Fog Carporte

Et projekt for Fog Trælast & Byggecenter

Sebastian Klitte Egeberg - cph-se164@cphbusiness.dk- Github: Sebbedeb
Lasse Baggesgård Hansen - cph-lh479@cphbusiness.dk - Github: kotteletfisk
Oskar Daniel Olsen - cph-oo221@cphbusiness.dk - Github: cph-oo221

hold_A_gruppe_4

01. - 31. Maj 2023

Github repository for opgaven: https://github.com/cph-oo221/Dat2-Eksamensprojekt

Indhold

Links

De følgende links til henholdsvis GitHub repository, demovideo, deployed website og Javadocs documentation:

- Link til GitHub repository af webapplikationen:

- https://github.com/cph-oo221/Dat2-Eksamensprojekt

- Link til demovideo:

- https://youtu.be/1ujnQZeDJ3k

- Link til kørende version af website på Digital Ocean Droplet

- http://64.226.113.14:8080/Dat2-Eksamensprojekt-1.0
    - Legitimationsoplysninger for admin:
        - Login email: admin@admin.com
        - Kodeord: admin

- Javadocs documentation: https://drive.google.com/file/d/14e127072CCTbpPznIotdNZp-rZ0oPXT2/view?usp=share_link (unzip og åben index.jsp)

Indledning

Projektet Fog Carporte er en besvarelse og implementering af Flow 5: Semesterprojekt - Eksamensprojekt - Fog Carport på 2. semester for datamatiker-studerende på CPHBusiness afdeling Lyngby. Projektet handler om at udvide og forbedre den allerede eksisterende webshop på Fog's hjemmeside (https://www.johannesfog.dk/). Udvidelserne og forbedringerne omfatter især muligheden for at designe en skræddersyet carport, men også administratorens muligheder for bl.a. ændringer af varelager. Opgavens krav er baseret på et besøg hos kunden, udfra hvilket vi i gruppen har brygget 8 user-stories, og diverse funktionelle- og ikke-funktionelle krav.
Projektet skal bygges med følgende teknologier: Java, MySQL, HTML, CSS, Twitter Bootstrap, Tomcat webcontainer og en Digital Ocean Droplet. Det er forventet at man ved projektets afslutning har en funktionel webshop der kører på en remote webcontainer.
Derudover har Fog ønsket muligheden for at kunne generere .scad filer til 3D-print af en kundes skræddersyede carport.
På de følgende sider vil vi, gruppe 4 fra hold A, forsøge at besvare den skriftlige del af Fog Carporte Projektet.

Baggrund

Baggrunden for projektet er Fog Trælast & Byggecenter's ønske om forbedring og udvidelse af deres webshop. På webshoppen skal kunderne kunne designe og bestille skræddersyede carporte med valgfri mål, med muligheden for at tilvælge et skur og/eller trapeztag. Fog har ønsket så meget mulig kundekontakt i denne proces, da det er hvad der adskiller Fog fra konkurrenterne.

For at kunne benytte sig af webshoppen skal kunderne oprette en profil, der bliver gemt i virksomhedens database. Databasen skal desuden indeholde de forskellige typer metal og træ, Fog har på varelager, samt priser, mål mm.

På administratorsiden af systemet, skal Fog have mulighed for følgende:

- at se og redigere i varelageret, herunder ændre priser og tilføje nye varer.
- at se alle ordrer i databasen.
- at kunne godkende, afvise og give tilbud på ordrer lagt af kunder.

De ovenstående ønsker fra Fog er udspecificeret yderligere i form af funktionelle- og ikke-funktionelle krav. Kravene vil ikke inkluderes som baggrund, men der kan læses yderligere om dem i afsnittet "krav".

Virksomheden/Forretningsforståelse

Som udviklere har vi forsøgt at sætte os ind i Fog's behov som forretning. Tilgangen til designet af systemet, gik ud fra et repræsentantmøde med personalet fra Fog.

Vi sendte repræsentanter ud til forretningen for at møde vores ressourceperson (Martin). Herfra kunne vi prøve at sætte os ind i de konkrete udfordringer forretningen står overfor med deres eksisterende systemer, samt hvilke behov de har til et fremtidigt system.

Ud over at systemet generelt var forældet, var der komplikationer der gjorde at systemet ikke længere kunne følge med forretningsudviklingen hos Fog. Blandt andet var det ikke muligt hverken at tilføje, slette eller opdatere information på varer. Dette betød, at man var nødsaget til manuelt at rette i generede styklister på ordrer, til stor frustration hos personalet.

Udover den manglende redigerbarhed, var systemet også meget afkoblet fra resten af it-systemet.
Når en kunde tilgik designportalen, foregik en ordreafsendelse således, at kunden udfyldte en form med adresse, kontaktinformation og ønskede mål samt andre specifikationer på carporten. Herefter ville deres bestilling blive sendt til vurdering hos Martin og øvrigt salgspersonale, for derefter at blive indtastet manuelt i et lokalt eksternt program for at generere styklister.

Martin var dog ikke helt utilfreds med de opdelte programmer. Han mente at en vigtig del af en salgsoplevelse hos Fog, var at kunderne var i kontakt og dialog med personalet undervejs. Selvom at det virkede unødvendigt, at Martin og salgspersonalet manuelt skulle indtaste data kunden allerede havde indtastet ved bestilling for at generere styklister, skabte det alligevel en situation hvor ethvert salg kom til at foregå igennem personalet.
Personalet var derfor imod en total automatisering af kundebestillinger.

Ud fra dette møde uddragede vi konkrete user stories, systemet kunne designes ud fra, hvilke svagheder de tidligere systemer havde, og hvor de primære fokuspunkter skal være.

Vi mener at have løst opgaven til et punkt, hvor Fog vil have fået opfyldt deres ønsker og behov, uden at gå på kompromis med deres ønsker om kundekontakt og personlighed i salg.

Krav

Fog ønsker at forbedre deres online værktøj til design og bestilling af skræddersyede carporte, samt udvide og forbedre mulighederne en administrator har i systemet. Vores design giver kunder muligheden for at designe en carport til de mål, den enkelte kunde ønsker, samt til- og fravælge evt. skur og/eller trapeztag. Derudover giver vores system administratorer muligheden for at behandle ordrer, give personlige tilbud til kunder, se alle ordrer i systemet, ændre på status i varelager samt tilføje, fjerne eller redigere varer, se styklister for de enkelte ordrer, og sidst at downloade genererede SCAD-filer, der kan bruges til 3D-print af de unikke carporte.

Funktionelle krav i form af user stories:

- Som bruger vil jeg gerne oprette login for at bestille en carport.
- Som bruger vil gerne kunne designe min egen carport.
- Som bruger vil jeg kunne logge ind for at se mine ordrer.
- Som bruger vil jeg gerne se styklisterne på mine færdige ordrer.
- Som bruger vil jeg kunne downloade en scad-fil med en 3D model af min færdige ordre.
- Som admin vil jeg kunne logge ind for at se alle lagte ordrer samt status.
- Som admin vil jeg gerne kunne opdatere mit varelager.

Tekniske krav:

- Løsningen skal baseres på en MySQL 8 database.
- Designet skal afspejle en flerlagsarkitektur, der afvikles på en Java server (Tomcat 9).
- Webapplikationen bygges af almindelige Java klasser, servlets, og JSP-sider.
- Websiderne skal fungere i enten Google Chrome eller Firefox.
- Websitet skal deployes i skyen (Digital Ocean)
- Kildekoden skal være tilgængelig i et GitHub repository.
- Den bestilte carport skal kunne printes på en 3D printer. I kan f.eks. aflevere en *.stl fil, som dokumentation og vise modellen frem til eksamen. I kan også indsætte fotos i rapporten af den printede udgave.

SWOT-opstilling af gruppen: [1]

Den følgende tabel er en SWOT-opstilling for henholdsvis gruppen, (hold_A_gruppe_4):

Interne forhold	
Stærke sider (Strengths)	**Svage sider (Weaknesses)**
● **Hærdet team** ● **Høj ambition** ● **Produktive medlemmer** ● **God arbejdsfordeling** ● **God stemning** ● **Projekterfaring** ● **P8 Jazz**	● **Vi er en mand færre end sidst.** ● **Nye programmører** ● **Nye opgaver (Droplets, 3D-print ol.)** ● **Dårlig sikkerhed vedr. hacking** ● **Ingen erfaring med phone first design**
Eksterne forhold	
Muligheder (Opportunities)	**Trusler (Threats)**
● **Centraliseret database/deployment** ● **Vejledning fra undervisere** ● **Søstergruppe til sparring ol.**	● **Hackere** ● **Lille gruppe betyder at mandefald er katastrofalt** ● **Der er mange private aftaler i Maj**

[1] (Systime Marketing. 2023)

Uddybelse af SWOT-opstilling

Stærke sider (Strengths):

I forhold til vores stærke sider, kan det deles op i gruppe, arbejdsforhold og stemning. I forhold til gruppe, vil vi mene at vi er en stærk gruppe, da vi har arbejdet sammen gennem alle projekterne på både første- og andet semester, hvilket har gjort os til et hærdet team med en del projekterfaring sammen.

I forhold til arbejdsforhold er nok en af de stærkeste side vi har, da vi har et højt ambitionsniveau for de givne projekter, samt den generelle tilgang og lyst til projektarbejde, der afspejles af lange og produktive dage. Derudover er vi gode til at afholde korte standup-møder, hvor arbejdsfordeling for det meste bliver planlagt.

Stemning i gruppen, er et meget vigtigt element/punkt for os før, efter og under arbejdsprocessen. Selvom alting under arbejdsprocessen ikke går som planlagt, forbliver stemningen på samme niveau. Stemningen har dog en undtagelse, da stemningen altid bliver lidt bedre under arbejdet, når vi lytter til noget P8 Jazz.

Svage sider (Weaknesses):

En lidt større ændring fra sidste projekt til, har været med til at skabe en ny svage side for vores gruppe, dette er angående færre gruppe medlemmer, da vi nu kun er tre, hvilket har gjort at der skal gøres mere individuelt for at komme i mål.

Det kommer nok ikke som nogen overraskelse, at vi er nye programmører, med kun knap et års erfaring. Dette har også gjort, at der er flere problemer der skal overkommes i nyere opgaver som for eksempel droplets, 3D-print, samt at designe en webapplikation og vores viden om håndtering af sikkerhed i forhold til webapplikation.

Sidst har vi erfaret i Cupcake-projektet, at vi ikke er stærke i design, der skal se lækkert ud både på en computerskærm og på en mobiltelefons skærm.

Muligheder (Opportunities):

Vi har et stærkt team af undervisere i ryggen, herunder skaberen af det library, vi bruger til generering af 3D-modeller. Derudover har vi resten af vores årgang til sparring, herunder vores “søstergruppe”, hold 8.

En ny mulighed for vores gruppe er, at vi nu har muligheden for at have en fælles database, så vi bl.a. ikke behøver at synkronisere vores private databaser med scripts hver dag, som vi før i tiden har gjort.

Trusler (Threats):

Da vi nu kun er en gruppe på tre medlemmer, kan dette have effekt på for vores gruppearbejde, hvis nu en af os tre går ned med sygdom, da dette ville resultere i mere arbejde for de resterende i gruppe, samt langsomme arbejdsprocesse der skal overkommes. Desuden er der allerede nogle datoer i maj måned, hvor nogle gruppemedlemmer ikke kan være til stede grundet private aftaler.

Vores viden om cybersikkerhed er desværre begrænset, og især vedrørende droplets og deployment er vores viden mangelfuld. Vi kan se via en serie af linux-kommandoer hvor ofte og hvor mange der har forsøgt at få adgang til vores droplet, og i løbet af de første 3 dage, var det i gennemsnit hvert andet minut, at nogen havde forsøgt det. Vi har beskyttet os så godt som vi kan, men det er definitivt en svaghed, at vi ikke har større viden om forsvar imod disse angreb.

Interessentanalyse: [2]

Gidsler: ● **Fog's kunder** ● **Fog's medarbejdere**	**Ressourceperson:** ● **Os selv (som udviklere)** ● **Martin (Fog)**
Eksterne interessenter: ● **Rektor / Virksomhedens chef** ● **Andre byggecentre, da den nye feature er inspireret af, hvad de andre <u>ikke</u> gør**	**Grå eminence:** ● **Undervisere / Teamledere i virksomheden** ● **Fog's ledelse**

[2] (Systime Projektarbejdet. 2023)

Teknologivalg

Følgende teknologier er anvendt til at løse opgaven:

- Koden til webapplikationen er skrevet i Java openjdk-17 version 17.0.7 og med Java Database Connectivity (JDBC).
- Databasesystemet der er benyttet til opgaven er en relationel model baseret på databaseserveren MySQL. Databasen ligger på en Digital Ocean Droplet.
- Stylesheet er kodet i CSS og implementerer både Twitter Bootstrap 5 og original kode.
- Som webcontainer til at køre og administrere webapplikationen bruges Apache Tomcat 9.0.
- Som Integrated Development Environment (IDE) er IntelliJ IDEA 2021.2.4 (Ultimate Edition) benyttet.
- Til versionsstyring bruger vi Git og Github og i den forbindelse både GitHub Desktop og IntelliJ's Git-integration og Git direkte i GitBash og Windows Powershell.
- Google Docs og Javadocs til dokumentation.
- Github Projects som Kanban Board.
- MySQL workbench og IntelliJ's indbyggede database-support til database management.

DoD

- Acceptkriterier skal være opfyldt. Alternativt er et eller flere kriterier ændret/fjernet i enighed med forretningen.
- Koden skal være unit testet hvor der er databearbejdning/beregning - whitebox af koderne selv. Herunder fejlhåndtering.
- Kodestandard er overholdt.
- Kendte fejl er dokumenteret.
- Koden er skalerbar.

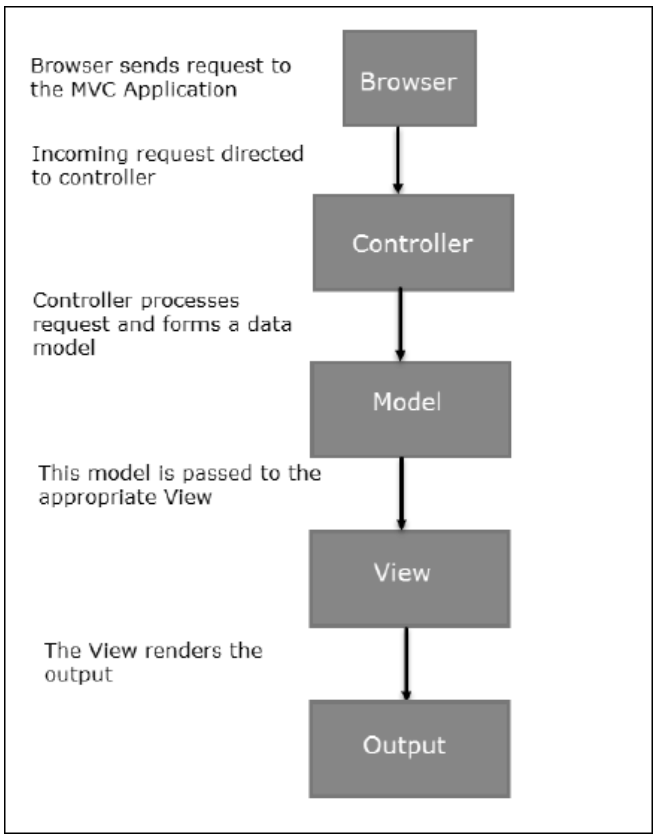
Kodestandard

- Fejlhåndtering: Mindre fejl som indtastning af ugyldige værdier giver fejlbesked på aktuel side. Andre fejl (ikke input) sender bruger til errorpage.
- Code cleanup - overflødige imports, variable, metoder og klasser er fjernet.
- Kode der ikke er i brug skal dokumenteres og begrundes.
- Navngivning:
    - Variable på tværs i systemet, der refererer til det samme, skal hedde det samme.
    - Servletlinks er all lowercase.
    - Direkte links er get og links fra forms er post.
    - jsp-sider er navngivet i camelCase med lille begyndelsesbogstav.
    - Java naming conventions overholdes: https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html. Curly brackets på **new line.**
    - Alt backend-kode er på engelsk, alt kunden ser er på dansk.
    - Alle medier har beskrivende navne.
- I mappen Images lægger man undermapper navngivet efter den side, de hører til. Her lægges billeder etc.
- Koden er dokumenteret med klassediagrammer og sekvensdiagrammer i UML. Lægges i mappen Documentation og findes i rapporten
- Overhold MVC (design pattern model-view-control), og det afspejles i projektstrukturen.
- Vi koder i Java 17 og IntelliJ 2021.

Valg af arkitektur

Vi har taget udgangspunkt i mvc-arkitektur, altså Model-View-Control. Entiteter, mappere og utility-klasser repræsenterer her Model, altså den tunge kode, der genererer alt hvad vores controller skal bruge. Servlets repræsenterer Control, altså bindeleddet imellem JSP-siderne og dataen, vi får fra backenden. JSP-siderne repræsenterer view, altså delen der viser outputtet til en bruger.

Når en bruger sender en forespørgsel ved f.eks. at submitte en form, behandler en Servlet dataen, lader utility klasserne køre evt. beregninger og metoder, og sender derefter brugeren videre til en ny jsp-side.



Til vores database-forbindelse bruger vi en ConnectionPool. Det giver os mulighed for at “genbruge” vores forbindelser til databasen, så vi ikke skal lave en helt ny hver gang vi har et kald til den. Det giver os muligheden for så at sige opbevare vores forbindelser i en pool, indtil vi skal bruge dem igen. Da vores program er afhængigt af en del databasekald, gør denne tilgang vores program en hel del hurtigere og “billigere”, end hvis vi blot brugte connections.

Matematik og logik

Herunder følger formlerne, vi er kommet frem til, og lidt beskrivende tekst dertil. Matematikken kan i projektet findes i klasserne PartslistCalculator og i MetalCalculator, der begge findes i src/main/java/model/utilities.
Algoritmerne er designet således at den højeste prioritet for et valg af træ, vil være med minimalt muligt spild frem for antal. Dette vil sige når vi f.eks. udregner sternbrædder på en carport med en bredde på 600, vil programmet prioritere 3 stykker træ på 205 cm i længde frem for 2 stykker af 410 i længde.
Dette gælder dog ikke udregningen af spær, da vi helst vil have færrest mulige samlinger for at styrke konstruktionen.

For matematiske udtryk herefter, bruger vi $\perp x \perp$ til at beskrive et tal der bliver rundet ned, og $\top x \top$ til et tal der bliver rundet op.

selectWood():
Det første vi gør i hver udregning er at finde ud af, hvilket træ vi skal bruge. Det gør vi via funktionen selectWood(). Den starter med at lave et Wood object vi kalder buffer. Derefter henter vi en liste over al vores træ af den specifikke variant (spær, stern, stolpe osv.) ned, og løber den igennem via et enhanced for loop sorteret fra længste til korteste træ. I loopet tjekker vi om det stykke træ vi kigger på er langt nok til at dække afstanden vi skal bruge. Hvis den er, bliver buffer-objektet sat til det træ, vi kigger på, hvorefter loopet kører igen. Når vi rammer et stykke træ, der ikke er langt nok, eller når vi ikke har mere træ at løbe igennem, bliver buffer-objektet returneret, så vi ender med at have det træ, der lige præcist er stort nok til at fylde området vi skal have fyldt ud. Hvis ikke der kan findes et stykke træ der er langt nok til at passe i længden, returnerer metoden null.

getOptimalItem:
Hvis denne metode kaldes, skyldes det et null fra selectWood(), og vi skal nu til at regne på hvilket stykke træ, og hvor mange vi skal bruge for at dække længden. Vi sammenligner de forskellige muligheder udfra to tal, waste og amount. Vi looper

igennem træet, og finder det træ, der skal bruges færrest mulige stykker af, med den laveste waste, altså mængde træ spildt. Vi returnerer til sidst et objekt af OrderItem, der indeholder det valgte træ og det endelige amount.

Stern:

For stern finder vi det optimale OrderItem, altså trætype og antal ud fra funktionerne ovenfor, først længden og derefter bredden, og ganger antallet med 2, da en carport er rektangulær, og derfor har 2 ens sider i længden og i bredden.

Tagplader:

For tagpladerne finder vi det samlede areal for carporten. Vi har i vores opgave valgt kun at have 1 type tagplade, der er 1 kvadratmeter. Hvis vi havde flere tagplader, ville vi loope igennem på samme måde som i stern, og have fundet den type tagplade vi skulle have færrest af med det mindst mulige spild.

$$\top Antal \top = \frac{Længde * Bredde}{10000}$$

Spær:

Spærene skal ligge med ca 55 cm mellemrum. Vi udregner hvor mange gange 55 går op i længden for at finde antallet og runder resultatet ned for ikke at lægge spærene tættere end med 55 cm mellemrum.

$$\perp Antal \perp = \frac{L}{Afstand\ mellem\ spær}$$

Hvis ikke et enkelt stykke træ dækker bredden af carporten, deler vi bredden i 2, og foretager samme kalkulation med det dobbelte totale antal spær.

Stolper:

Stolper har vi kun en slags af. Vi regner først flyvet fra carportens længde og bredde, og antager at vi skal have 4 stolper, en i hvert hjørne. Derefter regner vi ud, hvor mange stolper der skal bruges i hh. længde og bredde ved at sige

$$\frac{Afstand - Flyv * 2}{Maksimal\ afstand\ imellem\ stolper} * 2.$$

Design valg - frontend

Brugergrænsefladen

Under udarbejdelsen af brugergrænsefladen, har vi taget udgangspunkt i Laws of UX for at kunne opnå den bedste brugeroplevelse. Det har vi gjort, da Laws of UX menes at være en samling af bedste praksis, når der tales om brugergrænsefladen.

Vi har henholdsvis taget udgangspunkt i: Doherty Threshold, Jakob's Law, Miller's Law:

Doherty threshold

Et vigtigt element for en slutbruger er deres tid. Lige meget om den slutbruger er en ansat hos fog eller kunde, har det en stor betydning, hvor meget tid der bliver brugt på at udføre en handling på webapplikation. Derfor er diverse JSP-sider designet med Doherty threshold i baghovedet. Vi har prøvet at forenkle diverse side så meget som muligt, da for mange elementer på siderne kan være med til at øge den tid det tager at indlæse siden. Andre ting vi har gjort for, at formindske indlæsningstiden er for eksempel at billedstørrelse på et givent billedet er ca den samme størrelse som selve billedet er selv, for hvis vi eksempelvis siger at et billede er 1000x1000 pixels, men størrelsen på billedet bliver sat til 200x200 på siden ville det tage ekstra tid at indlæse dette element.

Jakob's Law

Vi har brugt Jakob's Law til at simulere nogle forskellige sammenligninger mellem vores webapplikation og fog's egen hjemmeside, så disse til en vis grad minder om hinanden. Dette har vi gjort i forhold til brugen af samme fog's logo, farver, samt den formen der udfyldes for en skræddersyet carport.

Miller's Law

Ud fra en brugers synspunkt kan en webapplikation hurtigt blive uoverskuelig og svær at gennemskue. Derfor har vi valgt at benytte Miller's Law, hvor vi holder os til arbejdshukommelse af brugeren. Her har vi eksempelvis aldrig for mange elementer

i spil på samme tid. Dette kan tydeligt ses i forhold til vores navbar, der befinder sig på toppen af alle JSP-siderne, hvor der aldrig er flere end tre muligheder at vælge imellem.

Særlige forhold

Det følgende særlige forhold er en serie af afsnit, der benyttes i programmet i dette afsnit vil vi gerne fremhæve disse:

Session scope

Vi benytter os kun af sessionscope til den givne User der logger på webapplikationen. Sessionen ugyldiggøres når der logges ud. Ud over brugen af session, bliver der gjort brug af requestscope for den resterende del af webapplikationen. Grunden til benyttelse af requestscope i stedet for sessionscope, er blandt andet for ikke bruge unødvendig hukommelse, samt for at sørge for, at de givne data/objekter i requestscope kun er tilgængeligt et bestemt sted og når response er sendt tilbage til klienten er denne data ikke længere tilgængelig.

Exception håndtering

Under tidlig udvikling håndterede vi SQL-exceptions i selve mapperen, og håndterede alle andre fejl i servletten. Dette gjorde, at vi har kunnet teste SQL-exceptions i udviklingen af programmet, og har sørget for at det ikke bliver et problem efter launch. Efterfølgende har vi lavet det om så alle CRUD fejl bliver kastet som “DatabaseExceptions”, og bliver sendt ud til servletten der har lavet kaldet, da det giver os mulighed for at sende brugeren til vores errorpage og beskrive fejlen. Fordelen ved dette er, at det er en del kønnere og mere informativt for en bruger, at ramme en specifikt designet errorpage, i stedet for bare at ramme en generisk fejl 500 side e.l.
I forhold til ugyldige indtastninger i forms på JSP-sider, kan vi tage udgangspunkt i login. Hvis der forsøges at logge på uden held, ved for eksempel forkert email eller kodeord, vil useren blive mødt af en “error message”. Det kan f.eks. være “Din e-mail eller kode er forkert!”. Derefter bliver brugeren sendt retur til login siden. Dette gøres ved på controller-siden at tjekke, om de informationer vi skal bruge svarer til informationerne på databasen, og på om der overhovedet er indtastet noget.

Validering af brugerinput

Brugerinput bliver tjekket først ved at læse fra databasen, om brugeren findes. Hvis dette ikke er tilfældet, fremvises der en fejlmeddelelse om forkerte login-oplysninger. Hvis brugeren findes, tjekkes der om brugerens rolle er admin eller user.
Vi benytter os af to brugertyper: admin og user. Disse to brugertyper bliver tjekket ved login, hvorefter brugeren bliver sendt til henholdsvis Admin Panel og User Page.
Når brugeren bruger webapplikationen, er brugerinputtet sat i faste rammer. Interaktionen mellem brugeren og applikationen vil som regel foregå ved tryk på knapper på siderne, så vi kan begrænse hvor mange forskellige input vi behøver at håndtere. Når data skal indtastes manuelt af brugeren, som f.eks. når en administrator skal tilføje eller opdatere et stykke materiale fra varelageret, er der minimum og maksimum værdier sat i HTML tags. Udover dette, bliver indsendte værdier tjekket inden der bliver foretaget CRUD operationer, og på den måde kan vi også kaste fejl og stoppe operationen, inden vi opnår et ulovligt stadie.

Sikkerhed

Vi har forsøgt at skabe en delvis sikkerhed i Fog's webapplikation. Her i blandt, benyttelsen af doPost i servlets, i stedet for doGet, når der udføres login sekvens til webapplikationen, da processen via. doPost ikke sendes til url'en med de givne brugerdata. Dog ønsker vi at understrege, at selvom doPost-requesten ikke er synlig i URL'en, har det ikke den store betydning, da disse informationer for login kan findes ved brug af udvikler-værktøjer, der kan tilgås i en browser via. "Ctrl + Shift + i" eller blot "F12". Når det er sagt, var dette dog blot et forsøg på at formindske uønsket tilgang til brugerdata.
Sikkerheden i forhold til MySQL er noget højere end login processen. Vi benytter os af en database, der ligger på vores virtuelle maskine. Vores MySQL database kan tilgås via to brugere: én administrativ bruger - Dev og én anden bruger - Fog. Den administrative bruger kan kun blive tilgået via. SHH-nøgle. Fog brugeren kan tilgås over internettet, men har kun adgang til den relevante database, og har kun rettigheder til at foretage basis CRUD-operationer. Samt har Fog brugeren et relativt langt og sikkert adgangskode på 19 tegn, med henholdsvis store, små, tal og ASCII-tegn.

Character Encoding

Den udleverede startkode har sat page encoding til UTF-8 i “pagetemplate.tag”. Dog har der stadigvæk været problemer/komplikationer med character encoding, hvor danske bogstaver “Æ, Ø og Å” vil blive vist som fx. “ä |” på JSP-sider, samt i servlets ved brug af: “request.getParameter(“String”)”. Dette har skabt flest problemer for søge-funktionaliteten i administratorens varelager side, samt i styklisterne. De fleste servlets har derfor “request.setCharacterEncoding("UTF-8");” i begyndelsen af servletten. Dog er det vigtigt at bemærke at, hvis denne ikke er i begyndelsen af servletten har den ingen påvirkning.

Stykliste

Styklisten bliver genereret af vores “PartsListCalculator” (PLC) og “MetalCalculator” (MC), der indeholder alle algoritmer der bruges til at generere styklister ud fra. Som udgangspunkt står PLC for genereringen af lister over “OrderItems”, som er et Java objekt vi har lavet til at indeholde både materialer, antal af materialer og beskrivelser af dem. Når en ordre bliver lagt, skal styklisten helst være klar med det samme, så en administrator kan logge ind og tjekke styklisten efter, og så prisen for materialer kan udregnes med det samme, når administratoren skal give et tilbud på ordren. Når en kunde lægger en ordre, bliver materialCalc() metoden kaldt i PLC. Den starter en sekvens af interne metodekald, der så hver især kalder på MC for udregning af skruer, beslag o.l.. Hver enkelt metode returnerer en liste over “OrderItems” der indeholder type, antal, og beskrivelser. Listerne bliver lagt sammen i en enkelt liste i materialCalc(), og bliver derefter skrevet til databasen med en reference til den ordre (“Receipt”), de er genereret ud fra.

3D print:

Der kræves programmer til at åbne SCAD-filer, til at rendere og transformere disse til STL-filer, og til selve 3D-printet. Vi har brugt OpenSCAD og Ultimaker Cura til disse formål. Læs mere om 3D-print i afsnittet 3D-print og OpenSCAD.

Udvalgte kodeeksempler

selectWood()

```
private static Wood selectWood(List<Wood> woods, double length)
{
    Wood buffer = null;
    for (Wood w: woods)
    {
        if (w.getLength() >= length)
        {
            buffer = w;
        }

        else
        {
            return buffer;
        }
    }
    return buffer;
}
```

selectWood() er vores metode til at beslutte hvilket træ, der skal bruges til at dække en bestemt længde. Vi looper vores liste af træ (sorteret fra længst til kortest) igennem og tjekker, om det træ vi kigger på dækker hele længden. Hvis det gør, sætter vi vores buffer til at være lig med det træ, så det bliver kortere og kortere. Når vi på et tidspunkt rammer noget træ, der ikke er langt nok, returnerer vi bufferen. På den måde sørger vi for at få det korteste træ, der stadigvæk dækker hele afstanden.

getOptimalItem()

```
private static OrderItem getOptimalItem(List<Wood> woods, double dist, String desc, int initialAmount, int multiplier)
{
    woods.sort(new Comparator<Wood>()
    {
        @Override
        public int compare(Wood s, Wood t1) { return t1.getLength() - s.getLength(); }
    });

    Wood selection = selectWood(woods, dist);
    int selectionAmount = initialAmount;

    if (selection == null)
    {
        Wood buffer = null;
        double amountBuffer = 1000000;
        double wasteBuffer = 100000;


        for (Wood w : woods)
        {
            double amount = Math.ceil(dist / w.getLength());
            double waste = w.getLength() * amount - dist;

            if (waste < wasteBuffer || amount <= amountBuffer)
            {
                amountBuffer = amount;
                wasteBuffer = waste;
                buffer = w;
            }
        }
        selection = buffer;
        selectionAmount = (int) Math.ceil(amountBuffer);
    }
        return new OrderItem( amount: selectionAmount * multiplier, selection, desc);
}
```

getOptimalItem() bygger videre på selectWood(). Vi starter med at kalde selectWood() for at se, om noget træ faktisk dækker hele afstanden. Hvis der ikke er det, løber vi vores træliste igennem igen. Den her gang tjekker vi først hvor mange stykker af et givent træ vi ville skulle bruge til at dække afstanden, og derefter hvor meget spild af træ der ville være. Så tjekker vi, om det træ, vi kigger på, er "bedre" end det bufferen er sat til. Vi prioriterer altid lavt spild over lavt antal, når vi bruger denne metode.
Når vi har loopet igennem hele listen, eller når vi har fundet et stykke træ via selectWood(), laver vi et OrderItem, der indeholder antallet af træ, hvilket træ det skal være, og en lille beskrivelse. “Multiplier” definerer hvor mange sider, der skal have denne behandling, så vi eksempelvis kan lave ens remme på hver side, med kun én funktion.

Status på implementering

- Alle user stories er fuldt implementerede.
- Javadocs dokumentation er kun lavet til klasserne i java.model.utilities.
- Programmet er designet til at have 2 typer skruer, og det er hardcoded hvornår de to typer bruges. Skal der kunne bruges flere typer skruer, skal der ændres i klassen MetalCalculator.
- Programmet er designet til at være så skalerbart som muligt. Lige nu kan man lave carporte med størrelser imellem 240*240 og 780*600, men programmet kan lave carporte i alle tænkelige størrelser. Skal man kunne designe carporte i andre størrelser end de, vi nu kan, skal der ændres i OrderForm.jsp.
- Alle CRUD metoderne er ikke lavet for alle tabellerne. Man kan redegøre for at hvis det ønskes at udvide applikationen, for eksempel at gøre det muligt at redigere eller slette en bruger, ville det være en god ide, hvis alle CRUD metoderne for alle tabellerne var lavet, dog har vi valgt at lave/taget udgangspunkt i de CRUD metoder der er vigtigst påvirkning for applikationen.

Idéer til fremtidige opgraderinger

- Administratoren kan manuelt ændre i en stykliste efter behov.
- Administratoren kan se alle brugere og deres oplysninger, samt at ændre disse.
- Brugere kan bestille carport med høj rejsning.

Test

Unit- og integrationstest

Test	Klasser	Metoder	Dækningsgrad
	User mapper	login, createUser, getUserByEmail og getAllUser.	Test succesfuldt login og opretning, samt forskellige fejltyper.
	Metal Mapper	getAllMetal, updateMetalPrice, deleteMetal, createMetal og getMetalById.	Test af CRUD operationer, samt exceptions ved fejlinput
	Wood Mapper	updateWoodPrice, deleteWood og createWood.	Test af CRUD operationer, samt exceptions ved fejlinput
	Receipt Mapper	getReceiptsByIdUser, createReceipt, acceptReceiptUser, acceptReceiptAdmin getAllReceipts og deleteReceipt,	Test af CRUD Operationer, samt exceptions ved fejlinput
	OrderMapper	createOrder,	Test af CRUD Operationer, samt exceptions ved fejlinput
	Model3D	generate3D	Test af succesfuldt output, samt exception

			ved forkert input
	ParsListCalculator	poleCalc, calcRafter, getShed, sternCalc og roofingCalc.	Metoder testet enkeltvis, til tilfredsstillende værdier er opnået
	Database Connection		Test af succesfuld databaseforbindelse.

Arbejdet med test

Arbejdet med test kan være en kritisk proces, hvis det ikke bliver gjort ordentligt. Vi har blandt andet tage den sikkerhedsforanstaltning, at benytte en test database "fog_test", for at sikre os at produktionsdatabase ikke bliver påvirket af diverse test der foretages, da dette kan have fortale konsekvens for systemet/webapplikationen, idet at blandt andet kundeoplysninger, ordrer, varer osv. kan gå tabt i processen.

Metoder som henholdsvis: Mapper, MetalCalculator, PartsListCalculator og Model3D, har vi identificeret som nogle mere kritiske metoder for webapplikationen, hvilket har gjort at vi især har testet disse før, at diverse branches er blevet merged ind i DeveloperBranch, for at sikker at den næste udgave af vores projekt er mindre fejlfrit og reagerer som planlagt.

Brugeroplevelse:

Vi har udført to black box tests af brugeroplevelsen af applikationen. De gik som følgende:

Første test - Businessmanden:

Vores første testperson er en mand på 52 år. Han er ejer af eget firma, og er en type med fart på. Han blev instrueret i, at han skulle bestille en carport, og så lod vi ham ellers prøve siden af.

Navigation:
Han navigerede fint igennem siden, fandt med det samme design-formen, og udfyldte den uden spørgsmål eller store overvejelser.

Feedback:
Han var forvirret over at skulle vente på, at Fog godkendte ordren, og mente også at der manglede information om, hvordan processen ved et køb forløber sig. Derudover ville han foretrække at have nogle skabeloner med tilhørende billeder, så han bedre kunne visualisere størrelsen på carporten, før han bestilte den.

Reaktion fra os:
Vi tilføjede 3 skabeloner til vores design-form, og tilføjede tekst til forklaring af processen under formen.

Anden test - Familiemoren

Vores anden testperson er en kvinde på 53 år. Hun har 5 børn i alderen 13-28, og bruger det meste af sin tid på at være mor, og på at arbejde i haven. Hun blev instrueret i, at han skulle bestille en carport, og så lod vi hende ellers prøve siden af.

Navigation:
Hun havde store udfordringer med at logge ind. Hun forsøgte fire gange at logge ind uden at lave en bruger i systemet først, og var ved at give op. Vi brød ind og gav hende hintet, at hun skulle oprette en profil først. Hun fandt knappen til oprettelse, og gik derefter i gang. På intet tidspunkt brugte hun vores "tilbageknap", men brugte i stedet for browserens egen tilbageknap. Derudover lagde hun ikke mærke til at man kunne bruge vores navigationsbar i toppen af siden. Hun fik dog lavet en bestilling uden problemer, da hun havde logget ind.

Feedback:
Hun kunne godt have tænkt sig nogle mere iøjnefaldende knapper, da hun både havde svært ved at finde tilbage- og sign-up knappen.

Reaktion fra os:
Vi har ændret vores sign-up knap fra at være et link under login knappen, til at være en mere iøjnefaldende knap. Vi har også gjort vores tilbageknap mere iøjnefaldende.

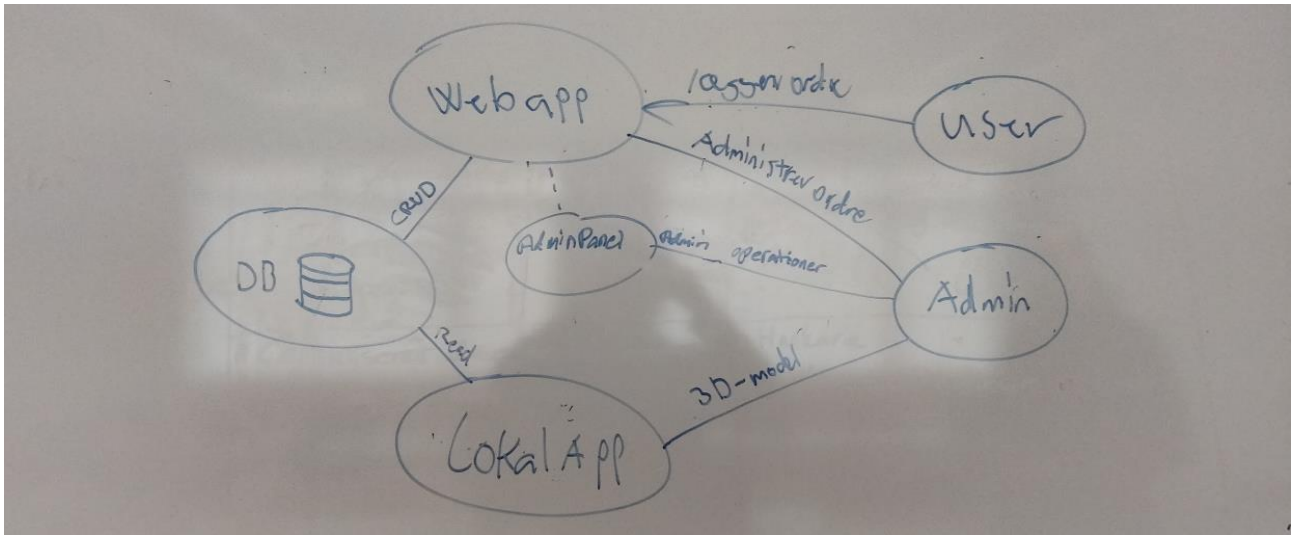
Funktionalitet:

Vi har skrevet mange af vores metoder som tests, før de er blevet en del af selve programmet. Især PartsListCalculator-metoder har fået denne behandling, da de alle står for matematiske operationer. Det har især i den kontekst været givende at starte i et miljø, hvor vi kunne teste output uden at skulle løbe hele programmet igennem. Først når vores tests har virket, har vi integreret dem i selve programmet.

Alle mappere og alle metoder til udregning af stykliste er unittestede.

Proces

Første design iteration



Første udkast af vores domænemodel. I starten var programmet designet i 2 dele ligesom det Fog allerede bruger. Tanken var, at administratoren kørte et lokalt program til at generere 3D modeller, men kunne hente data til programmet via samme database som webappen, og dermed slippe for at skulle indtaste allerede indtastet data manuelt.
Vi endte med helt at fjerne den lokale app, og i stedet for have en “download”-knap, hvor man kan generere og hente en SCAD-fil, med alt der behøves til 3D-print. (se “Domænemodel”)

Den følgende proces er en serie af punkter, af arbejdsproces i projektperioden:

The Great Wars of connectionpool

En af de største og mest interessante designudfordringer vi stødte ind i, var behandlingen af ConnectionPools. Siden vores ConnectionPool kun bruges i forbindelse med CRUD operationer, besluttede vi oprindeligt at nøjes med at hente den direkte i vores mappers, i stedet for at hente den i servlets og sende den med ned som argument igennem Facaden. Alt gik som smurt i olie, og poolen var altid tilgængelig i de mappers de skulle bruges i. Vi endte dog med at støde panden hårdt imod muren, da vi for alvor begyndte at unit-teste mapper-metoder. Vi opdagede, at

vi ikke længere kunne styre hvilken database vi benyttede til test, da vi effektivt havde hardcoded valget af database ind i alle mappers!
Vi måtte krybe til korset og kaste os ud i en større refaktorering af både mappers og servlets, så vi igen skulle sende en connection pool med som argument, til når vi skulle kalde metoderne fra testklasser. Operationen blev herefter refereret til som "The Great Wars of ConnectionPool".

Matematisk process ved udregning af materialer

En af de største udfordringer ved at skrive dette program, har været det matematiske aspekt. At finde de bedste algoritmer, der ikke bare præcist udregner det optimale træ/metal i de optimale mængder, men også er skalerbare, så Fog fortsat kan ændre og udvide deres sortiment, har vist sig at være en større logisk udfordring. Vi mener dog alligevel at have løst opgaven. I afsnittet "Matematik og logik", kan der læses mere om dette.

Testdriven development

De "tunge" dele af opgaven såsom algoritmerne til udregning af materiale til styklister og til at generere 3D-modeller er skabt under meget test-drevne forhold.
Algoritmerne består af mange små "hjælpe-metoder", som typisk er startet som JUNIT5 test-metoder i en test-mappe. Denne tilgang har gjort det meget nemt at eksperimentere sig frem, holde overblik og ikke påvirke resten af programmet imens udviklingen foregår. Når udviklingen af algoritmerne var ved at være på plads, er koden blevet implementeret i programmet, og testmetoderne har i stedet kaldt på de implementerede metoder, for at sikre at de virker som de skal.

Brug af polymorfiske byggematerialer

Vi har valgt at lade begge typer af materialer (træ og metal) udvide den abstrakte klasse Material. Det har vi gjort, da den eneste forskel i deres opførsel og opbygning er, at træ har størrelsesforhold. Dette gør, at vi igennem programmet kan behandle de to typer ens, og samle dem i collections. Det har gjort behandlingen af materialer meget mere strømlinet, samtidig med at det har gjort selve designet mere uniformt.

Intet er uden hager, og denne polyforme tilgang er ingen undtagelse. Desværre har JSTL svært ved at forstå polymorfi, og i grunden ville det også bryde MVC-modellen, hvis vi skulle lave logik og adskille de to typer materiale på en jsp-side. Derfor har vi valgt at behandle typerne polyformt i vores backend Java, for så at splitte dem op i en liste af hver slags materiale, før vi kobler dem på et requestscope.
Med denne approach udpensler vi også snitfladerne imellem front-end og back-end programmering, så fremtidige front-end-programmører ikke behøver kende Java, og forstå polymorfi.
I det oprindelige design behandlede vi også vores OrderItems, altså objektet, der indeholder både materiale og antal, som hhv. WoodOrderItem og MetalOrderItem. Da vi skiftede retning til den polymorfiske tilgang, valgte vi også at samle vores OrderItems i én klasse, der i stedet for at indeholde et Wood- eller Metal objekt, blot indeholdte et Material objekt, der kunne være enten en Wood eller Metal. Det uniforme OrderItem forsimplede både vores database og vores program, da vi nu kunne lave halvt så mange kald til databasen ved oprettelse af ordrer, samt kunne samle OrderItems i collections.

Konfiguration af Timezone i mysql

Vi ramte en udfordring ift. at generere Timestamps til vores ordrer. Da serverens timestamps altid blev genereret i UTC. Dette kunne løses på to måder. Enten ved at sætte et Event Trigger op, der ændrer Timestamps når de bliver genereret, eller ved at ændre på filen my.cnf, hvorfra MySQL får sin tid. Vi valgte at gå ad rute nummer to, da det i sidste ende betyder færre operationer på serveren.

Overgang fra Java 11 til Java 17

Vi startede med at tage udgangspunkt i startkoden, hvor denne kun var testet fra Java 8 til 11 i forhold til pom.xml og diverse dependencies. Derfor startede vi med at skrive vores program/webapplication med Java 11.
Vi besluttede os for at integrere vores 3D modeller med webappen i stedet for at lave en stand-alone applikation. Denne beslutning viste sig, at skabe nogle problemer da Java biblioteket (library) JavaCSG kun kan kompileres med Java 17 eller nyere versioner. Af den grund måtte vi konfigurere resten af vores program til at køre i Java 17 i stedet for 11, samt ændre versionen af Java på dropletten.

Arbejdsprocessen faktuelt

De fire faser

1. Uge - Opstart og overblik

   Den første fase (uge 1) tog udgangspunkt i at opstille diverse modeller som: SWOT-analyse, interessentanalyse, aktivitetsdiagram, domænemodel osv. Vi brugte tid på at sætte os ind i opgaven og dens krav, ud fra det givne materiale. Derudover brugte vi også første fase til opsætning, konfiguration af selve projektet, database og droplet der hører til.

   Herefter tog vi vores viden og ideer om opgaven i brug og dannede 8 user stories. De 8 user stories var et udkast, der efter diverse overvejelser og få rettelser fæstnede sig som grundlaget for resten af projektet.

2. Uge - Varelager og Stykliste

   Den anden fase (uge 2)

   Næste opgave var at få webapplikationen nogenlunde op at køre. Vi tog udgangspunkt i nogle af de lignende funktionaliteter som vi brugte i cupcake-projektet, såsom login-system, brugerside og administrator panel. Vi fokuserede først på at få login-systemet op at køre, så vi havde et grundlag for at teste brugerinteraktioner med det samme. Herefter kunne vi dele opgaver op imellem os, og på den måde arbejde på ordredesign-siden, administrator-panelet og bestillings-siden på samme tid. Midt på ugen havde vi en basis-webapplikation oppe at køre, og kunne begynde at tænke over designet af algoritmer til beregning af styklister.
   I slutningen af ugen havde vi udkast til det første design af algoritmer.

3. Uge - 3D

Den tredje fase (uge 3)

Efter at have fået basis-webapplikation, varelager og stykliste op at køre, begyndte vi på tredje fase, der deltes op i to dele. Generation af 3D modeller ud fra en ordre, og at kunne downloade en given model ud fra en ordre. Først startede vi med at finde en måde hvorpå man kunne downloade en vilkårlig fil, hvorefter det blev specificeret til en OpenSCAD fil. Efter at funktionaliteten af download var på plads, begyndte vi at generere 3D modellen ved brug af JavaCSG biblioteket. I denne del af processen startede vi med den forudsætning at printe hver del selvstændigt og derefter samle modellen. Senere besluttede vi os for, at modellen skulle printes som en hel carport. Dette besluttede vi, da vi kunne risikere, at processen med at printe alle dele særskilt til modellen vil kræve for mange printere og kunne ende med at være kritisk for arbejdsprocessen i forhold til tid.
Herefter forsøgte vi at printe modellerne i form af test print, hvor tydelige fejl endte i justering af de enkelte dele af modellen.

4. Uge - 4 Rapport, bugfixing og færdiggørelse

Den fjerde fase (uge 4)
Den sidste fase gik ud på at dokumentere og gennemteste applikationen så grundigt som muligt. Her havde vi bl.a black box brugertest af applikationen, udført af eksterne personer. Diagrammer blev rettet til og finpudset, der blev skrevet Java Docs til relevante dele af programmet, og kildekoden blev finkæmmet for at lave små effektiviseringer, og at rette småfejl.
Alt kode blev refaktoreret, til den aftalte kodestandard, og kommentarer blev skrevet for at gøre det nemmere at lave fremtidige opgraderinger.
Herudover færdiggjorde vi og finpudsede vores rapport.

Vejledningsmøder

Vi havde 3 vejledningsmøder undervejs i processen:

I starten af fase 2, havde vi besøg af Nikolaj, til en snak om designet af programmet. Alt i alt virkede han tilfreds over den funktionalitet, og arkitektur vi havde designet, men mente at vi godt kunne dokumentere designet bedre med modeller, som vi så har arbejdet på efterfølgende.

I slutningen af fase 2, havde vi et mere teknisk møde med Signe og senere Jon omkring opdelingen mellem vores frontend og backend. Vi har designet således at alle byggematerialer bliver behandlet polymorfisk, men vi skulle gerne have delt materialerne op i typer, når vi viser materialerne på en stykliste. Her debatterede vi om hvorvidt det gav mest mening at opdele listerne i en servlet inden de blev sendt til JSP-siden, eller om man skulle sende en samlet liste afsted, og opdele den i tabeller med scriplets i JSP.
Vi endte med at blive enige om den bedste måde at overholde MVC designet, var så vidt som muligt at holde logik ude af JSP-siderne, så der er bedre snitflade for eventuelle frontend og backend udviklere som (meget hypotetisk) skal arbejde på projektet i fremtiden.

GitHub Projects

Vi har benyttet os af Github Projects for at holde styr på arbejdsprocessen. Vores Github Projects board er opdelt i henholdsvis: backlog, bugs, todo, in progress, in review og done.

Vi har som team holdt styr på GitHub Projects, når vi har afholdt daglige møder, og undervejs på dagen når diverse opgaver er blevet løst. Der kan læses mere om ugentlige og daglige møder i det nedenstående afsnit kommunikation i teamet.

Kommunikation i teamet

Planlægningen i løbet af arbejdsprocessen, opdelte vi i ugentlige og daglige møder.

Ugentlige møder var primært til at skabe et overblik fra sidste uges arbejde og proces, samt for at samle op på de ting og elementer vi nåede at lave og de vi ikke nåede at lave. Her satte vi mål for ugen i forhold til hvad vi skulle nå, samt hvordan vi færdiggør sidste uges udstående opgave.

De daglige møder forløb sig som stand up møder, hvor vi så på forrige dag. Hver person fortalte, hvad de havde lavet i går, hvad de skulle lave i dag og om de har brug for hjælp med noget, samt andre indvendinger for i dag, i går eller generelt.

Når vi ikke er på skolen, benytter vi os af vores egen discord server, til at hjælpe og oplyse hinanden.

Arbejdsprocessen reflekteret

- **Demokratisk projektledelse**. Vi har dannet os overblik over status til morgenmøder og har taget beslutninger om hvilke drejninger projektet skulle tage, samt hvordan arbejdsfordelingen skulle foregå.
  Den eneste reelle rolle der har været besluttet har været rollen som "gitmaster".
  Her har vi aftalt at kun den udvalgte gitmaster kan lave merges til main-branch, for at være sikker på at den er gennemtestet inden den bliver merged, og for at være sikker på at der ikke er nogen der ved et uheld kommer til at sidde og skrive i main-branch.
- **Tæt arbejdsproces**. Vi har haft konstant intern dialog omkring projektet, og har vendt selv meget små detaljer med hinanden undervejs, for at sørge for at vi hele tiden er på bølgelængde, og har samme forståelse for hvor udviklingen er på vej hen. Det har betydet at hele gruppen har været opdateret i alle

forgreningerne af programmet, og at det derfor har været nemt at reflektere over problemer sammen undervejs.

- **Daglige morgenmøder.** Hver morgen har vi startet med at danne os et overblik over hvor vi er henne i processen og hvor langt forskellige processer er undervejs, opdateret vores Kanban-board, og delt tanker og ideer om de forskellige projekter.
- **Underopgaver baseret på teknisk opdeling frem for user stories.**
  Frem for at danne underopgaver ud fra user stories, har vi valgt at designe en samlet arkitektur af programmet ud fra dem, og derefter opdelt underopgaver som tekniske elementer ud fra den samlede arkitektur. Det synes vi har hjulpet os med at finde bedre og mere konkrete snitflader af opgaverne, og vi har kunne snakke om hvordan vi senere hen kunne koble tingene sammen.
- **Færdiggørelse frem for alt**
  Vi har haft stort fokus på at lave de ting færdige som vi er begyndt på frem for at starte på nye opgaver. Det har vi gjort for at være sikre på, at vi ikke gaber over mere end hvad vi kan klare på en gang. Selvom vi fejlestimerer hvor lang tid tingene tager, kan vi på den måde være sikre på at de er lavet og testet ordentligt.
- **Streng git protokol**
  For at sikre os at git-workflowet har været så strømlinet som muligt, har vi udover en dedikeret gitmaster, arbejdet ud fra en “Git Workflow” struktur med en dedikeret Developing Branch. Se diagram 5 “Git Workflow versionsstyring” under bilag.

3D-Print og OpenSCAD

I dette afsnit forklares processen hvorved vi generer og henter vores SCAD-filer, hvordan vi printer dem, og vores tanker heromkring.

I vores webapp har vi en knap, hvorpå der står “Download 3D Model”. Når en bruger trykker på den, downloader de en SCAD-fil, der indeholder en 3D-model af deres carport. Dette gøres ved at kalde på funktionen Model3D.generate3D(). Programmet genererer “helstøbte” carport modeller, som kan give Fogs kunder en idé om dimensionerne på deres bestilte carport. Modellerne bliver genereret ud fra et færdigt perspektiv. Det betyder, at der ikke bliver taget højde for længden på selve plankerne, men derimod de længder, de tilskårne planker vil have. Formerne på træet er dog genereret ud fra målene der hentes fra databasen, så man får en fin fornemmelse af hvordan det færdige produkt er dimensioneret ud fra det træ der er blevet valgt. Funktionen tager imod et idReceipt, udfra hvilket den henter en stykliste fra databasen, der er koblet til den givne kvittering.

‘Vi har opdelt 3D-modellen i 5 lag. Tag, stern, spær, remme og stolper. Vi generer carporten på hovedet, og starter fra taget og bevæger os derfra igennem carporten, til vi har printet stolperne. Dette gør vi ved at have separate funktioner, der laver hvert sit lag, og returnerer en model, der består af hvad end metoden bygger. i Model3D.generate3D() kalder vi så metoderne i den rigtige rækkefølge, og returnerer så til sidst én model ved at lave et union af alle modellerne. Klassen indeholder en global variabel der holder styr på lagene, ved at opdatere udgangspunktet på z-aksen hver gang de har kørt, så næste lag bliver naturligt genereret ovenpå

Download af 3D

Download af 3D print/generelt file fra droplet, samt lokalt via. tomcat, krævede en smule undersøgelse for, hvordan dette skal udføres, vi tog inspiration og lavet en

sammensætning udfra Mkyong.com[3] og javatpoint.com[4]:

Servlettens funktion doDownload() håndterer download således:
Vi starter med at instantiere et PrintWriter objekt via Response-objektets funktion getWriter(). Så henter vi "ServeltContext" fra servlet-konfigurationen. Vi bruger så ServletContext-objektets funktion getMimeType for at finde ud af, hvilken filtype vores File skal have. Mime-typen for Response objektet bliver sat ved brug af dens metode setContentType(). Herefter bliver der tjekket om "mimetype" lokale variable ikke er null. Hvis Mime-type er null, så bliver ContentType sat til "application/x-openscad", da det er den filtype, vi ønsker.

"Content-Disposition"'s header bliver sat for at informere klienten, at Response objektet skal blive behandlet som en vedhæftet fil. Parameteret "filename" bliver brugt til at specificere navnet på den fil, der skal downloades.

Der laves en FileInputStream, der læser indholdet af filen. Filnavnet er sammensat af en "SCADPATH" og "filename". Herefter looper vi vores FileInputStream igennem, læser filen byte til byte ved brug af FileInputStream.read() og skriver hvert byte til vores PrintWriter. While loop'et fortsætter indtil slutningen af filen. Efter dette bliver FileInputStream- og PrintWriter objekterne lukket

Derefter åbnes SCAD-filen i OpenSCAD, scales ned til 1.5% størrelse og centreres. Filen renderes og bliver eksporteret til en STL-fil. STL-filen åbnes så i Ultimaker Cura, hvor modellen kan placeres. Vi har designet carporten, så den på skolens printere ikke har brug for support, så modellen slices, gemmes og sendes til print.

[3] (Mkyong. 2010)
[4] (Javatpoint)

Litteraturliste

Internetkilder:

Systime Marketing. 2023. SWOT-opstilling[Online] Accessed May 1, 2023, https://marketing.systime.dk/api/fileadmin/indhold/skabeloner/SWOT-opstilling.docx.

Systime Projektarbejdet. 2023. 4.2 Interessentanalyse[Online] Accessed May 1, 2023, https://projektarbejdet.systime.dk/?id=185#c762.

MVC structure explained, [Online]Accessed May 26, 2023 https://www.tutorialspoint.com/mvc_framework/mvc_framework_architecture.htm

Mkyong. 2010. How to download file from website- Java / Jsp [Online] Accessed May 19, 2023, https://mkyong.com/java/how-to-download-file-from-website-java-jsp/.

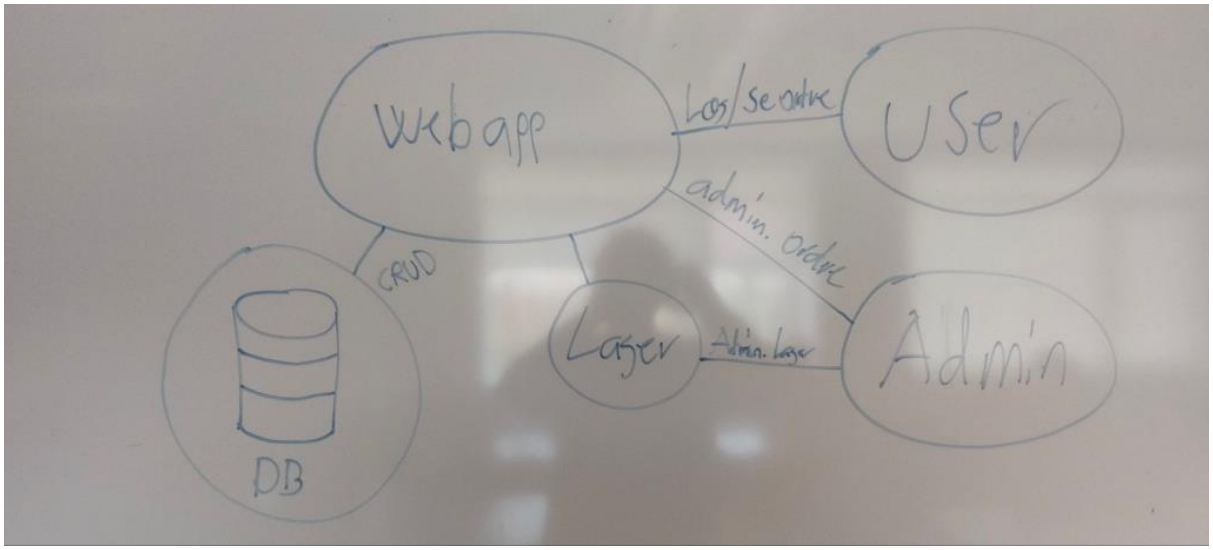
Javatpoint. Example of downloading file from the server in servlet [Online] Accessed May 19, 2023, https://www.javatpoint.com/example-of-downloading-file-from-the-server-in-servlet.

Sundhedsdatastyrelsen - Eksempel på domænemodel. Accessed May 30, 2023 https://sundhedsdatastyrelsen.dk/-/media/sds/filer/strategi-og-projekter/patientoverblik/domaenemodel.pdf?la=da

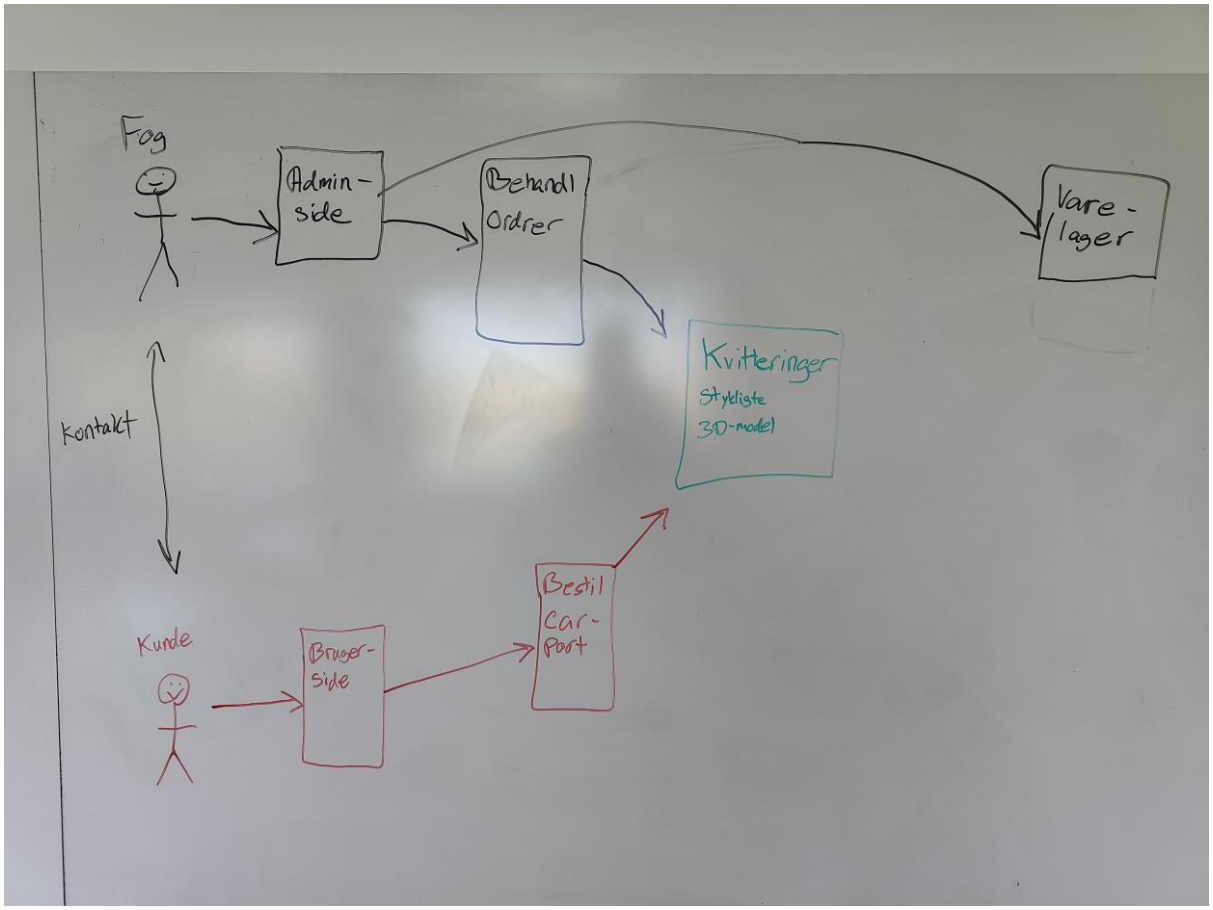
Git flow diagram overview - Accessed May 30, 2023 https://leanpub.com/git-flow/read

Bilag

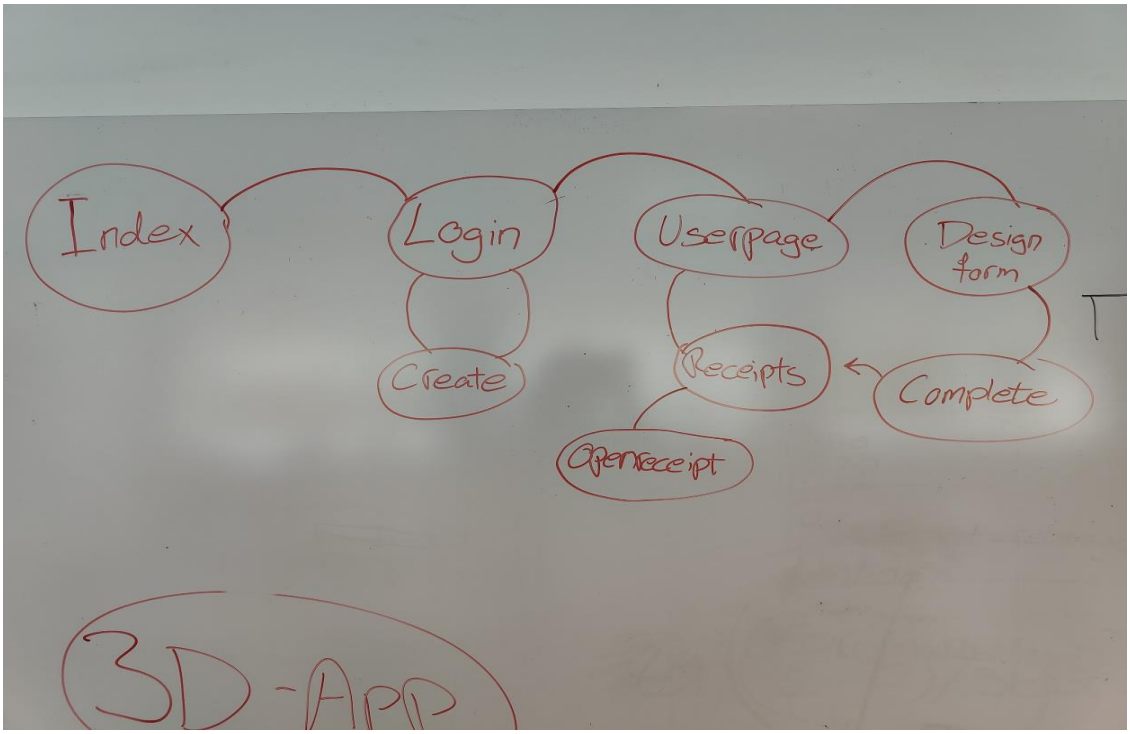
1. Domænemodel



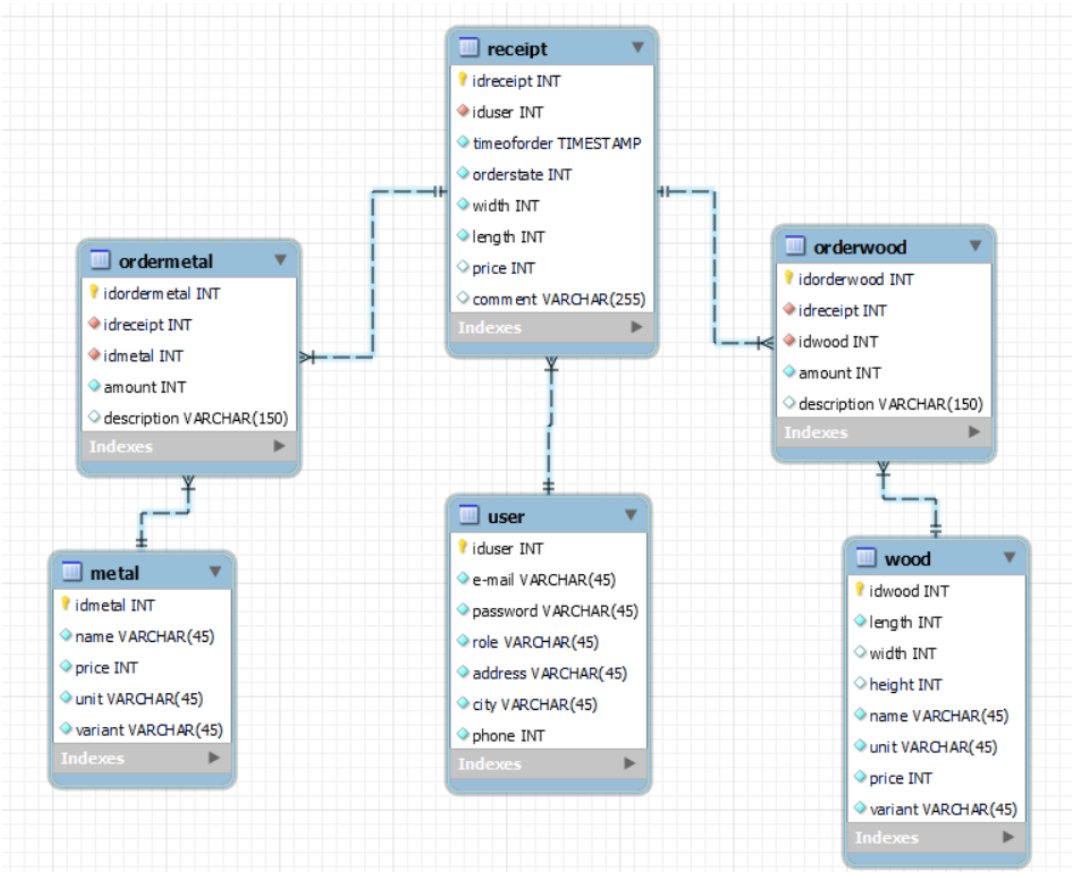
1.1 Domænemodel forsimplet - En efterrationalisering.



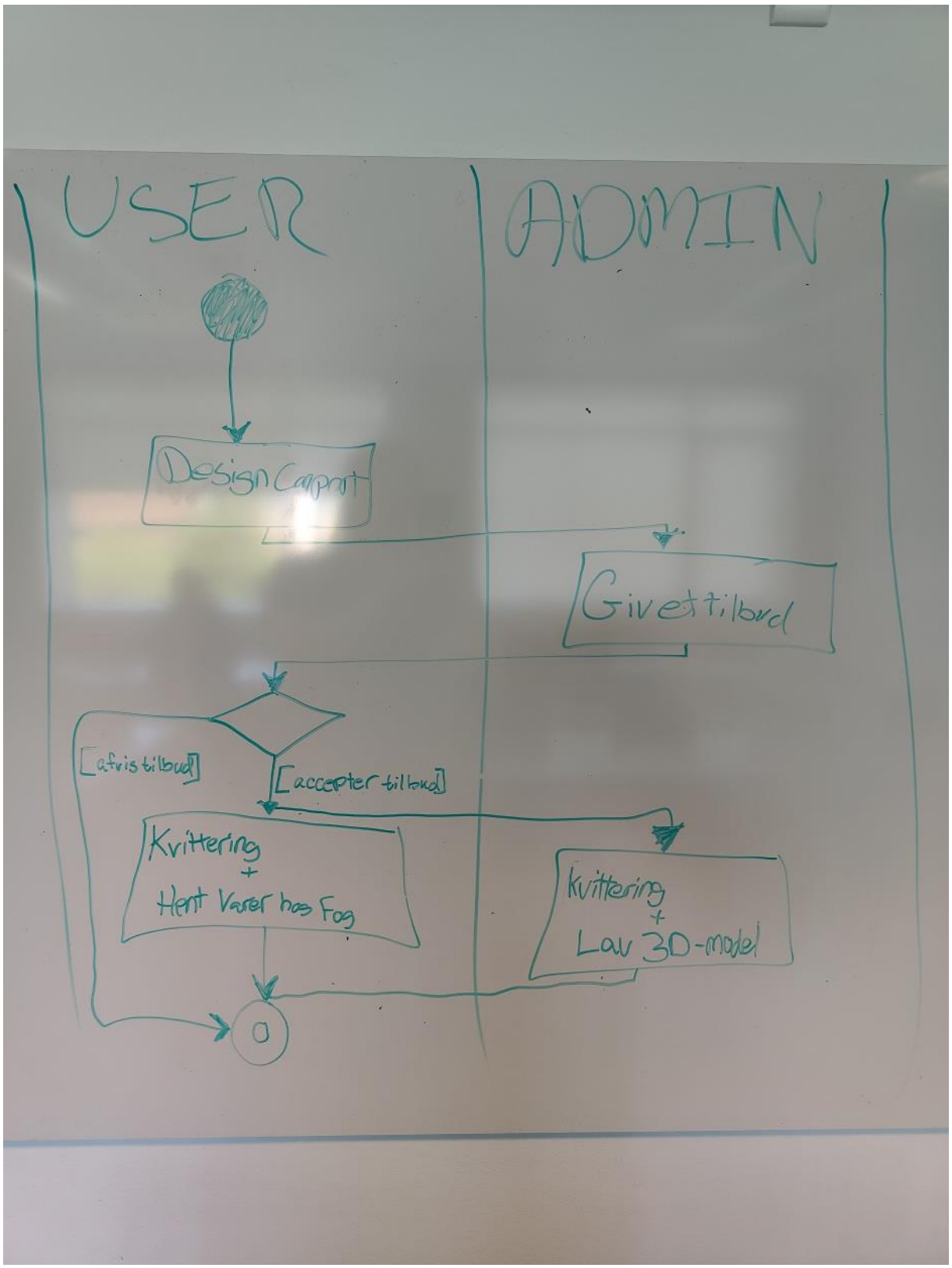
2. Navigationsdiagram for en bruger.



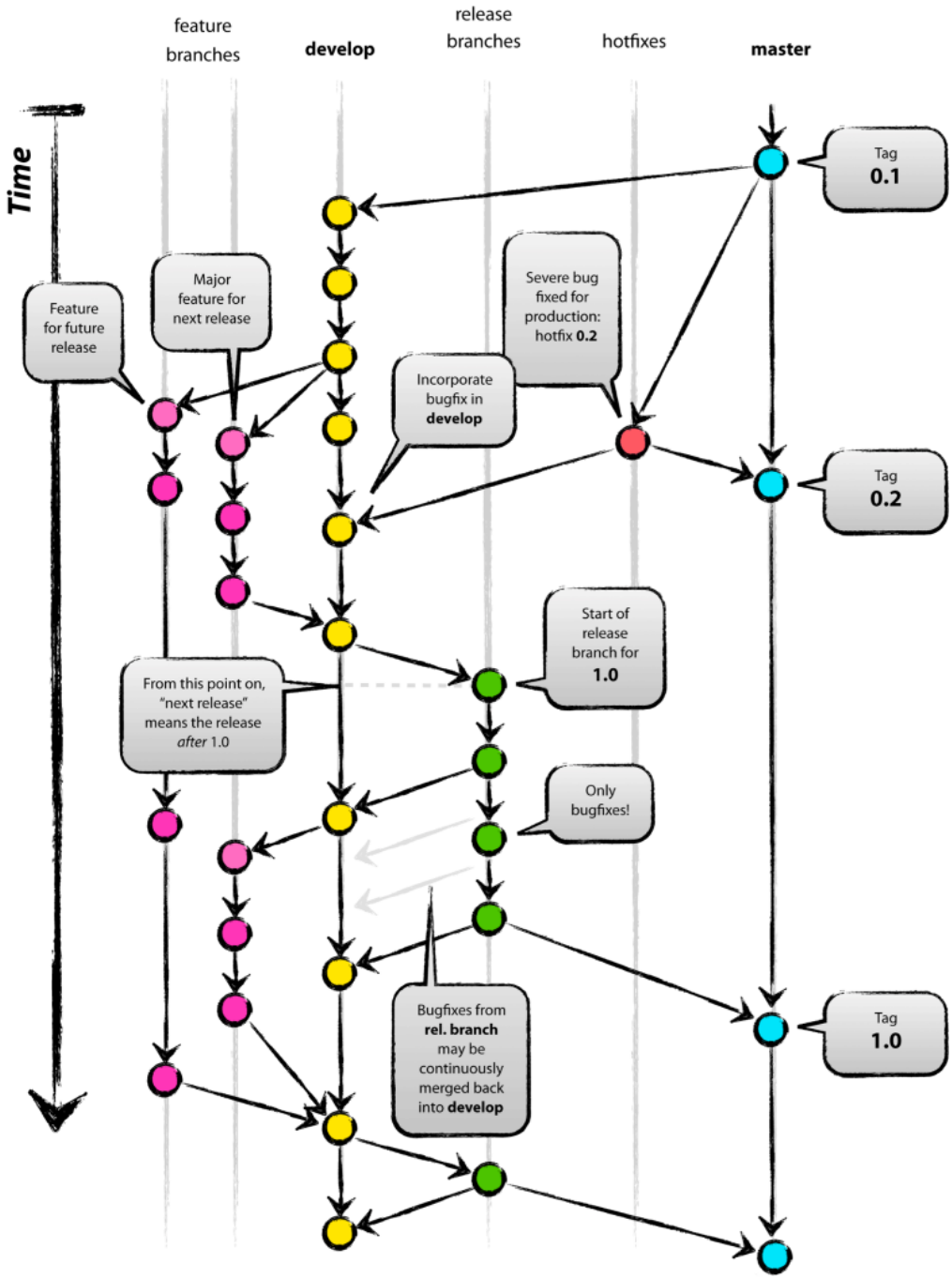
3. EER Diagram



4. Aktivitetsdiagram foruden login/o.l.



5. Git Workflow versionsstyring



5

[5] https://leanpub.com/git-flow/read

6. Klassediagram

Ses bedst på:

https://imgur.com/a/saDFN9R

Thumbnail: