundt på siderne. Vores userstory til dette ser således ud:

User Story - "JSP Header"

As a user, I want to have a header with the most commonly used buttons, that is consistent across multiple web-pages, so that navigating commonly used web-pages is easier.

Estimate: M-L

Task:

- Create .JSP file (and css file)
- HREF

Requirements:

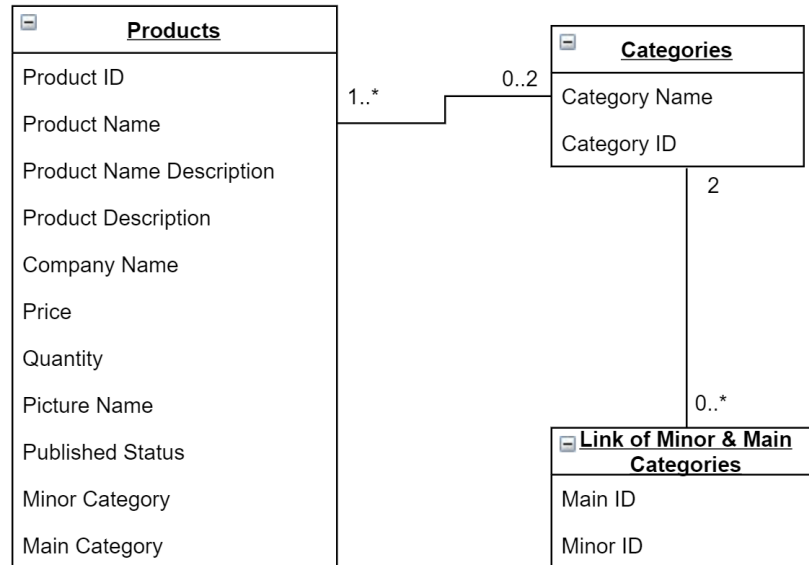
- Completed all tasks.
- Able to show a demo to the product owner.

Denne userstory var på ingen måde den største eller sværeste at lave, men alligevel var dét, at have en header til PIM systemet, essentielt både for brugeren, men også for os til f.eks. at foretage manuelle tests eller lave frontend relateret implementeringer. At kunne navigere hurtigt og nemt rundt på en side er ofte taget for givet, indtil man for tredje gang skal trykke sig igennem fem forskellige sider, bare for at endelig komme hen på det man oprindeligt ville.

4 Domænemodel & ER Diagram

- Malthe Woschek

Domænemodel

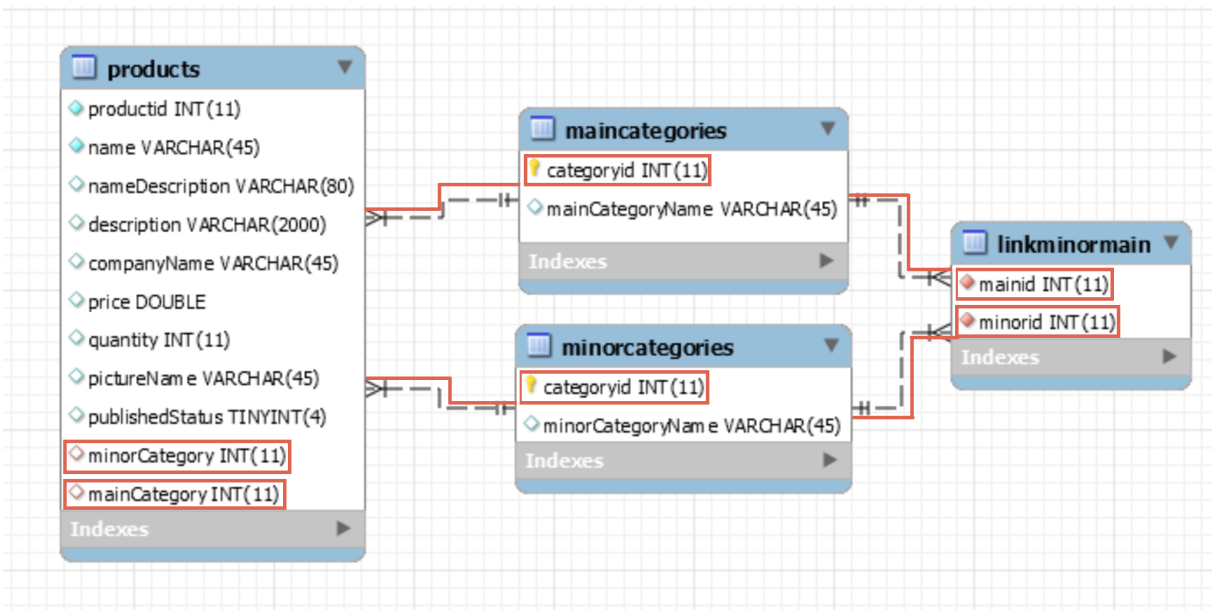


På domænemodellen vist ovenfor har vi tre forskellige logik klasser med diverse informationer fra vores database. I selve databasen er “Categories” klassen delt op i hver deres “Main Categories” og “Minor Categories” klasse. Men da de netop er så ens og for at gøre det lettere at forstå for f.eks. køberen af vores projekt, har vi valgt at samle dem i én klasse. I øvrigt har vi også en tredje klasse ved navn “Link of Minor & Main Categories”, som vi inkluderer med i domænemodellen, da dette bliver gjort i vores facade.

Sådan som vores relationer hænger sammen, har vi ét eller mange produkter (Products) der kan have ingen kategorier eller én overkategori (Main Category)/underkategori (Minor Category) eller både en over- og underkategori. F.eks. kan en flaske Coca-Cola godt have underkategorien “Sodavand” men ingen overkategori og omvendt, eller ingen kategorier overhovedet. Tilgængæld, hvis et produkt ikke har nogen eller kun har én kategori knyttet til sig, bliver produktets “Published Status” sat til “False”. Dette vil sige, at produktet ikke har alle nødvendige felter udfyldt som er påkrævet for, at produktet kan blive offentliggjort på en hjemmeside for kunden.

Derudover har vi en klasse, som kun indeholder unikke forbindelser mellem en over- og underkategori (Link of Minor & Main Categories). Denne klasse syntes vi også var relevant at inkludere, da den også ligger i vores facade. Med dette menes der, at vi for hvert produkt, som har både en over- og underkategori, laver en ny, unik forbindelse imellem de to kategorier og sætter dem ind i klassen, hvis ikke de allerede findes. Dermed siger vi at en over- og underkategori enten kan have ingen forbindelser eller mange.

ER Diagram

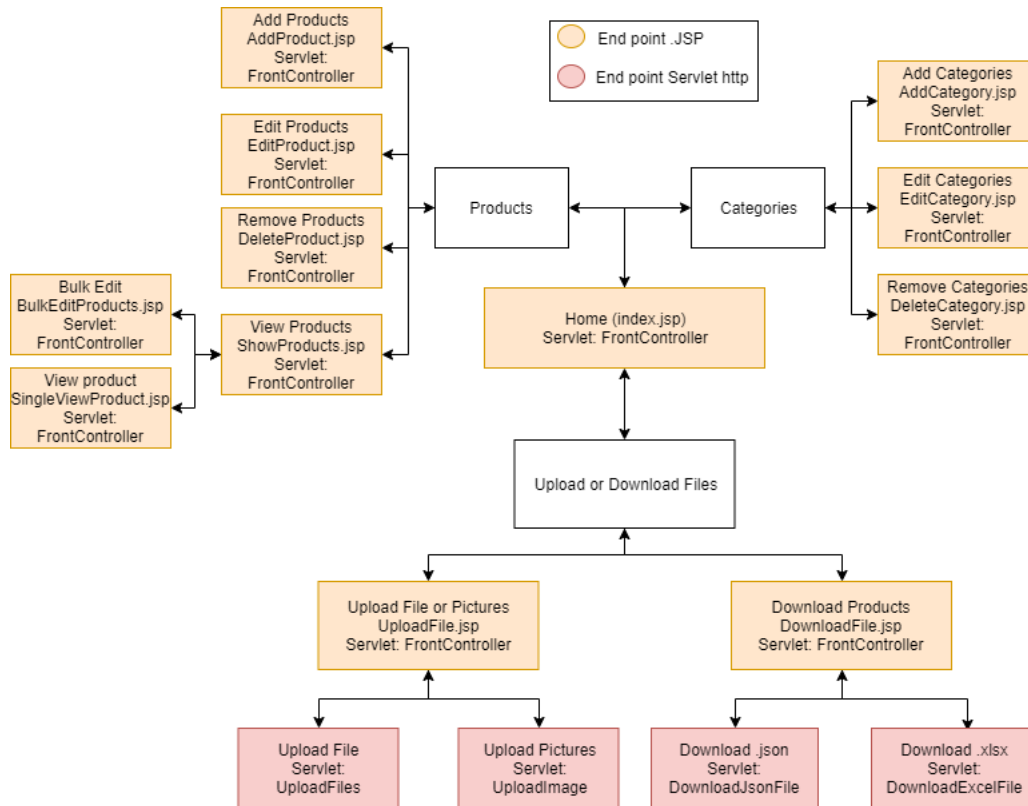


I vores ER-diagram vist ovenfor har vi fire tabeller i vores database. I modsætning til domænemodellen, viser vi med vores kategori-tabeller, at de er splittet op i to, bl.a. for at tydeliggøre deres relationer til de andre tabeller.

Udover de sædvanlige information pointerer vi her, at vi som 'primary keys' har categoryid under main- og minorcategories. Products tabellen tager en 'foreign key' af begge categoryid'er fra main- og minorcategories tabellerne som vi i products tabellen har kaldt for minorCategory og mainCategory. Tilligemed er mainid og minorid under linkminormain tabellen også 'foreign keys' af begge categoryid primary keys. Dette har vi illustreret med røde streger på ER-diagrammet ovenfor, samt et gult nøgle-ikon på de informationer som er primary keys.

5 Navigationsdiagram

- Frederik Braagaard



Vi har lavet en header som bliver vist på alle sider undtagen når man har uploadet en fil eller billede.

Vi bruger som standard FrontControlleren til at navigerer os frem til sider og for at vise indhold på siderne.

Som standard har vi lavet "gotoCommands" som er en command som henter information til siden når vi skal besøge den og navigerer os til den bestemte .jsp side. Alle andre commands bliver brugt til at eksekverer en opgave hvor den giver et resultat tilbage som bliver vist på den samme .jsp side. Ved Upload bruger vi UploadFiles servlet og ved Download bruger vi Download.JsonFile/Download.ExcelFile.

6 Særlige Forhold

- Frederik Braagaard

6.1 Hvordan vi håndterer brugerinput validering.

Som standard håndterer vi vores brugerinput igennem input felterne. Dette betyder at ved alle (med undtagelse med Product Description) har vi lavet felterne required så at man ikke kan klikke submit uden at have angivet noget i input feltet, dette er både gjort vores egen skyld og brugerens for at undgå exception, DB eller bruger fejl.

```
<input type="text" id="companyname" name="CompanyName"
      value="" required />
```

Ved nummer input har vi valgt at fjerne "e" som er et "tal" og derved kan blive indtastet i input feltet. Dette er både for vores egen skyld og brugerens så at der ikke kommer eventuelle fejl ved en fejltastning. Ved tal inputs har vi pris som er en double og kvantitet som er int. Så vi har lavet at input feltet ikke kan være negativt og derved 0 eller højere. Ellers har vi lavet "step=0.01" hvilket vil sige at en double værdi maksimum kan have 2 decimaler.

```
<li class="form-row">
<label for="price">Add Price:</label>
<input type="number" id="price" min="0" step="0.01"
      name="Price" value="" required onkeydown="return
      event.keyCode !== 69" step=".01"/>
</li>
<li class="form-row">
<label for="quantity">Add Quantity:</label>
<input type="number" id="quantity" min="0"
      name="Quantity" value="" required onkeydown="return
      event.keyCode !== 69" step=""/>
</li>
```

6.2 Hvordan vi håndterer exceptions.

Vi har prøvet på at lave vores input felterne så gode at vi teknisk set aldrig skulle få en exception fejl. Nogen små steder har vi dog taget det i betragtning at det kan ske. Til at starte ud kan vi forklare hvordan vi har valgt at en fejl bliver vist for brugeren. Som eksempel kan vi tage "Add Product". Det første vi gør er at ved headeren klikke at vi vil gå til add product. Derved bliver GoToAddProductCommand valgt igennem vores command. Ved denne command tjekker vi først om der er main og minor kategorier i vores database. Derefter Sætter vi en setAttribute på:

```
request.getSession().setAttribute("returnproductvalue",  
    "empty");
```

Dette er gjort for at vi ikke får en nullpointer exception. Men det er også gjort, så at hvis vi opretter et produkt og får en besked at produktet er oprettet, at når vi så engang går tilbage igen til Add Product at vi så stadig ikke ser denne besked 10 timer senere. Valget bag at vi laver vores validering med Strings er fordi vi i starten ville gøre det med Boolean men det skabte problemer at parse vores object eller validere informationerne. Så da vi skulle have mindst 3 fejl beskeder valgte vi at vælge at give en specifik streng for at skrive en besked til brugeren.

På vores .jsp side bliver vores besked vist på denne måde:

```
<% if (session.getAttribute("returnproductvalue") ==  
    null) {  
} else if (session.getAttribute("returnproductvalue")  
    .toString().equals("productadded")) { %>  
<h2>Product created!</h2>  
<%} else if (session.getAttribute("returnproductvalue")  
    .toString().equals("alreadyexists")) {%>  
<h2>Unable to execute request: Product with this ID  
    already exists.</h2>  
<%} else {}%>
```

Set i bakspejlet kunne vi godt havde fået dette på færre linje i jsp siden ved at lave valideringen på command siden.

Når en bruger så har oprettet et produkt går vi igennem AddProductCommand. Vi sætter vores DB kald til en streng for at holde return værdien

```
String returnvalue = db.addProduct(products,
    "/db.properties");
```

og derefter henter vi så sessionen og sætter attributen til retur værdien vi har fået.

```
request.getSession().setAttribute("returnproductvalue",
    returnvalue);
```

Til lige at forklare hvordan der bliver gjort nede i vores ProductMapper så tjekker vi først om et produkt med dette ID allerede findes:

```
Boolean booleanIDCheck =
    checkIfProductExists(String.valueOf(product.getId()),
        propertyname);
```

Og hvis det ikke findes allerede oprettes produktet og sætter return værdien til,

```
returnvalue = "productadded";
```

hvis det allerede findes,

```
returnvalue = "alreadyexists";
```

eller hvis noget skulle gå galt

```
returnvalue = "something went wrong";
```

6.3 Hvordan vores GoToCommands fungerer.

Fra start af blev de bare brugt til at gå over til en jsp side. Vi fandt dog hurtigt ud af de blev nød til at kunne en del mere. Så, hvad en GoToCommand faktisk gør er at initierer siden. Om det er alle produkter der skal hentes i databasen for at vise en side med alle produkter, eller bare lave en setAttribute på at vi ikke får en nullpointer exception på siden. Så vores GoToCommands er vores væg der gør at vores kode virker og kan blive vist. Så hvis en bruger fik lyst til at skifte cmd=gotoViewSingleProduct&selected=12345 til et produkt der ikke findes vil siden bare være tom, dette er noget som gerne ville havde fikset, men med den tid vi havde valgte vi ikke at fokusere på andre ting.

6.4 Upload & Download.

Upload:

1	ProductID	ProductName	ProductNameDescription	ProductDescription	CompanyName	Price	Quantity	PictureName	MinorCategory	MainCategory
2	111	Solsikkeskud øko.	25 g / Danmark / Klasse 1	Økologiske solsikkeskud fra Mille's Mikrogrønt.	Mille's Mikrogrøn	25	600	solsikkeskud-oek	Salat	Økologisk

Hvad angår upload, både ved .json og excel, virker det kun på denne måde, at vi har filens indhold i denne rækkefølge: ProductID, ProductName, ProductNameDescription, ProductDescription, CompanyName, Price, Quantity, PictureName, MinorCategory og MainCategory. Som sådan er det ikke nødvendigt at disse hovedtitler bliver placeret i excel-dokumentet. Dog bliver første linje så nødt til at være tom, da vi springer denne linje over, så den ikke bliver læst. Denne form-struktur er nødvendig for, at vores program kan læse filerne rigtigt. Ved .json skal filen også være i dette pattern.

Download:

Ved download vil excel dokumentet kunne blive uploaded igen med det samme til vores server. Dette virker dog ikke for .json filen da ved skrivningen af filen komprimeres for at have den mindste filstørrelse størrelse.

7 Udvalgte Kodeeksempler

- Frederik Braagaard

Vores udvalgte kodeeksempel er det eksempel vi har valgt i [Sekvens-Diagram picture] som vi vil forklare hvordan fungerer. Vi har en servlet ved navn UploadFiles. Når man er inde på UploadFile.jsp og har valgt en .xlsx eller .json fil til upload, vil UploadFiles begynde at oprette en temp fil i vores path som vi har skrevet i vores filepath.property fil.

```
InputStream sa = UploadFiles.class.getResourceAsStream("/filepath.properties");
```

Når det så er gjort kalder vi metoden

```
uploadToDB(fileNameTest);
```

I denne metode tjekker vi på det filnavn vi har givet om den indeholder .json eller .xlsx

```
public void uploadToDB(String fileNameTest) throws
    ClassNotFoundException, FileNotFoundException,
    SQLException, IOException {
    DBFacade db = new DBFacade();
    if (fileNameTest.contains(".json")) {
        JsonHandler handler = new JsonHandler();
        ArrayList<Products> s =
            handler.makeJSONFileIntoArray(fileNameTest);
        db.jsonInsertOrUpdateToDB(s, "/db.properties");
    } else if (fileNameTest.contains(".xlsx") ||
        fileNameTest.contains(".xml")) {
        ExcelHandler excelhandler = new ExcelHandler();
        ArrayList<Products> products = new ArrayList();
        products =
            excelhandler.extractInfo(fileNameTest);
        db.excelInsertOrUpdateToDB(products,
            "/db.properties");
    }
}
```

Når det så er gjort tjekker vi ved excelInsertOrUpdateToDB først om produktet findes i forvejen. Hvis det allerede findes vil vi opdatere produktet ellers vil oprette et nyt.

```

for (Products products : list) {
    ProductID = products.getId();
    //BOOLEAN CHECK
    Boolean booleanIDCheck =
        checkIfProductExists(String.valueOf(ProductID),
            propertyname);
    if (booleanIDCheck == false) {

```

Det første vi gør i for each loopet at hente gettere på ArrayListen til at assigne værdierne til attributterne vi vil tilføje til vores DB. Ved main og minor kategori laver vi et kald til getMinorValuesFromDBFile og getMainValuesFromDBFile hvis vi ikke kan finde et kategori navn opretter vi det og tager ID'et fra kategorien og tilføjer det til vores produkt. Hvis det ikke findes opretter vi kategorien og igen tilføjer ID'et til produktet.

```

//Checking minorvalue
if (products.getMinorCategory().contains("empty")) {
    //do nothing
} else if (category.getMinorValuesFromDBFile
    (products.getMinorCategory(), propertyname) == 0) {
    int minorIDCreated = category.createMinorIDInDB
    (products.getMinorCategory(), propertyname);
    MinorCategory = minorIDCreated;
} else if (category.getMinorValuesFromDBFile
    (products.getMinorCategory(), propertyname) > 0) {
    int reuseIDCreated =
        category.getMinorValuesFromDBFile
    (products.getMinorCategory(), propertyname);
    MinorCategory = reuseIDCreated;
}

```

Som standard har vi sagt at et produkt som minimum skal have et produkt name, product name description, product description og et company name. Dette tjekker vi ved at se om en af attributterne er null. Hvis en af dem er null så sætter vi published status til false så produktet evt. ikke ville blive vist på en shop. Hvis alle kravene er opfyldt vil det blive sat til true.

```
//Requirements for publishing
if (ProductName == null || ProductNameDescription
== null || ProductDescription == null ||
CompanyName == null || MinorCategory == 0 ||
MainCategory == 0) {PublishedStatus = false;
} else {
ublishedStatus = true;
}
```

Når dette er tjekket opretter/opdateres produktet i DB. Til slut i oprettelsen laver vi et kald til vores CategoryMapper for at tjekke eller oprette et link imellem main og minor kategorien.

```
category.checkOrCreateLinkminormain(MainCategory ,
    MinorCategory , propertyname);
```

Når alle produkter så er blevet kørt igennem kommer vi tilbage til UploadFiles servletten som sletter vores temp fil og udskriver til brugeren om alt gik godt eller om der er sket en fejl.

```
Files.deleteIfExists(Paths.get(path+fi.getName()));

out.println("Uploaded Filename: " + fileName + " has been
    uploaded to the server" + "<br>");
```

8 Status på Implementation

- Malthe Woschek og Frederik Braagaard

Ud af vores 20 userstories er der tre, som vi af forskellige grunde, ikke fik nået at implementere ind i vores projekt. Disse tre lyder som følgende:

- Oprettelse af en bruger med en bestemt rolle.
- En loginside for brugere
- En 'logger' som kan notere en fejlmeddelelse + et ID









Én af de største grunde til at de ikke blev implementeret eller blev prioriteret lavt, er fordi deres nødvendige eksistens kunne diskuteres. F.eks. ville muligheden for at kunne oprette brugere med specifikke roller ikke være det mest nødvendige for mindre virksomheder at have. Dét og logger'en er ting der er "nice-to-have" og ikke "must-have", og af logger har vi jo faktisk også allerede en. Nemlig vores Catalina server hvor en person nemt kan finde programmerings fejlen ud fra et tidspunkt og evt hvilken side fejlen skete på. Derudover er der også andre ting, som vi gerne ville have haft med, og fejl som vi gerne ville have fikset såsom:

- Generel CSS-finpudsning. F.eks. at putte input felterne ind i grå kasser, eller at have et mere responsivt design.
- editMainCategory og editMinorCategory mangler begge at have checkMainCategory og checkMinorCategory metoden implementeret ind.
- 100% tests ved alt der kunne integrations testes.
- Selenium tests.
- Bedre fejlhåndtering.
- Fikse catches til at ramme bedre / bredere.
- Ingen mulighed for blank pages.
- Omskrivning af metoder for at mindske kode linjer.
- Refaktorering af metoder og klasser for bedre forståelse.

- Cleanup, fjerne redundans.
- Bedre javadoc/dokumentation.
- Mere Javascript.
- Contact i headeren til faktisk at vise noget FAQ og kontakt info.
- Baggrund billede / Baggrundsfarve på alle .jsp sider.
- UTF-8 (ÆØÅ) fejl ved diverse ting.
- At lave JSON handler om.

9 Tests

- Frederik Braagaard

Total Coverage: <div><div style="width: 48.90%;"></div></div> 48,90 %			
Filename	Coverage	Total	Not Executed
 persistence.DBFacade	<div><div style="width: 100.00%;"></div></div> 100,00 %	37	0
 logic.Categories	<div><div style="width: 100.00%;"></div></div> 100,00 %	7	0
 logic.Products	<div><div style="width: 100.00%;"></div></div> 100,00 %	35	0
 logic.JsonHandler	<div><div style="width: 100.00%;"></div></div> 100,00 %	9	0
 logic.ExcelHandler	<div><div style="width: 94.87%;"></div></div> 94,87 %	78	4
 persistence.ProductMapper	<div><div style="width: 87.70%;"></div></div> 87,70 %	439	54
 persistence.CategoryMapper	<div><div style="width: 85.41%;"></div></div> 85,41 %	233	34
 persistence.DBConnection	<div><div style="width: 80.00%;"></div></div> 80,00 %	15	3
Total	<div><div style="width: 48.90%;"></div></div> 48,90 %	1550	792

Af tests har vi både lavet manuelle tests for at se om programmet virkede oppe på Tomcat serveren, men vi har også lavet JUnit tests. Hvilket vil sige at vi ikke har lavet selenium tests og har derved ikke fået lavet automatiske tests på servlet klasser. Ved logic har vi lavet 100% coverage på Categories, Products og JsonHandler. Ellers har vi 94.87% ved ExcelHandler hvor der er 2 exceptions som der ikke er blevet testet.

Ved ProductMapper, CategoryMapper og DBConnection har vi 87.70%, 85.41% og 80%. Alle metoder herunder er blevet testet, vi har dog ikke fået testet exceptions da det ville være et stort arbejde at lave alle metoder korrekte, til at throwe den korrekte fejl som man kunne teste på. Vi har ikke lavet tests på JsonWriter, FileReaderLogic eller ExcelWriter da vi har hardcoded ind at vi læser fra en properties fil ind. Der kunne godt have lavet JUnit tests på dem men det valgte vi ikke ville have en konflikt med prod og test.

Hvordan vi har opbygget vores tests: Ved DBtests har vi som start angivet alle metoder efter vores klasse så alle vores metoder kan bruge disse metoder, både for at det ser kønnere ud, men også for at undgå programmerings fejl.

```
DBFacadeExtra dbextra = new DBFacadeExtra();
DBFacade dbfacade = new DBFacade();
ProductMapper productmapper = new ProductMapper();
CategoryMapper categorymapper = new CategoryMapper();
JsonHandler jsonhandler = new JsonHandler();
ExcelHandler excelhandler = new ExcelHandler();
String DBPROPERTYTEST = "/dbtest.properties";
```



```
DBConnection connection = new DBConnection();
```

Vi har lavet en @Before som cleaner vores database hver gang før en DBtest bliver tested for at vi er sikre at vores tests kan blive kørt 1000 gange uden at et problem pludselig opstår. Så vi dropper alle tabeller og opretter dem igen så den er "tom".

```
@Before
    public void setUp() throws SQLException,
        ClassNotFoundException, IOException {
        dbextra.droptable(DBPROPERTYTEST);
        dbextra.createtable(DBPROPERTYTEST);
    }
```

En ekstra smart ting vi har gjort er at vi har lavet en getCustomDataFromDB.

```
public String getCustomDataFromDB(String s) throws
    ClassNotFoundException, SQLException, IOException {
    String output = "";
    String sql = s;
    ResultSet result =
        getConnection("dbtest.properties").
        prepareStatement(sql).executeQuery();

    try {
        while (result.next()) {
            output = result.getString(1);
        }
    } catch (SQLException ex) {
        System.out.println(ex);
    }
    return output;
}
```

Det smarte ved denne metode er at vi ikke behøver at lave 100 metoder for at få et metode kald vi kan asserte på. Vi kan skrive hvad som helst ind. For eksempel kan vi tage

```
@Test
    public void uploadOfJsonUpload() throws SQLException,
        ClassNotFoundException, IOException,
        FileNotFoundException {
```

```

String fileName =
    "src/test/java/files/singledata.json";
ArrayList<Products> product =
    jsonhandler.makeJsonFileIntoArray(fileName);
dbfacade.jsonInsertOrUpdateToDB(product,
    DBPROPERTYTEST);

int expected = 1;
String dbcall =
    dbextra.getCustomDataFromDB("select
    COUNT(name) from products;");

assertEquals(expected, Integer.parseInt(dbcall));
}

```

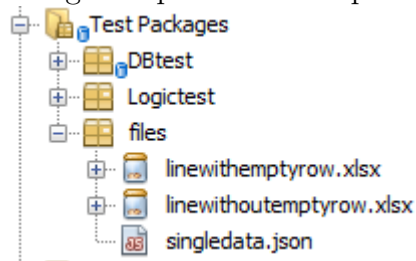
Vi uploader en fil 4 gange. Vores upload er lavet så at hvis produkt ID'et allerede findes så vil den ikke blive oprettet igen men opdateret. Så derved kan vi så sige

```
String dbcall = dbextra.getCustomDataFromDB("select
COUNT(name) from products;");
```

og asserte dette

```
assertEquals(expected, Integer.parseInt(dbcall));
```

En anden ting vi har gjort er at sætte filerne vi laver tests på ind i selve vores projekt og ikke på vores "computer".



Dette er gjort så vi kan nemt alle i gruppen eller andre ude fra ikke vil få en sti fejl når projektet bygges eller laver tests.

```
String fileName = "src/test/java/files/singledata.json";
```

10 Process

- Malthe Woschek

10.1 Arbejdsprocessen Faktuelt

I løbet af dette projekt havde vi fire SCRUM-sprints, inklusiv fire SCRUM Review og Planning møder med PO'en. Dette gav os fire uger af hverdage, hvilket betød at vi ligeledes fordelte rollen som SCRUM-master imellem os. På den måde fik alle i gruppen oplevet, hvad det betyder at være SCRUM-master og fik dermed også en dybere forståelse af denne rolle. Rækkefølgen, af hvem der skulle være SCRUM master i hvilke uger, blev tilfældigt fastlagt og er som følgende: Malthe, Casper, Frederik og Claus. De daglige standup SCRUM møder holdte vi altid om morgenen når vi mødtes i hverdagene. I disse møder fortalte vi hver især hvad vi fik/ikke fik opnået dagen før og om nogen havde brug for hjælp fra de andre i gruppen, imens ugens SCRUM-master noterede det ned på et online dokument delt imellem os. I de tilfælde, hvor der var nogen der spurgte om hjælp, lavede vi 'pair programmering', hvilket betyder at man sidder to mennesker på én computer, hvor den ene skriver kode mens den anden tjekker det igennem. Dette skiftes man til i hyppige intervaller.

I bund og grund var der ikke én SCRUM-master, som gjorde noget særligt ekstraordinært i forhold til alle andre i løbet af deres uge. Dog satte vi altid stor pris på at SCRUM-master'en f.eks. noterede alle spørgsmål vi havde til PO imellem hvert SCRUM Review og derefter ligge både spørgsmålene og svarene fra PO op på et online dokument, så alle i gruppen havde en chance for at se, hvad der blev svaret og på hvilke spørgsmål.

Retrospektive møder blev altid holdt mellem vores SCRUM Review møde og SCRUM Planning møde med PO. Generelt har vi alle været glade og positiv overrasket over den effekt, et møde fokuseret på ens tidligere præstation som gruppe, kan have på fremtidigt samarbejde. Dette var især gjort tydeligt, da vi efter første sprint stadig havde nogle userstories der enten ikke blev helt færdige og/eller skulle tilbage i backloggen, ved andet sprint havde nået næsten alle userstories bortset fra 1 og efter tredje sprint havde nået alle userstories, hvor de også alle blev accepteret af PO.

10.2 Arbejdsprocessen Reflekteret

I de retrospektive møder kiggede vi tilbage på hvordan vi havde lavet det vi lavede, hvor godt eller dårligt vi syntes det var og hvordan vi kunne forbedre os i fremtiden. Derudover snakkede vi om, hvad der blev sagt til review-møderne og lavede overvejelser til, hvilke ekstra ting der muligvis kunne effektivisere vores arbejdsproces eller hjælpe med at undgå uheldige misforståelser og fejl i fremtiden. For eksempel aftalte vi, at hvis ikke man kunne nå at lave en bestemt opgave inden for et aftalt tidsrum, så skulle man sige det til de andre i gruppen, så de havde en chance for at tage opgaven til sig i stedet. Tilligemed lavede vi også en tabel på et delt online dokument, som viste hvilket tidspunkt vi hver dag aftalte at mødes, netop for at undgå misforståelser, forsinkelser eller lignende.

Vores estimeringer har ikke altid været helt korrekte. Dog mener vi, at vi efter de første par sprints har været rimelige gode til at estimere mere præcist på de efterfølgende userstories. F.eks. fandt vi ud af, at det ikke altid er nemt at pludselig tage længere sprints i betragtning når man skal estimere, eftersom man før har haft kortere sprints eller at der dukker uforudsete forhindringer op, som man ved andre projekter ikke havde. Herforuden skal det også nævnes, at vi startede projektet som en gruppe på fire mennesker, men efter anden uge mistede én, af private årsager. Dette har yderligere resulteret i estimeringer der ikke har været lige så nøjagtige som håbet.

I sidste ende føler vi alle i gruppen, at review, planning, standup og de retrospektive møder har haft en positiv indflydelse på vores generelle produktivitet og forståelse for hinandens arbejdstilgange. Dét at tage tid fra programmering og i stedet fokusere på, hvor godt gruppen egentlig programmerer, forbedrer simpelthen kvaliteten af selve arbejdet og understøtter langvarigt teamwork.

11 Konklusion

I vores arbejdsperiode har vi lært og mærket, hvordan et SCRUM-team kan fungere. Dette både på godt og ondt. Vi har i vores SCRUM-periode fra start sat et højt mål med, hvad vi ønskede at nå. Eftersom vi mistede et gruppemedlem halvvejs inde i processen, blev vi udfordrede og var mere opmærksomme på, hvem der havde brug for en ekstra hånd. Vurderet på, at vi er tre i gruppen, er vi meget tilfredse med det, vi har nået, og vi synes, at vi har fået opfyldt vores målsætning.

Vi har lavet et funktionsdygtigt PIM-system, hvor brugeren på en let og designvenlig måde kan navigere rundt i.

Alt i alt har vi nået langt hovedparten af de opgaver, vi er blevet stillet. Hvis vi havde haft tiden til det, ville vi dog gerne have lavet noget mere eller lave nogle ting om.¹

Vi kan konkludere, at vi i de sidste SCRUM-sprints fik godt hånd om, hvordan man benytter SCRUM. Vi har tillige til næste gang lært, hvad der kunne forbedres og gøres anderledes fra start af.

¹Læs afsnittet om status på implementering.

12 Bilag

12.1 Sekvens Diagram

Da vores Sekvens Diagram er for stort til at vise på en fornuftig måde her i rapporten, vil vi i stedet give et link direkte til vores diagram via GitHub:

<https://github.com/Bringordie/PIM/blob/master/sekvensdiagram.png>

12.2 Klasse Diagram

Da vores Klasse Diagram er for stort til at vise på en fornuftig måde her i rapporten, vil vi i stedet give et link direkte til vores diagram via GitHub:

<https://github.com/Bringordie/PIM/blob/master/classdiagram.png>

Vi ville stærkt anbefale, at downloade denne fil ned på ens egen computer, for at få et bedre overblik over hele diagrammet.