

“New-Yorker Vægge” app rapport

Af: Jenna, Simon, og Mikkel.



(Antal tegn inklusive mellemrum: 64439)

Indholdsfortegnelse

Indledning	3
Problemformulering(Jenna)	3
Virksomhedsbeskrivelse (Jenna)	3
Mission	3
Vision	3
Forretningsmodel	4
Business Model Canvas	6
Finansiering	6
Implementering	7
Skalering	7
Metodevalg (Jenna)	7
Udviklingsproces	8
Projektstyring	8
Planlægning	8
Risikoanalyse	10
Github	10
Git, versionskontrol - og forskellen til Github	12
Environment	14
Programmering(Jenna)	15
Android	15
iOS	15
Generel udvikling	16
Persistering af data i Android	17
Unit testing af kode	17
Threads	17
Procesbeskrivelse (Jenna)	17
Iteration 1	18
Iteration 2	18
Iteration 3	19
Kravindsamling og -specifikation(Jenna)	20
Problemdomæne	20
Visionsdokument	20
Use cases	21
Use case diagrammer	23
Supplerne Kravspecifikation(FURPS+)	24

Object-Oriented Analyse (OOA) (Simon)	25
Aktivitetsdiagrammer	25
Data Ordbog (Simon og Mikkel)	29
Domæne model (Simon)	31
System Sekvens Diagram (SSD) (Simon)	34
Object-Oriented Design (OOD)(Mikkel)	39
Sekvensdiagrammer (SD)	39
Klassediagrammer (KD)	40
Klasser (Mikkel)	41
Kvalitetssikring (Mikkel)	49
Test af produkt	49
Review	49
Procesforbedring	50
Metrikker	50
Udgivelse (Mikkel)	50
Konklusion	51
Bilag	52

Indledning

Der er i dag mange virksomheder, som kan drage fordel af at få udviklet en app som kan få deres kunder tættere på dem og hjælpe dem i den daglige tilbudsproces. Vi vil i løbet af dette projekt undersøge hvordan vi kan understøtte New-Yorker.dk i dette forløb og vi håber at stå tilbage med et godt resultat som også kan udbredes til andre.

Problemformulering (Jenna)

New-Yorker.dk oplever stor efterspørgsel på deres vægge. Dette presser dem, da de både skal tage imod kundeforespørgsler, udarbejde tilbud og designe hver enkelt skillevæg.

Vi vil derfor undersøge hvordan vi kan udvikle en app som kan tage imod målene på en væg og derudfra udarbejde et tilbud og en tegning, som kan bruges til den videre produktion af væggen. Dette vil vi gøre ved at gøre brug af de værktøjer som vi har fået kendskab til på 1. og 2. semester af uddannelsen. Det skal samtidig hjælpe os med at opfylde læringsmålene som de er beskrevet i studieordningen for Datamatikere på Zealand Næstved årgang 2020.

Virksomhedsbeskrivelse (Jenna)

Vi betragter projektet som en mulighed for at danne en virksomhed, som kan udvikle Android apps og eventuelt viderebygge på fundamentet på dette projekt til en mere generisk app som kan sælges til andre virksomheder.

Vi ser vores virksomhed som interessentselskab, da det gør det muligt at stifte selskabet med flere ejere og uden en stor startkapital. Vi er bevidste om ulemperne ved solidarisk hæftelse, men betragter det som en fornuftig måde at starte på. Vi forventer at virksomheden senere kan konverteres til et anpartsselskab, når der er en større omsætning og vi så slipper for det direkte personlige ansvar.

Mission

Vores mission med dette projekt er at kunne levere den bedst mulige løsning til New-Yorker.dk så det kan lette deres hverdag, give mere tilfredse kunder og dermed øge indtjeningen i deres virksomhed. Det vil give os den erfaring der er nødvendig til, at vi kan tage vores virksomhed til næste niveau.

Vision

Vi vil appellere bredt til andre virksomheder, der sælger New Yorker vægge, ved at bygge et produkt som i fremtiden kan tilpasses både visuelt og funktionalitetsmæssigt, så det matcher andre kunders udtryk og ønsker.

Forretningsmodel

Produktet er en Android app, som kan benyttes til at skabe et tilbud samt generere en skitse på en New Yorker væg ud fra indtastede mål. Derudover kan der tilvælges andre tillæg til væggene såsom glastyper eller døre og app'en skal være i stand til at tilpasse pris og tegning ud fra alle disse valg.

Der findes i forvejen en lang række virksomheder, som producerer denne type vægge og efter at have undersøgt en del af deres hjemmesider kan vi se, at langt de fleste ikke har en konfigurator hvor man kan bygge en væg ud fra mål. De fleste virksomheder står i samme situation som New-Yorker.dk, hvor de har en kontaktformular som man skal udfyldet hvis man ønsker et tilbud.

Vi mener derfor, at der er en oplagt forretningsidé i, at samle erfaring ved at bygge denne app til New-Yorker.dk, men samtidig have for øje, at app'en senere kan ændres så den appellerer til andre virksomheder indenfor feltet, der har lignende udfordringer.

Det vil derfor være vigtigt, at vi i forbindelse med overleveringen af app'en til New-Yorker.dk, sikrer os, at vi fortsat har ret til at videreudvikle ideen, platformen og at videresælge denne til andre virksomheder indenfor feltet.

I forbindelse med udviklingen af forretningsmodellen, har vi benyttet et business modelling canvas (BMC) til hjælpe os med at få et overblik over de forskellige nøgleområder i vores virksomhed. Vi har udarbejdet i diagram, som er vedlagt som **bilag**, men vi vil også gennemgå de forskellige ting kort her.

Et BMC består af følgende:¹

- **Partnere**
 - Hvilke partnere arbejder vi sammen med i vores virksomhed. Det kunne være i forbindelse med markedsføring, produktion eller lignende.
 - I vores tilfælde har vi kun identificeret én partner, New-Yorker.dk, som i dette tilfælde også er vores kunde. Vi betragter dem samtidig som en partner, da de bidrager med vigtig viden omkring branchen og produktet, som gør det muligt for os at udvikle app'en.
- **Aktiviteter**
 - Hvilke kerneopgaver løser vi som forretning.

¹ [Bilag1](#): Business Model Canvas

- Vores virksomhed specialiserer sig i app udvikling, med fokus på Java og Android. I det her tilfælde laver vi specifikt en app rettet mod virksomheder der leverer New Yorker vægge.

- **Ressourcer**

- Hvilke ressourcer virksomheden har til rådighed. Det kan for eksempel være økonomiske, fysiske eller bemanding.
 - Vi har en bemanding på 3 personer, hvor vi fordeler opgaverne ligeligt. Økonomisk arbejder vi under studiet gratis og vi arbejder fra vores hjemmearbejdsplads, så vi ikke har nogle fysiske eller økonomiske ressourcer direkte til rådighed. På sigt vil det dog ændre sig.

- **Kundeværdi**

- Hvad er det for en ydelse vi sælger, som giver en værdi for kunden.
 - Vi sælger vores tid og specialisering i softwareudvikling og derigennem hjælper vi kunden med at løse deres behov.

- **Kunderelation**

- Hvordan står vi til rådighed for kunderne.
 - Vi sørger for at holde tæt kontakt til kunden i udviklingsprocessen. Det sker både personligt med jævnlige møder og elektronisk via mail. Derudover står vi til rådighed efter at produktet er leveret så vi kan tilføje yderligere funktionalitet og bistå med fejlrettelse.

- **Kanaler**

- Hvordan sælger vi til kunderne.
 - Vi laver opsøgende salg og demonstrerer vores idé og produkt, enten fysisk eller ved online møder med kunderne. På den måde kan vi hurtigt vise kunden den værdi vi skaber og der kan åbnes dialog om eventuel tilpasning og yderligere ønsker fra kunderne.

- **Kunder**

- Hvem er vores kunder.
 - Vi har identificeret vores kunder som producenter af New Yorker vægge, deres forhandlere og deres kunder. Det vil være dem, som primært benytter den app vi er ved at udvikle.

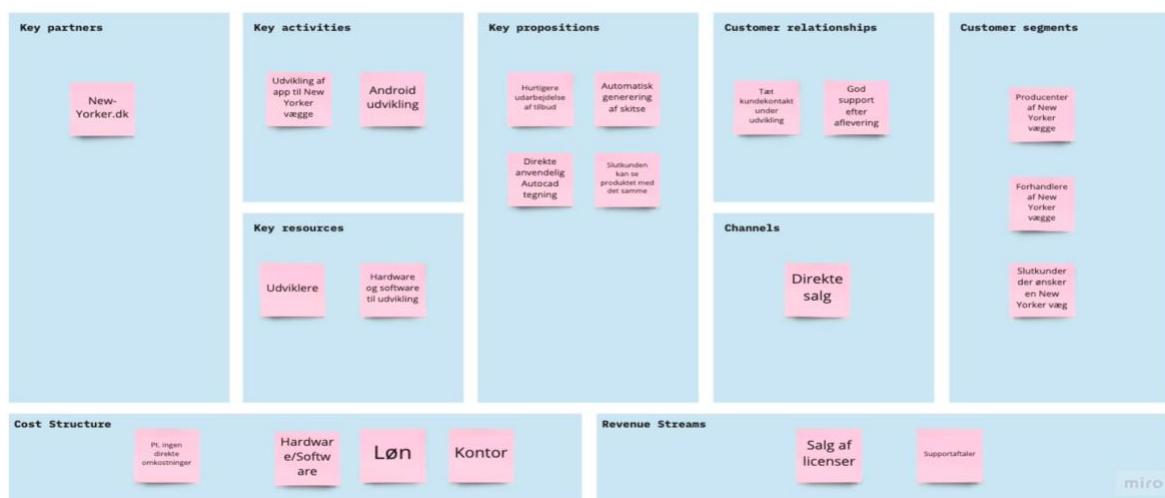
- **Omkostninger**

- Hvilke omkostninger har vi identificeret.
 - På nuværende tidspunkt har vi ikke nogen direkte omkostninger i forbindelse med virksomheden, da den kører sideløbende med vores uddannelse. På sigt, vil der dog opstå omkostninger i forbindelse med leje af kontorpladser, lønudgifter, licensering af software til udvikling, indkøb af forskelligt

kontormateriel, hardware og så videre. Det vil dog først være efter endt studie og kan indfases løbende efter behov.

- **Indtægter**

- Hvordan har vi tænkt os at tjene penge på produktet.
 - Vi tænker på sigt at udvikle et standardprodukt, som kan tilpasses til de enkelte producenter af New Yorker vægge. Det kan vi sælge både som et enkelt produkt, hvor kunden efterfølgende overtager ansvaret for drift og opdatering, eller med en licensering hvor vi kan bistå med holde løsningen kørende.



Business Model Canvas

Finansiering

Vi har vedlagt et investeringsbudget², hvor dykker nærmere ned i detaljerne men ellers er den overordnede tankegang som følger:

- **Indtægter**

- Vi forestiller os at kunne lave et generisk stykke software, som kan tilpasses de enkelte New Yorker væg producenter der findes på markedet. Der har vi sat en fast pris på 50.000 kr. for køb af licens og tilpasning til den enkelte kunde.

Derudover sælger vi vedligehold (support, opdatering af priser og lignende) for en fast månedlig pris på 50 kr. pr. bruger hvor vi har sat det forventede antal brugere til 30 pr kunde, som dækker interne brugere samt eventuelle sælgere. Vi forventer at sælge én licens i det første halvår og herefter et langsomt stigende antal over tid.

² [Bilag 2](#): Virksomhed finansiering excel ark

- **Udgifter**

- Da vi fortsat er studerende og virksomheden derfor vil være et projekt ved siden af studiet, kan vi holde vores indledende omkostninger lave. Vi har ikke nogen udgifter til husleje eller løn under studiet og derfor er vores andre administrative omkostninger også lave. Til gengæld vælger vi at sætte penge til side til investeringer, som kan føres tilbage i virksomheden når der er behov. Vi forventer at starte med en forholdsvis lav løn, hvor vi har tilsidesat 40.000 pr. ansat, som også skal dække sociale omkostninger såsom pension, feriepenge og lignende.

- **Opstartslån**

- Vi forventer at kunne starte med en mindre kassekredit, da vi allerede har materiale i form af laptops og lignende rent privat, som vi kan starte med. Vi har derfor ikke behov for de store omkostninger i forbindelse med opstarten af virksomheden.

Alle er de angivne tal er naturligvis overslag og der må forventes at være afvigelser i forhold til det realiserede. Men vi føler ikke at det er urealistisk prissat, når man ser på den tid der indledningsvis skal lægges i udviklingen af app'en.

Implementering

Vi forventer at implementering af appen vil foregå let, da New-Yorker.dk er en tæt samarbejdspartner i hele udviklingsprocessen. Vi regner med at der vil ske en overlevering af app'en og vi så vil bistå med oplæring og levere dokumentation på hvordan app'en skal benyttes. Den burde dog være udviklet så brugervenligt, at der ikke er behov for den store support.

Skalering

Vi har undersøgt hvordan andre producenter af New Yorker vægge modtager tilbud, og her kan vi se at der er en stor del af dem som på nuværende tidspunkt kun har en kontaktformular.

Vi mener derfor det er oplagt at tage kontakt til disse firmaer efterfølgende og demonstrere hvad vi kan gøre for, at hjælpe dem med på en lettere måde at lave tilbud til deres kunder og hjælpe deres sælgere i salgsprocessen. Vi mener, at vi på den måde langsomt kan vækste og tage flere kunder ind.

Som vi tager flere kunder ind, kan vi forvente at der vil være en større og større arbejdsbyrde der relaterer sig til support og vedligehold af den leverede app, så vi må på sigt forberede os på at skulle lave supportydelser.

Metodevalg (Jenna)

Udviklingsproces

Vi har valgt at gå til projektet med en iterativ udviklingsproces. Hermed menes, at vi arbejder i ugentlige iterationer hvor vi hver uge udvælger de dele som vi ønsker, skal være gennemarbejdet i denne iteration. Når ugen er gået runder vi af og gennemgår de opgaver vi har løst. På den måde sørger vi for, at vi har fået alt det planlagte og det giver os mulighed for at planlægge næste uges iteration.

Vi har også valgt at starte hver dag med et kort møde hvor vi taler om hvad vi hver især skal fokusere på i løbet af dagen. Når dagen er ved at være slut, holder vi igen et kort møde hvor vi ser på hvad vi har fået og hvilke udfordringer vi har haft.

Det har især været værdifuldt når vi nu på grund af corona, har arbejdet på hjemmefra. Vi har derfor sørget for, at vi er tilgængelige og kan tage fat i hinanden via Zoom og Discord hvis der er problemer eller spørgsmål.

Som en del af udviklingsprocessen sørger vi samtidig for, at reviewe hinandens kode og dokumentation. Når et stykke kode eller dokumentation er færdigudviklet, bliver det lagt i Github og her sørger en anden i gruppen for at gennemgå og validere det, inden det bliver lagt i repository. Dermed sikrer vi kvaliteten og minimerer risikoen for fejl.

Grunden til at det er en god ide at dele udviklingsprocessen op i iterationer er, at vi så sikrer flere små leverancer, både af kode og dokumentation, i stedet for én stor leverance. Dette kan bruges til, at demonstrere dele af vores program for kunden løbende og dermed sikre at vi stadig er på rette spor. På den måde kan vi hurtigt rette til hvis der er eventuelle misforståelser i forhold kunden og vi er derfor langt mere sikre på at vi det vi udvikler er det som kunden ønsker.

Projektstyring

- **Planlægning**

En anden fordel ved den iterative proces er, at vi får et projekt som er langt mere overskueligt at gennemføre. Dermed sikrer vi os bedre imod skred i tidsplanen og kan hurtigt rette til i tilfælde af uventede problemer.

Her kommer projektstyring ind i billedet og den måde vi har valgt at gribe det an er så simpelt og overskueligt som muligt på ved hjælp af Github Project og excel ark:

- Overordnet projektplanlægning og tidsplanlægning
- Github Projekt til enkeltopgaverne

Vi har lavet en overordnet projektplan,³ som vi bruger til at planlægge vores iterationer ud fra. På den måde har vi et overblik og vi kan nemt flytte rundt på ting som projektet udvikler over tid og tilpasse iterationer.

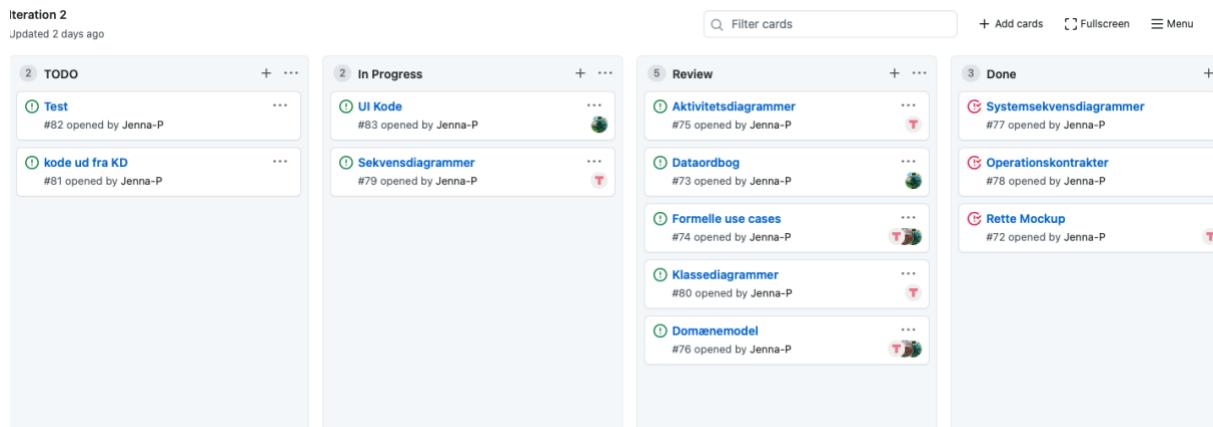
I den daglige udvikling har vi så en løbende tidsregistrering, som lægger sig op af den enkelte iteration. Her har vi lavet en et indledende estimat på hvor lang tid den enkelte opgave vil tage samt hvilken dato vi forventer den er færdig. Når vi så har løst opgaven, flytter vi den i Github Project og opdaterer arket med den tid vi har brugt samt dato for færdiggørelse. På den måde har vi løbende styr på tidsforbruget og vi kan reagere hurtigt hvis tidsplanen skrider.

Grunden til at vi har valgt en kombination af Github Project og excel ark er, at det er let og overskueligt at benytte og at andre projektstyringsværktøjer, som for eksempel Microsoft Project, typisk har en større indlæringskurve samt at det koster penge. Vi har derfor valgt at holde det simpelt og føler at projektet fint kan bære det.

Tidsregistrering_gruppe5_Simon, Mikkel og Jenna											
TODO	Opfølgning	Dato	Simon	Mikkel	Jenna	Estimeret antal timer	Estimeret startdato	Estimeret slutdato	Faktisk antal timer	Faktisk startdato	Faktisk slutdato
Systemsekvensdiagrammer					Jenna	1	20/05	20/05	1	20/05	20/05
Operationskontrakter					Jenna	3	25/05	25/05	3	25/05	25/05
Sekvensdiagrammer					Jenna	1	20/05	20/05	2	20/05	21/05
Klassediagrammer					Jenna	1	24/05	25/05	1	24/05	25/05
Dataordbog						25/05					
kode ud fra KD			Simon		Jenna	16	25/05	26/05	16	25/05	28/05
Test			Simon	Mikkel		4	28/05	28/05			
UI Kode					Mikkel	30	25/05	28/05	30	25/05	28/05
Rapport		28/05									
Iteration 2 31/05-06/06											
rette Mockup					Jenna	1	27/05	27/05			
Dataordbog				Mikkel		1	01/06	01/06	2	01/06	01/06
Formelle use cases			Simon	Mikkel	Jenna	1	31/05		3	01/06	01/06
Aktivitetsdiagrammer					Jenna	1	31/05	31/05	1	01/06	01/06
Domænemodel			Simon	Mikkel	Jenna	1	01/06	01/06	1	01/06	01/06
Systemsekvensdiagrammer			Simon			1	01/06	01/06	1	01/06	01/06
Operationskontrakter			Simon			3	01/06	01/06	3	01/06	01/06
Sekvensdiagrammer					Jenna	2	02/06	02/06	3	02/06	02/06
Klassediagrammer				Mikkel	Jenna	2	02/06	02/06		02/06	02/06
kode ud fra KD					Jenna	4	02/06	06/06		02/06	
Test					Jenna						
UI Kode					Mikkel	Jenna	30	02/06	06/06		02/06
Rapport		04/06									

Overordnet projektplanlægning og tidsplanlægning

³ [Bilag 3](#): Projektplan excel ark



Github projekt

● Risikoanalyse⁴

Som en del af projektstyringen har vi også kigget på de risici vi føler der kan være i projektet og lagt det ved som et bilag i en risikoanalyse.

Grunden til, at vi har gjort det er, så vi kan være på forkant med eventuelle problemer og tage højde for dem i vores planlægning. Det er vigtigt, at vi fra starten af har overblik over hvad der kan gå galt, da det gør at vi kan komme i mål med projektet med større sikkerhed.

Et godt eksempel på en risiko er sygdom i gruppen. Der vil altid være mulighed for, at der er enkelte dage hvor vi ikke alle kan arbejde og at vi derfor kommer bagud i forhold til planen.

Vi har derfor forsøgt at planlægge efter, at det sker og lagt en buffer ind i vores projektplan.

Risiko analyse

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Sygdom blandt udviklerne	4/10	Tidstab til review af input(4/10)	lav	1/10	Stadig tidstab, men bedre oversigt (3/10)

Imødekommedes strategi:

Der skal tages højde for et vist antal sygedage i tidsplanen, et normalt estimat er 5% fravær. Planen er lavet så der er taget højde for sygdom og vil derfor kunne gennemføres hurtigere uden sygdom.

Eksampel på en risiko

Github

Vi benytter som nævnt Github som et understøttende værktøj i den iterative udviklingsproces. Det er et fantastisk værktøj, som både fungerer som repository til vores kode og understøtter vores valgte udviklingsproces. Her er en kort gennemgang af de funktioner vi har valgt at bruge i Github:

⁴ [Bilag 4](#): Risikoanalyse

- **Code**

Her gemmer vi hele vores projekt, både den dokumentation vi laver, vores rapport, samt den udviklede kode. Det gør det muligt at arbejde sammen omkring hele projektet og at validere hinandens bidrag.

- **Pull Requests**

Når vi har udviklet enten et stykke kode, eller en del af vores rapport, opretter vi et pull request. Det gør det muligt for de andre i gruppen at gennemgå bidraget og kommentere på det, inden det bliver en del af projektet. På den måde kan vi rette fejl før vi tager kode ind og minimere risikoen for problemer.

- **Issues**

Vi opretter i første omgang udestående opgaver i To-Do. Herefter kan de konverteres til et Issue og så arbejder vi på dem her. Typisk ville man oprette bugs eller andre problemer her, men vi har valgt at benytte funktionen lidt bredere.

- **Projects**

Projects favner de enkelte iterationer. Vi planlagde indledningsvis at lave ugentlige, men ændrede det til to-ugers iterationer. Når en iteration er udløbet, sørger vi for at oprette et nyt projekt i Github og navngive projektet efter den nye iteration. Så opretter vi de forskellige trin og genopretter eventuelle opgaver fra tidligere iteration, som ikke blev løst.

Hver enkelt iteration er opdelt på følgende måde:

Den indeholder et board hvor vi opretter iterationens opgaver. Dette board er underdelt i flere kolonner som skal sørge for at holde overblikket over iterationens backlog, hvem der laver hvad og hvor meget vi har nået. Vi har opdelt boardet i følgende kolonner:

- **To-Do**

- Iterationens backlog. Her oprettes alt ved iteration start og det er her gruppens medlemmer tager deres opgaver.

- **In-progress**

- Her kan man se de opgaver som er ved at blive løst. Opgaven flyttes og tildeles en enkelt person i gruppen.

- **Review**

- Når man er færdig med at løse en opgave, flytter man den til review kolonnen og tildeler en anden person fra gruppen. På den måde får vi altid flere øjne på opgaven og minimerer risikoen for at der opstår. Det hjælper også til at vi altid følger vores aftalte kodenstandarder.

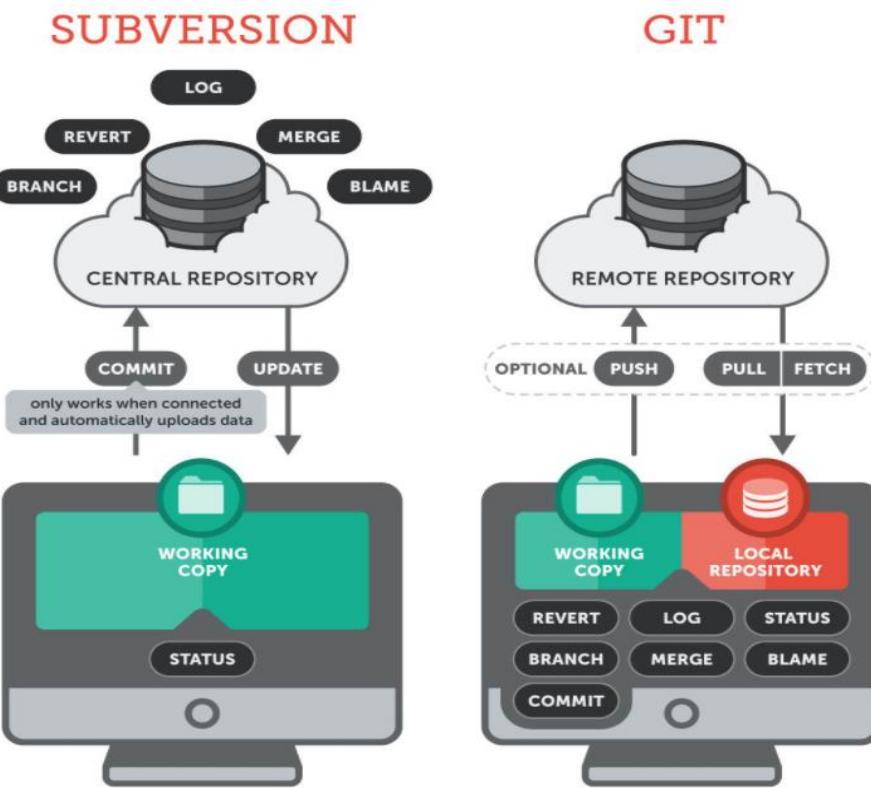
- **Done**

- Efter at opgaven er løst og har været igennem review processen bliver flyttet til denne kolonne. Her kan vi så følge med i hvor meget vi har nået i løbet af ugen.

Git, versionskontrol - og forskellen til Github

Git blev udviklet i 2005 og er langsomt steget i popularitet, til nu at være det mest benyttede versionskontrol værktøj. Styrken ved Git er, at det benytter sig af en model hvor man har et remote- og flere distribuerede repositories. Det betyder kort sagt, at hver enkelt udvikler har en kopi af den samlede kode lokalt, som man arbejder i og herefter committer tilbage til det remote repository. På den måde kan hver enkelt udvikler arbejde offline og i sin egen kopi indtil man er klar til at sende sin kode tilbage. Det har også den fordel, at hvis det remote repository går i stykker har hver enkelt udvikler en backup af det så det kan retableres nemt. Derudover påvirker nedetid ikke udviklingsprocessen, da man blot arbejder videre lokalt til det remote repository er oppe og køre igen.

Dette er i modsætning til andre versionskontrolsystemer, såsom Apache Subversion (SVN), hvor man har en klient-server model. Her har man en server, der hoster ens repository og herfra kan man hente branches ned lokalt og arbejde på dem. Ulempen er dog, at man så har et single-point-of-failure. Hvis ens repository er korrupt, er det sværere at reparere og da man kun henter mindre dele af koden ned lokalt af gangen, gør det også udviklingsprocessen mere besværlig i tilfælde af at serveren er nede eller man arbejder offline.



(kilde: <https://securityonline.info/difference-git-svn/>)

Generelt om versionskontrol kan siges, at det benyttes til at sikre sig, at man ikke mister kode, håndtere at flere udviklere kan arbejde på samme funktionalitet samtidig og at man kan rulle tilbage til tidligere versioner hvis man finder fejl. Det understøtter Git til fulde og det er en virkelig vigtig del at have styr på i ethvert softwareudviklingsprojekt.

Og hvad er Github så? Det korte svar er, at det er en cloud baseret hosting service til ens Git repositories. Det er her man uploader den centrale kopi af koden og her udviklerne så kan hente en lokal kopi ned fra og arbejde videre på. Man kan så benytte forskellige plugins i ens udviklingsværktøjer til at forbinde til Github med, i vores tilfælde har vi benyttet VCS funktionen i Android Studio.

Servicen startede tilbage i 2008 og har langsomt tilføjet en bredere funktionalitet, der som beskrevet tidligere gør det muligt at styre ens udviklingsprojekt og følge op på problemer i koden.

Derudover har det vundet stort indpas i open-source miljøet, da det er nemt at dele kode med andre og samarbejde på udviklingen - samt at man kan få langt det meste funktionalitet helt gratis så længe ens repository er public. Det er først når man som virksomhed vil benytte det, at det kommer til at koste penge.

Environment

Kunden har valgt, at de ønsker en app udviklet til Android telefoner. Dette sætter nogle naturlige valg og begrænsninger for den teknologi som vi kan benytte i forbindelse med planlægningen og løsningen af projektet. Vi har derfor valgt følgende:

- **Android Studio**

Da kunden ønsker en app der fungerer på Android telefoner har vi valgt at udvikle den i Java og ved hjælp af IDEen Android Studio. Vi føler at dette giver os den bedste mulighed for succes i det givne scenarie.

- **Github**

Vi har valgt, at benytte Github da det kombinerer muligheden for et repository til vores kode samt et overblik over vores ugentlige iterationer og daglige opgaver. Vi føler det letter vores hverdag at have kode og opgaver tæt på hinanden.

- **Adobe XD**

Vi mener at det er et godt produkt til at vise mock-ups til kunden af vores app. Valget faldt på dette da vi har benyttet det med stor succes tidligere.

- **Google Docs/Sheets**

Vi har valgt at bruges docs/sheets da vi her nemt kan arbejde sammen omkring dokumenter. Både til rapportdelen, men også i forbindelse med dokumentation af selve udviklingen.

- **Visio**

Visio fungerer rigtig godt til at visualisere processerne i forbindelse med systemudvikling. Vi har derfor valgt at benytte dette til alle vores diagrammer i rapporten.

- **Google Play**

Derudover betyder valget af Android, at app'en vil skulle lægges på Google Play. Det er Googles platform til distribution af apps på Android og det vil gøre det muligt for brugeren at downloade og installere appen direkte fra deres telefon.

For at vi kan lægge vores app på Google Play kræver det at vi har en udviklerkonto, samt at vores app pakkes som en APK-fil (Android Package). Det er det format på en installationspakke som Android telefoner bruger og det kan vi lave direkte i Android Studio.

Det er også muligt at dele og installere APK filer udenfor Google Play. Det kræver dog at brugeren aktiverer muligheden på sin telefon og vi føler det vil give en dårligere brugeroplevelse.

Programmering (Jenna)

Android

Android er et operativsystem, som er udviklet til håndholdte enheder med touchscreen såsom tablets og mobiltelefoner. Det baserer sig på en Linux kerne og på andet open source-software. Det bliver udviklet af et konsortium kaldet Open Handset Alliance som består af forskellige mobiltelefonproducenter og er kommersielt, understøtter af Google.

Android er gratis og open source-software, hvor kildekoden er licenseret under “Apache License” (https://en.wikipedia.org/wiki/Apache_License), som giver alle mulighed for at benytte og modificere softwaren frit.

Det er muligt at hente frit og bidrage til “Android Open Source Project” på <https://source.android.com/> hvor man kan finde forskellig dokumentation omkring Android og selve udviklingsprocessen.

Fordi Android er open source, oplever man typisk at de forskellige mobiltelefonproducenter laver deres egne tilpasninger til styresystemet på deres telefoner. Man kan derfor sagtens få en forskellig brugeroplevelse alt efter hvilken producent man køber telefonen af, selvom det er samme styresystem telefonen benytter. På de fleste telefoner vil der dog stadig være de forskellige Google services til rådighed, såsom Google Play, Gmail og lignende. En undtagelse hertil er Huawei, som er underlagt en amerikansk handelsblokade der gør at Google pt. ikke kan levere disse services på telefoner produceret af Huawei, man skal derfor være opmærksom når man køber en telefon.

iOS

iOS er, i modsætning til Android, ikke open-source. Det er fuldt ud udviklet og kontrolleret af Apple og alle Apples mobiltelefoner og tablets kører derfor på en version af iOS som er specielt udviklet til dem. Denne kontrol giver den fordel for Apple, at de kan optimere deres software til deres produkter og samtidig giver det et fast regelsæt og en kontrol af alle de apps som er tilgængelige på platformen. Det gør til gengæld, at det kan tage længere tid for features at finde vej til iOS, samt at der er en lidt større barriere for udviklere.

Generel udvikling

Vores tankegang i forbindelse med udviklingen, har fra starten af været at vi ville have et userinterface, som var afkoblet fra de resterende klasser i programmet. Vi ønskede at opdele det så al databehandling og logik lå for sig selv og ikke som en del af brugergrænsefladen.

Vi har derfor opdelt vores kode i 3 områder:

- UI
- Model
- Logik

Forskellene på de 3 er at modelklasserne indeholder data, logik klasserne indeholder primært den logiske del og i nogle tilfælde data og til sidst UI-klasserne, som har med frontend delen at gøre. I model og logikklasserne har vi fokuseret på at udvikle ud fra vores klassediagrammer og den øvrige systemudvikling, så det i videst muligt omfang modsvarer det design vi er kommet frem til. Der har dog i løbet af udviklingen været tidspunkter hvor vi har måttet gå tilbage og rette vores design til, da det har vist at der var ting som ikke var muligt i praksis.

Et par eksempler på logik klasser i vores design er:

- CalculateOffer
 - Har ansvaret for at beregne tilbudsprisen
- WallLayout
 - Har ansvaret for at beregne antallet af ruder og fag

Et par eksempler på model klasser i vores design er:

- GlassTypes
 - Indeholder de forskellige glastyper som kan vælges
- DoorTypes
 - Indeholder de forskellige dørtyper som kan vælges

Vi har ikke fulgt et bestemt design pattern i vores udvikling, såsom MVC eller MVVM, men vi har dog haft det i baghovedet og det er en ting som vi vil prøve at komme tættere på i et senere projekt. Et eksempel på det er vores controller klasse, NewYorkerController, som ikke står for kommunikationen mellem UI og model/logik klasserne som det ellers ville være tilfældet i MVC.

Persistering af data i Android

Til at gemme data i vores Android app har vi valgt at benytte en database. Valget her faldt på SQLite, da vi tidligere har benyttet det i andre projekter og vi derfor følte det var let at gå til. Som udgangspunkt prøvede vi at gemme vores via en standalone SQLite database som vi forbandt til via JDBC.

Det viste sig dog at skabe problemer for os, så vi blev nødt til at ændre det til den SQLite database som er direkte indbygget i Android. Vi måtte derfor lave vores kode om så det matchede og herefter kunne vi som planlagt gemme data som ønsket.

Unit testing af kode

Til test af vores Android kode har vi benyttet os af jUnit4. jUnit4 er et framework, som kan bruges til at lave unit-test i Java og det udgør en vigtig del af test drevet udvikling.

Ideen med test drevet udvikling er, at man når man implementerer en ny funktionalitet faktisk starter med at lave en test, som skal verificere resultatet af den nye funktionalitet inden den er udviklet. På den måde bliver man tvunget til at fokusere på kravene inden man udvikler og man ender at have en masse tests, som kan bruges til at verificere at funktionaliteten virker.

Vi ikke har benyttet test drevet udvikling, men har i stedet gået direkte i gang med funktionalitet og bygget tests på efterfølgende. Vi følte os lidt usikre i processen og derfor valgte vi at udvikle mere traditionelt.

Threads

Vi har ikke benyttet os af threads i vores program. Vi overvejede at implementere mail afsendelsen ved brug af JavaMail API og her vil vi så benytte threads, da det ville give mening at bruge AsyncTask i den forbindelse. Her kan flere tråde håndtere mailafsendelse i baggrunden samtidig med at det håndterer UI. Vi løb dog ind i det problem at AsyncTask er deprecated i API level 30, så vi skal finde en anden løsning. Det er noget vi håber på at arbejde videre med efter afleveringen.

Procesbeskrivelse (Jenna)

Når man går ind til et sådant udviklingsprojekt, er det vigtigt at man har en god overordnet proces for hvordan vil komme i mål. Det er vigtigt, at man allerede fra starten gør sig det helt klart, hvilke ting man skal opnå og i hvilken rækkefølge det skal ske, da man ellers forhøjer risikoen for at projektet fejler. Som nævnt tidligere har vi fulgt en iterativ udviklingsproces, og denne fremgang har vi også fulgt rent planlægnings- og designmæssigt. Her er en kort oversigt over hvordan vores forløb har været:

Iteration 1

- Uge 1
 - Kundemøde og indsamling af krav
 - Udspecifcering af krav
 - Indledende design
 - Udarbejdelse af indledende diagrammer, dokumentation og rapport
 - Udarbejdelse af mock-up af app'en
 - Præsentation og feedback fra kunden
- Uge 2
 - Tilretning af mock-up, krav og design ud fra feedback fra kunden
 - Påbegyndelse af softwareudvikling ud fra designet
 - Videre forfining af diagrammer og dokumentation
 - Videre arbejde på rapport
 - Præsentation af program for kunden

Den første iteration er på mange måder den vigtigste, da den lægger hele fundamentet for resten af projektet. Det meget vigtigt, at vi får indhentet de korrekte krav fra kunden og omsat dem til et design som vi kan vise frem og få feedback på. Denne proces gør, at vi hurtigt får sikret os at vi har forstået kunden korrekt og kunden føler sig tryg ved, at det hurtigt bliver håndgribeligt.

Vi har bevidst valgt ikke at gå i gang med programmeringen i den første del af iterationen, da vi gerne ville have styr på design og krav, samt fremvist det for kunden inden. Dette har vi gjort for ikke at spilde unødig tid såfremt vi havde misforstået noget.

I anden del af iterationen gjorde vi det meget mere håndgribeligt og påbegyndte programmeringen. Vi afsluttede det med at fremvise det vi havde fået til kunden og fik feedback.

Indledningsvis var planen at iterationerne skulle være en uges tid, men på grund af feedback og at vi ikke kom helt rundt om de ting vi gerne ville, valgte vi at forlænge dem.

Iteration 2

- Uge 3
 - Tilretning af mock-up, krav og design ud fra feedback fra kunden
 - Videreudvikling af software ud fra det tilrettelæggede designet
 - Videre forfining af diagrammer og dokumentation
 - Udvidelse af use cases - tilføjelse af extensions
 - Videre arbejde på rapport
 - Præsentation af program for kunden
- Uge 4

- Færdiggørelse af rapport
- Sluttilretning af software i forhold til sidste kundemøde
- Afgrænsning af krav vi ikke kan nå at få med i den endelige aflevering af software
- Aflevering af rapport og software

Den anden iteration blev som den første strakt over 2 uger, i stedet for en uge som vi tænkte til at starte med. Vi fandt ud af at det passede vores måde at arbejde på godt og det holdt også vores iterationer sammenlignelige.

Denne iteration har handlet meget om 3 ting; videreudvikling af vores program, finpudsning af tilhørende diagrammer i forhold til kundens ønsker, samt arbejde på vores rapport.

I softwareudvikling delen har vi fokuseret meget på vores UI ([android udvikling](#)) samt at få selve tilbud flowet i appen til at fungere som kunden ønskede. Det har været vigtigt at sørge for, at vores tilbud bliver korrekt udregnet og at vi får designet af væggen udregnet med det korrekt antal ruder og fag. Derudover har fokus været på at få et katalog til at virke samt at starte mail delen.

Eftersom vi har lavet ændringer i vores program, har vi til dels også måttet tilpasse vores diagrammer, da alt ikke altid har vist at virke som vi forestillede os. Derfor har vi løbende måtte lave tilretninger rundt omkring, så vi har vores dokumentation på plads.

Rapporten har vi løbende arbejdet på igennem projektet, så vi har sørget for at hele arbejdspuklen ikke ligger til sidst. Vi har stadig ting der skal gøres, men det er delvist kommer på plads og planen er at være færdig 1-2 dage før aflevering så vi har tid til finpudsning og yderligere kodning.

Iteration 3

Vi har konkluderet, at vi ikke når til iteration 3 og har i stedet fokuseret på at nå de ting som vi havde planlagt fra starten, så vi kan aflevere et produkt der fungerer, men som ikke har den fulde funktionalitet fuldt implementeret. Vi har vurderet at vi må flyttet følgende til iteration 3:

1. Færdiggørelse af mail afsendelse med Java mail API
2. Generering af fil der kan åbnes i Autocad
3. “Kurv” funktion så kunden kan konfigurere flere vægge i et tilbudsforløb

Det er ting som vi arbejder på efter aflevering og som vi håber at kunne vise noget af til vores eksamen. Vi vil arbejde på det i ovenstående prioriterede rækkefølge.

Kravindsamling og -specifikation (Jenna)

Problemdomæne

Inden man laver sin kravspecifikation, er det en god ide at se på sit problemdomæne. Man kan beskrive et problemdomæne som årsagen til at et stykke software skal udvikles samt den miljø/industri som det udvikles til. I vores tilfælde kan problemdomænet defineres som "Et stykke software der kan udregne specifikationerne for en New Yorker væg, vise et design, samt give et tilbud på denne ud fra de angivne mål og tilvalg."

Visionsdokument

Vi har udarbejdet et visionsdokument⁵ og vedlagt det som bilag. Formålet med dette dokument er at skabe et samlet overblik over hvad vi gerne vil løse, for hvem og med hvilke virkemidler. Det er opdelt i følgende 3 områder:

- **Vision**
 - Hvad er det for et problem vi gerne vil løse
- **Interessentanalyse**
 - Hvem er interesseret i en løsning af problemet og hvordan får de gavn af det
- **Featureliste**
 - Hvilken funktionalitet vil vi skabe for at understøtte løsningen

Dokumentet samler disse 3 områder og hjælper derfor med til at give et hurtigt og overskueligt overblik over hvad visionen er for projektet.

⁵ [Bilag 5](#): Visionsdokument

Visions Dokument

Vision

Produkterne henvender sig til New-Yorker.dk, som er en virksomhed der producerer New Yorker skillevægge. De ønsker sig en app som kan hjælpe dem med hurtigt og let at lave et tilbud på en skillevæg, blot ud fra indtastning af størrelse samt valg af tillæg. Formålet er, at lette processen både for forhandleren, som let og elegant kan lave et tilbud og vise kunden med det samme, samt for producenten, som vil få tilsendt et diagram over hvordan væggen ser ud dermed hurtigt kan komme i gang med produktionen.

Interessentanalyse

Brugere:

- Producenten ønsker en let måde selv at lave tilbud på, samt at modtage færdige tilbud fra deres forhandlere så de hurtigt kan komme i gang med produktionen.
- Forhandleren ønsker let og elegant at kunne vise kunden hvordan væggen ser ud samt at kunne give et færdigt tilbud.

Featureliste:

- Oprettelse af tilbud ud fra de angivne mål
- Tilpasning af væggen med forskellige materialer
- Visning af en færdig skitse af væggen ud fra mål og materialevalg
- Generering af skitse i autocad format
- Afsendelse af tilbud til producenten, forhandleren og kunden via mail

Visionsdokument

Use cases

• Casual

En casual use case⁶ benyttes i den indledende fase til, at give et hurtigt overblik over hvilke interaktioner en bruger har med systemet. En casual use case holdes typisk i ikke-teknisk sprog og kan udfyldes ved at stille opgavestilleren spørgsmål såsom, "hvad er det brugeren skal kunne", "hvilke roller er der", "hvad skal brugeren indtaste i systemet", "hvilken information skal systemet give tilbage" og så videre.

UC1: Lav et tilbud (iteration 1)

- Brugeren starter oprettelse af nyt tilbud
- Systemet viser indtastnings mulighederne for en ny væg
- Brugeren indtaster væggens højde og bredde.
- Systemet beregner antallet af lodrette ruder ud fra at højde skal ligge så tæt på 60 cm som muligt
- Systemet beregner antallet af fag, ud fra at bredden på ruderne skal ligge så tæt på 45 cm som muligt

Casual Use Case

• Formelle

Formelle use cases⁷ har et langt mere fast format og skal sikre, at man får afklaret alle de nødvendige ting. Vi har benyttet os af følgende skabelon til udfyldelsen af disse:

- Use Case Name

⁶ [Bilag 6](#): Casual Usecase

⁷ [Bilag 7](#): Formelle Usecase

- Navnet på Use Casen. Det bør starte med et udsagn, såsom "Lav...", "Gem..." eller lignende.
- Scope
 - Systemet som er under udvikling.
- Level
 - Indeholder normalt "user-goal" eller "sub function"
- Primary Actor
 - Hvem der benytter systemet
- Stakeholders and Interests
 - Hvem der har en interesse i denne use case.
- Preconditions
 - Hvad er forudsætningen for at denne use case kan benyttes
- Success Guarantee
 - Hvad skal være opnået efter denne use case kørt færdig
- Main Success Scenario
 - En gennemgang af et typisk scenarie for hvordan denne use case forløber succesfuldt
- Extensions
 - Beskrivelser af andre mulige udfald for hvordan denne use case kan forløbe
- Special Requirements
 - En liste over andre ikke-funktionelle krav
- Technology and Data Variations
 - Hvis der benyttes forskellige dataformater eller I/O metoder kan dette beskrives her
- Frequency of Occurrence
 - Hvor tit sker dette scenarie

- Miscellaneous
 - Andre åbne problemstillinger

Vi har vedlagt vores casual og formelle use cases som bilag til rapporten.

UseCase navn	UC Lav nyt tilbud
Scope	New Yorker
Primary Actor	Tømrer/Kunder
Level	User-goal
Description	Systemet skal kunne modtage angivelse af størrelse på den ønskede væg, udregne beregne et færdigt tilbud på væggen samt sende disse til kunden og producenten
Stakeholders and Interests	Byens Vinduer/New-Yorker.dk
Preconditions	Den android enhed er tændt. Applikationen er installeret og klar til at bruge.
Succes Guarantee	Der er oprettet et gyldigt tilbud og det er sendt til kunden og producenten
	1. Brugeren starter oprettelse af nyt tilbud. 2. Systemet viser indtastnings mulighederne for en ny væg. 3. Brugeren indtaster væggens højde og bredde.

Formel Use Case

Use case diagrammer

Formålet med et use case diagram⁸ er at give et overblik over hvilke interaktioner en bruger kan have med systemet. Det er generelt ikke detaljeret, så man skal ikke forvente at se en skridt-for-skridt gennemgang af det som systemet foretager sig, men i stedet en høj-niveau tegning af relationerne mellem de forskellige use cases, brugere og systemer.

Et use case diagram består typisk af følgende komponenter:

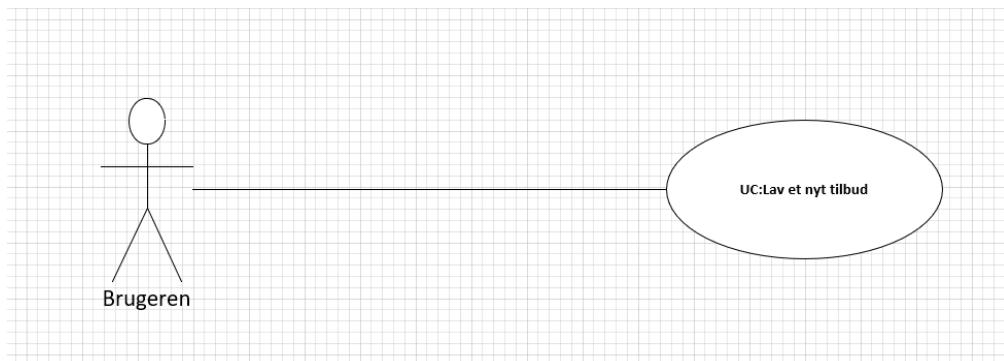
- Use cases
 - De forskellige interaktioner en bruger har med systemet.
- Actors
 - De forskellige aktører/brugere som interagerer med systemet. Primære aktører står til venstre og sekundære til højre i diagrammet. Systemaktører er angivet som en firkant og mennesker som en streg-figur.
- Associations

⁸ [Bilag 8](#): Usecase diagram

- Streger imellem aktører og use cases, som viser hvilke aktører der benytter hvilke use case.

- **System boundary boxes**

- Angiver scopet for systemet. Use cases som ikke relaterer sig til systemet, vil være uden for den firkant der angiver scope.



Use case Diagram

Supplerne Kravspecifikation (FURPS+)

9

Efter at have tænkt over sit problemdomæne, kan man gå lidt dybere og kigge på de enkelte krav. Her er det en god ide samtidig at tænke indledningsvis på de use cases som brugeren har uden nødvendigvis at skrive dem ned direkte. I stedet kan benytte sig at skabelonen for FURPS+, som giver en god ide om en de forskellige krav. De ting man beskriver i FURPS+ er som følger:

- **Functionality**

- Beskriver den funktionalitet som programmet skal indeholde for at løse de stillede krav

- **Usability**

- Omhandler typisk brugervenlighed og forskellige ønsker til user interfacet

- **Reliability**

- Beskriver ting som aftalt oppe tid, hvor let det er at genskabe i tilfælde af nedbrud og lignende

- **Performance**

- Omhandler den forventede ydelse, altså hvor hurtigt applikationen er

⁹ [Bilag 9](#): FURPS+

- **Supportability**
 - En beskrivelse af systemunderstøttelse, testmuligheder og lignende
- **Precision**
 - Omhandler hvor det forventes at applikationen er meget præcis i sine beregninger

FURPS+

Functionality

- Skal hjælpe brugeren, med at lave et tilbud.
- Skal have et design som følger sammen tema, som det der brugt på New-Yorker.dk's hjemmeside.
- Skal hjælpe producenten, med at lave en produktionsklar tegning.

Usability

- Applikationen skal være let at benytte, og skal støtte brugeren mest muligt, ved at præsentere alle tilvalg overskueligt.

Reliability

- Alle beregninger, og skitsering, foregår lokalt på telefonen for at sikre hurtig behandling.

Performance

- Udregninger af tilbud, samt udfærdigelse af skitse, skal ske hurtigst muligt, så brugeren ikke oplever ventetid.

Supportability

- Applikationen skal være understøttet på Android, som udgangspunkt. Det er dog muligt, at iOS skal understøttes på et senere tidspunkt.

Precision

- Alle målangivelser på skitserne skal være korrekt detaljerede, så de kan anvendes i produktionen.

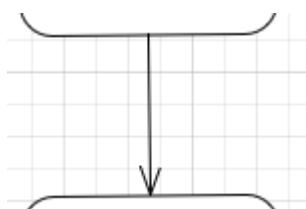
FURPS+

Object-Oriented Analyse (OOA) (Simon)

Aktivitetsdiagrammer

Et aktivitetsdiagram bliver brugt til vise rækkefølgen på forskellige aktiviteter i et program.

Her er betydningen af de forskellige tegn i et aktivitetsdiagram:



Streg: En streg med en pil, betyder at den næste aktivitet er der hvor pilen peger hen.



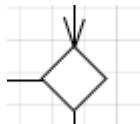
Sort prik: Den sorte prik betyder, at det er dér programmet starter, og det er programnets pre-condition.



Cirkel med sort prik indeni:

Dette tegn betyder at programmet er slut, og det er

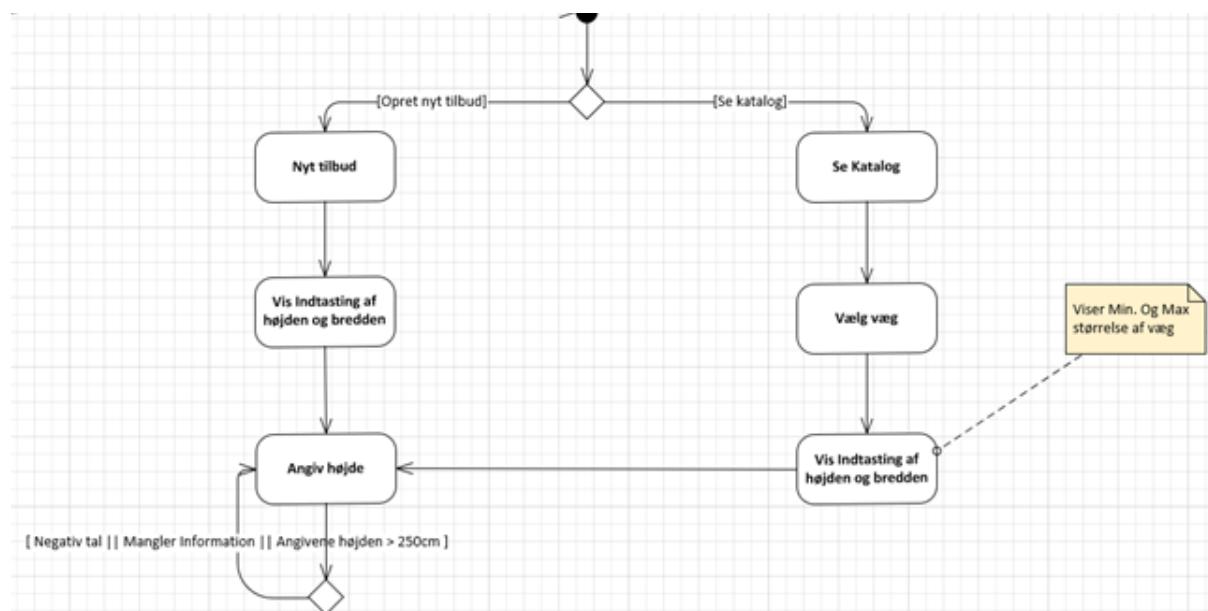
programmets post-condition.



Firkant:

Dette tegn betyder, at der er flere aktiviteter for brugeren at vælge imellem, (et alternativt flow). For hver streg ud af firkanten, er der en aktivitet.

Vi har lavet et aktivitetsdiagram hvor vi beskriver de aktiviteter en bruger af vores program skal gennemgå, for at lave et tilbud.



Aktivitetsdiagram

Ovenover er der et udsnit af toppen af vores aktivitetsdiagram.¹⁰

Programmet starter ved den sorte prik øverst, som er vores programs pre-condition. Derefter kommer der en firkant, der lader os gå to forskellige veje i programmet. Man kan vælge enten at oprette et nyt tilbud, eller vælge at se kataloget.

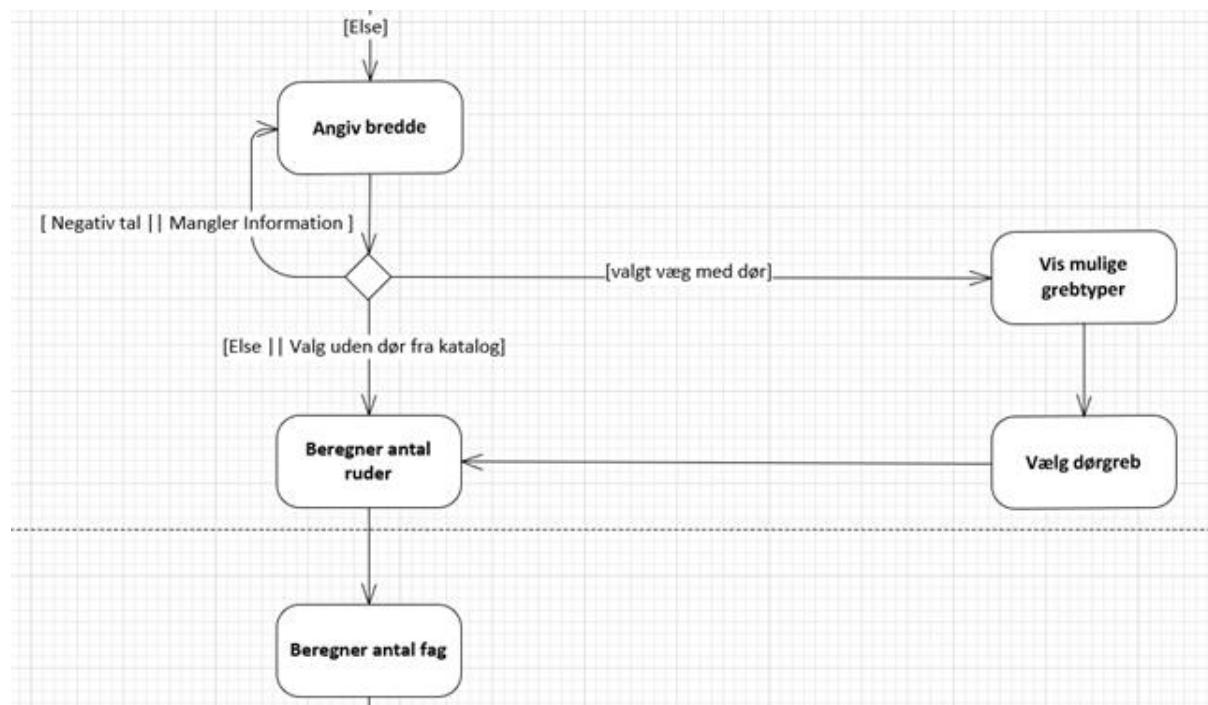
Hvis brugeren vælger at se kataloget, skal brugeren vælge en væg. Herefter får brugeren vist en skærm hvor de senere skal indtaste højde og bredde specifikationer på den valgte væg.

¹⁰ [Bilag 10](#): Aktivitetsdiagram_iteration1 og iteration2

Brugeren skal derefter indtaste højden på deres væg.

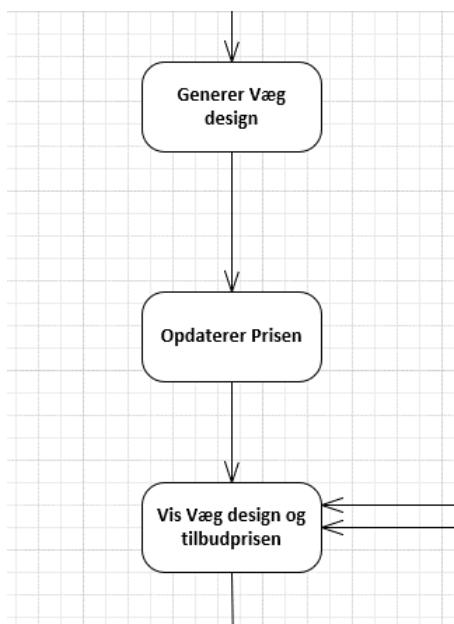
Tilbage til firkanten: Hvis brugeren vælger at oprette et nyt tilbud, vil derefter at brugeren trykker på knappen, komme en skærm op hvor de skal indtaste deres højde og bredde mål.

Brugeren indtaster højden, og systemet tjekker om højden er indtastet korrekt, da højden for eksempel ikke kan være negativ eller over 250 cm.



Brugeren skal nu angive bredde specifikationer. Ligesom ved højden, vil systemet bede brugeren om at indtaste bredden igen, hvis det er et negativt tal de har indtastet.

Hvis brugeren har valgt at have en dør med, bliver de nu vist de mulige typer af dørgreb, hvorefter brugeren bliver bedt om at vælge en. Derefter vil systemet beregne det antal ruder, og det antal fag der skal bruges ud fra de mål brugeren indtastede tidligere. Hvis brugeren valgte ikke at have nogen dør, så vil systemet ikke bede brugeren om at vælge en type dørgreb.



Efter systemet har beregnet antallet af fag, vil systemet generere et væg design, og derefter beregne et estimat på en pris. I næste step vil dette prisestimat og væg designet blive vist for brugeren.

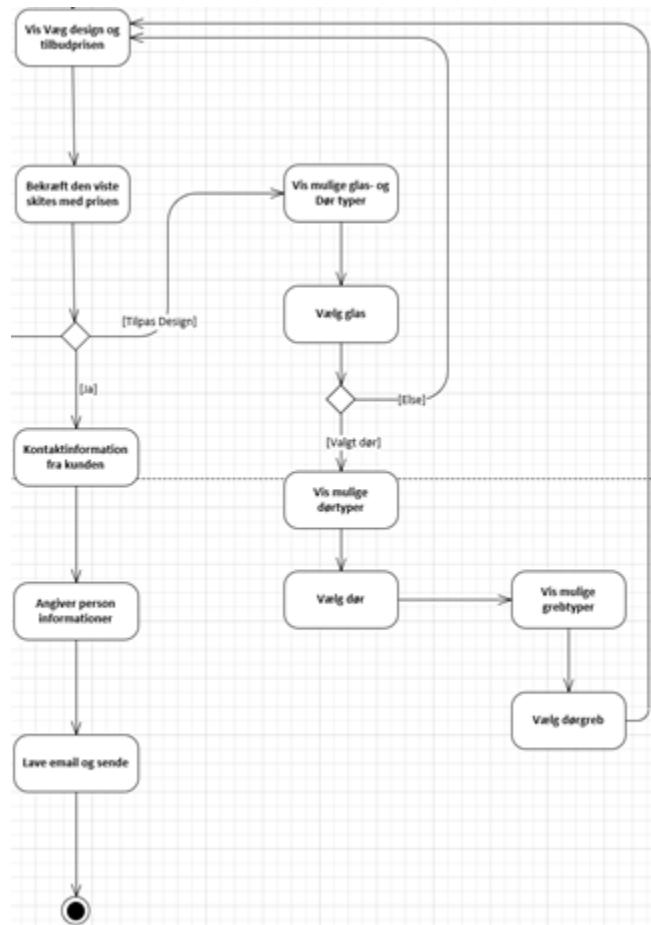
I udsnittet (fodnote 9), skal brugeren vælge om de vil bekræfte den viste skitse, og de har 3 valgmuligheder: 1. Brugeren kan vælge at tilpasse designet på væggen, 2. brugeren kan vælge at sige nej til den viste skitse, 3. brugeren kan vælge ja, og bekræfter skitsen.

Hvis brugeren vælger at tilpasse sit vægdesign, vil brugeren blive vist en skærm med de typer af glas og døre der er mulige at vælge imellem. Brugeren skal så først vælge en type glas, og derefter bliver brugeren få valget om de ønsker en dør. Hvis brugeren ikke vælger en dør, får de vist en skærm hvor brugeren kan se væg design og prisestimat.

Hvis brugeren i stedet vælger at have en dør i deres væg, vises der en skærm med de mulige dørtyper. Brugeren skal så vælge en dør. Brugeren får så vist de mulige typer af dørgreb, hvorefter de skal vælge et greb. Brugeren får nu vist en skærm hvor de kan se deres væg design og den estimerede pris.

Brugeren skal nu igen vælge om de vil bekræfte den viste skitse og estimat af pris. Hvis brugeren vælger at sige nej, og afviser den viste skitse, bliver de sendt hen til startskærmen, og de starter programmet forfra og kan lave et tilbud fra begyndelsen.

Hvis brugeren vælger ja, bliver brugeren bedt om at udfylde vigtig kontakt- og personinformation, hvorefter systemet vil lave en e-mail som vil sendes til forhandleren, og dermed er programmet kørt helt igennem.



Data Ordbog (Simon og Mikkel)

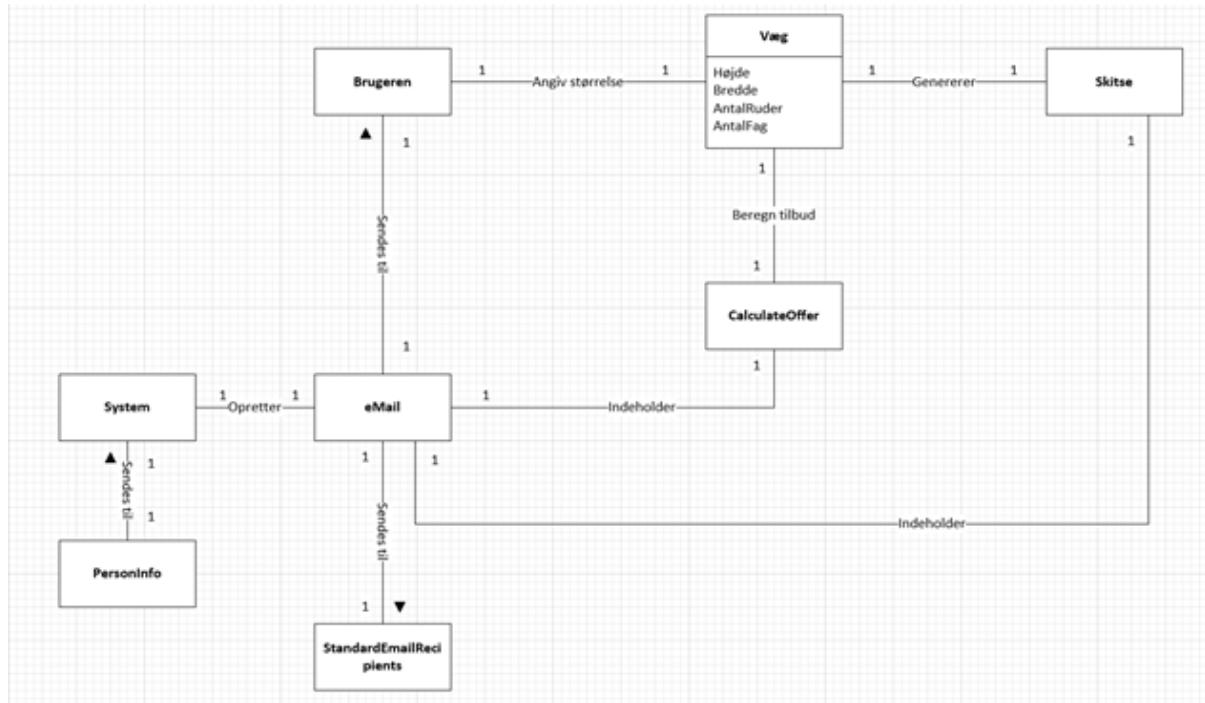
Begreb	Forklaring
New Yorker Væg	Er et produkt som firmaet New-Yorker.dk, en del af Byens vinduer, udvikler. New Yorker væggen består af en stålramme med glasruder og der kan tilføjes en dør.
Højde	Højden beskriver New Yorker væggens højde i centimeter.
Bredde	Bredden beskriver New Yorker væggens bredde i centimeter.

Fag	Et fag er en ramme, hvori man sætter en glasrude, så det bliver til et vindue.
Glas felt	Et glas felt er det tomme rum inde i et fag, hvor man sætter sin glasrude ind.
Tilbud	Et tilbud er en værdi sum, som en sælger vil give for sine varer. Dette er oftest betegnet med ordet ”pris”, men ”tilbud” bliver bedre modtaget, og virker mere tilbydende hos en kunde.
Glastyper	Glas kan blive fremstillet på flere forskellige måder, baseret på hvilke materialer, og metoder man bruger til at lave glasset. Man kan vælge mellem 3 typer af glas: Standard, satin glas og lyd glas
Dørtyper	Man kan vælge imellem to dørtyper: en dobbelt dør, eller en enkelt dør.
Dørgrebstyper	Dørgreb er tilgængelige i forskellige materialer og udseende. Fx kan et dørgreb blive lavet i kobber og stål.
Backend	Er alt hvad der foregår i programmet, som brugeren ikke kan se.

Frontend	Er alt hvad der foregår i programmet, som brugeren kan se, dette er mest UI.
UI	UI er kort for User Interface, og består af alt hvad brugeren kan se og trykke på.

Domæne model (Simon)

I vores første iteration af vores program har vi lavet en domænemodel¹¹, som illustrerer de konceptuelle klasser i vores program.



DM_LavEtNytTilbud_Iteration1(Tidlig Version af domænemodel)

I System klassen bliver for eksempel personlige informationer, som er sendt fra PersonInfo klassen, gemt.

Klassen Bruger er forbundet til den konceptuelle klasse Væg, hvor brugeren skal angive størrelsen på de fire attributter: højde, bredde, antal af ruder og antal fag, med en multiplicitet på 1 begge veje da 1 instans af Bruger kun kan blive associeret med 1 Væg instans.

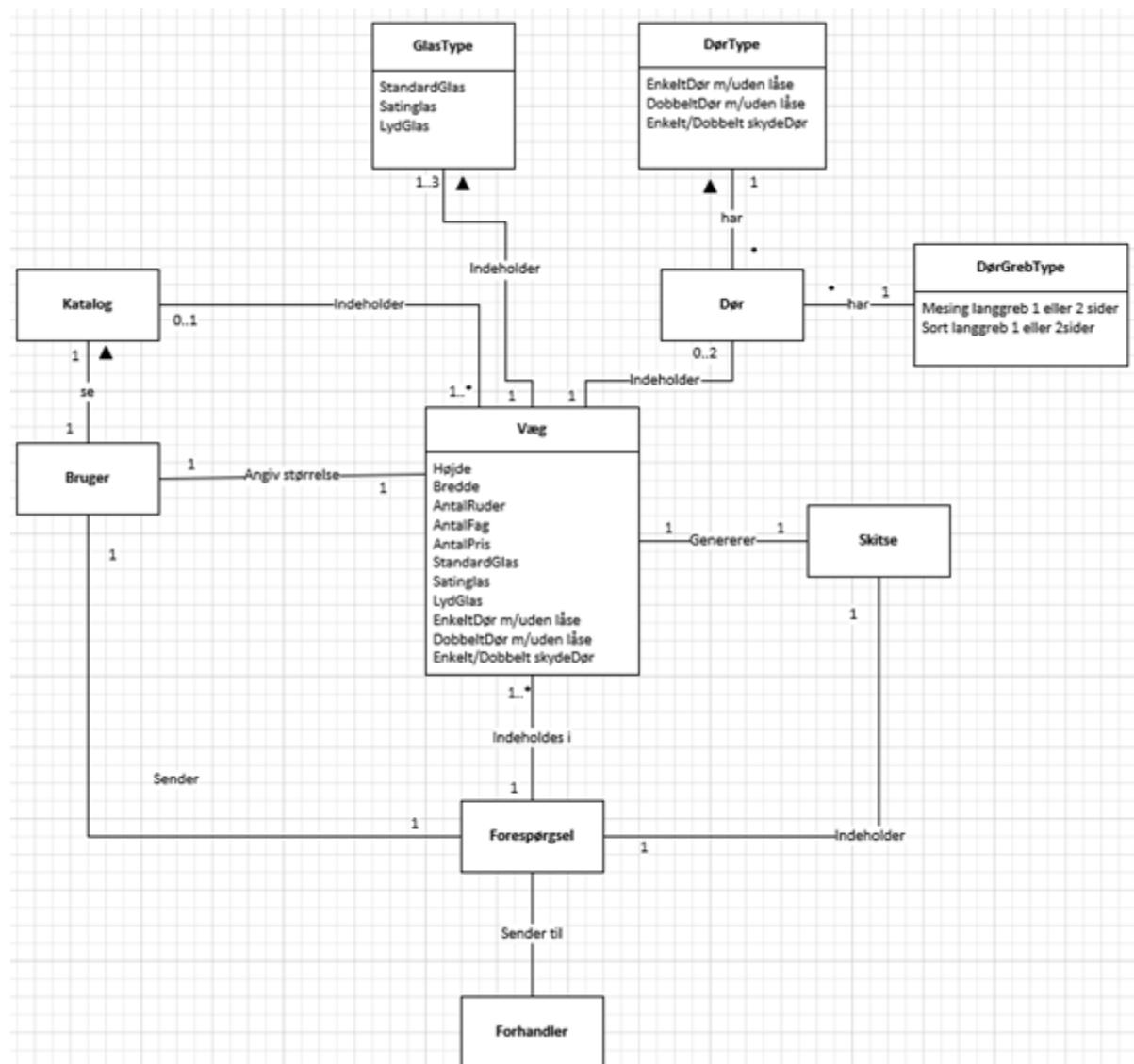
¹¹ [Bilag 11](#): Domænemodel_Iteration 1 og iteration2

Udover klassen Bruger er Væg klassen også forbundet med Skitse klassen. Skitse klassen vil generere en skitse ud fra hvad der er blevet indtastet i Væg klassen.

Væg klassen er også forbundet til CalculateOffer klassen, hvor i et tilbud på en pris kan blive beregnet, ud efter hvad Bruger indtaster af mål i Væg klassen.

eMail klassen er forbundet med flere forskellige klasser. Den er forbundet med: Bruger, CalculateOffer, Skitse, StandardEmailRecipients og til sidst System. Efter der er blevet generet en skitse og beregnet et tilbud, vil System klassen sende en besked til eMail klassen om at den skal oprette en email, der indeholder informationer fra CalculateOffer klassen og skitsen fra Skitse klassen. eMail klassen vil så sørge for at få sendt denne email til både brugeren, og til StandardEmailRecipients, som er forhandleren og producenten.

I den anden iteration af vores domæne model, lavede vi en del ændringer.



Vi fjernede flere klasser fra vores model, som eMail, System, Personinfo, StandardEmailRecipients og CalculateOffer, og har sat nye på til at erstatte dem. De ny klasser er Katalog, Forespørgsel, Dør, Dørtype, Dørgrebstype, Glastype og Forhandler.

Starter ved den mest centrale klasse Væg, som er forbundet med de fleste af klasserne i modellen. Vægklassen indeholder mange attributter, de er Højde, Bredde, AntalRuder, AntalFag, AntalPris, StandardGlas, SatinGlas, LydGlas, Enkelt-/Dobbelt-/SkydeDør, EnkeltDør med eller uden lås og DobbeltDør med eller uden lås.

Vægklassen er forbundet til Dør klassen, hvor 0-2 instanser er muligt, da det ikke er sikkert at brugeren vil have en dør, og måske vil de have mere end 1 dør. Hvis brugeren har valgt at have en dør, skal de også vælge en af de mulige dør typer, som hentes fra DørType klassen. Brugeren skal også vælge et dørgreb som hentes fra DørGrebType klassen, vælge hvilket materiale deres dørgreb skal laves af og om dørgrebet skal være på begge sider af døren.

Vægklassen er forbundet til GlasType klassen, hvor der er 1-3 instanser, siden man skal vælge 1 og da der findes 3 forskellige slags glas at vælge imellem.

Vægklassen er forbundet til Katalog klassen, med en multiplicitet på 1-mange fra væg, og 0-1 ved katalog.

Bruger klassen er forbundet til vægklassen, og det er gennem væg klassen at brugeren kan angive de forskellige størrelses specifikationer, hvilken type glas de har brug for og om brugeren vil have en dør i sin væg. Bruger og væg klasserne har et 1-1 forhold.

Bruger klassen er også forbundet til Kataloget, med et 1-1 forhold.

Den sidste klasse Bruger klassen er forbundet med, er Forespørgsel klassen, som sender en forespørgsel til brugeren, hvor i man kan finde en skitse og brugerens væg detaljer. De har et 1-1 forhold.

Skitse klassen bruges til at genererer en skitse af væggen, og Skitse klassen er derfor forbundet med Væg klassen. Der genereres kun 1 skitse pr væg, så forholdet imellem Skitse og Væg klassen er 1-1. Skitse klassen er også forbundet med forespørgsel, da forespørgslen indeholder en skitse af væggen. Her er forholdet også 1-1.

Forhandler klassen er den sidste klasse, den er forbundet med Forespørgsel klassen, og sørget for at forespørgslen bliver sendt til forhandleren.

Multiplicitet: Multiplicitet (tallene uden for klassen) i en domænemodel, bruges til at kommunikere hvor mange instanser kan blive associeret med hinanden lige på et hvilket som helst tidspunkt, og ikke over en større længde af tid. For eksempel forholdet imellem Væg og Skitse klassen, en instans af Væg klassen vil kun generere en instans af Skitse klassen, så derfor står der 1 ud fra de to klasser.

System Sekvens Diagram (SSD) (Simon)

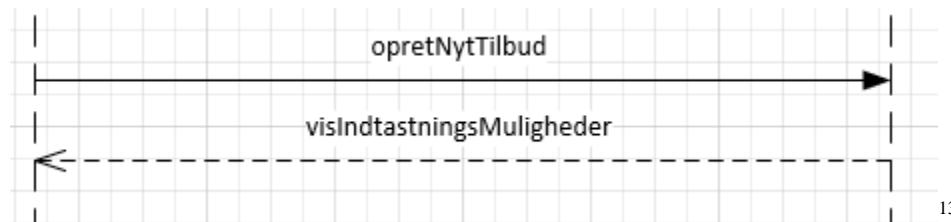
Et system sekvens diagram (forkortet til SSD)¹², er et nemt at lave og overskueligt artefakt der illustrerer de input en bruger laver i et system, og hvilket slags output der sker i følge af input.

Når man laver et program, er det en god ide at kunne forstå, hvordan en bruger af dit program vil interagere med programmet, for at få en bedre forståelse for hvordan programmet er opbygget.

I en SSD går tiden nedad, og SSD'ens events skal følge den samme opstilling som den use-case man skriver sin SSD ud fra.

Når man vil lave sin SSD, starter man med at putte en 'Bruger' ind, og derefter laver en lifeline der hænger på brugeren. En lifeline repræsenterer den tid brugeren bruger programmet i.

I vores gruppess SSD (som set nedenunder) starter brugeren med at trykke på en knap, for at oprette et nyt tilbud, dette sender en besked til applikationen om at den skal køre metoden opretNytTilbud(). Dette illustreres ved en streg med en pil fra brugerklassen lifeline, pegende hen imod Newyorker applikationens lifeline. Newyorker applikationen vil så besvare brugerklassen request, ved at vise de forskellige muligheder for indtastning der findes. Dette svar fra applikationen vises ved, at der fra Newyorker applikationens lifeline, går en stiplet streg med en pil hen imod brugerklassen lifeline.



¹² [Bilag 12](#): System sekvens diagram_iteration1 og iteration2

¹³ ScreenShot: Udsnit af SSD_LavEtNytTilbud_Iteration2 - Øverste pil sender systemet et request, nederste stippled pil er et svar til brugeren fra systemet.

Længere nede af lifelinen er der en boks, hvor i øverst til venstre står der 'alt', dette 'alt' står for 'alternative' og det er til for at vise at alt inde i alternative boksen, er en alternativ mulighed til at oprette et tilbud.

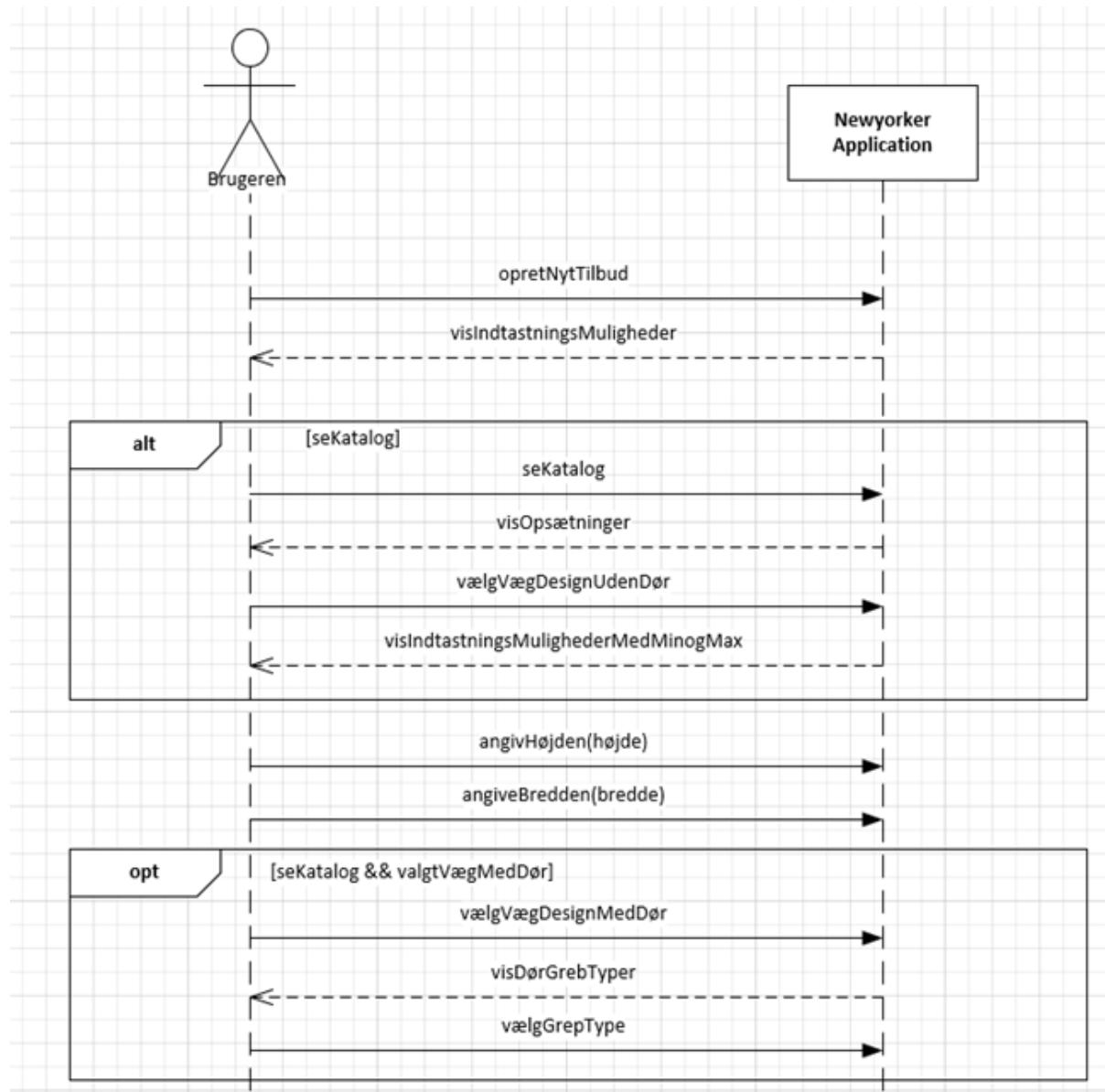
Efter man starter programmet, har man den mulighed at trykke på en knap for at se på kataloget. Hvis brugerne vælger at trykke på knappen for kataloget, vil der sendes en besked til applikationen, som vises med stregen med pilen der peger mod applikationens lifeline, om at kataloget skal frem på skærmen. Applikationen vil da svare brugerne ved at vise de mulige opsætninger af væg, som vises med den stiplede streg hvor pilen peger mod brugerens lifeline.

Brugerne kan så vælge det vægdesign som de ønsker, og efter brugerne har valgt et design, vil systemet vise brugerne hvilken højde og bredde der er muligt med det valgte design.

Ned ad lifelinen igen, bliver brugerne bedt om at indtaste hvilke højde og bredde mål de skal bruge på deres væg, og dette bliver sendt til systemet.

Der er endnu boks nu, hvor der øverst til højre står 'opt', hvilket står for 'optional'. Dvs at alt inde i opt boksen er valgfrit, og ikke noget man behøver at skulle gøre som bruger, for at komme videre i programmet.

'Opt' boksen beskriver et scenarie, hvor at brugerne vil trykke på knappen for at se kataloget, og derefter vælge et vægdesign med en dør. Det starter med at brugerne vælger at se kataloget lige som i 'alt' boksen nedenunder, dog i dette scenarie vælger brugerne et vægdesign med dør. Systemet svarer brugerne ved at vise en liste og mulige dørgrebstyper, hvor efter at brugerne skal vælge en af de viste dørgrebstyper.



Udsnit af SSD_LavEtNytTilbud_Iteration2

Operationskontrakter (OC) (Simon)

En operationskontrakt¹⁴, også forkortet til OC, er en måde at hjælpe med at definere system opførsel. Operationskontrakter bruges til at bygge videre på use-casen, ved at beskrive kompleks system opførsel.

En operationskontrakt består af fire sektioner:

Operation: Navn af operationen, og forskellige parametre.

¹⁴ [Bilag 13](#): Operationskontrakt_iteration1 og iteration2

(Cross-)References: Navnet på den use-case (eller flere) operationen foregår indeni.

Preconditions: Vigtige antagelser omkring systemets tilstand, eller objekter i domænemodellen, før man eksekverer operationen. Disse antagelser antages for at være sande.

Postconditions: Beskriver systemets tilstand efter operationen er færdiggjort.

Til vores projekt har vi lavet flere operationskontrakter, for eksempel OC13_seKatalog.

OC13_seKatalog

Operation:

Brugeren vælger at kigge i kataloget og får vist et tilbud ud fra det valgte.

Cross-reference:

UC_LavNytTilbud_Iteration 2

Pre-conditions:

- Android telefonen er tændt.
- Applikationen er installeret, og startet.
- Brugeren er klar til at vælge designe.

Post-conditions:

- Systemet er klar til at vise de forskellige muligheder i kataloget.

Operationkontrakter_seKatalog

I operationskontrakt 13 vises der en operation, hvor brugeren af systemet vælger at trykke på knappen for at vise kataloget, og får vist de forskellige valgmuligheder af vægdesigns.

Under cross-reference, refererer vi til iteration 2 af vores use-case, fordi det først var i iteration 2 vi tilføjede et katalog.

Under pre-conditions, har vi listet de forskellige ting der skal være i orden før operationen kan udføres. Android telefonen skal være tændt, appen skal være korrekt installeret og åben på telefonen. Brugeren skal være inde på kataloget.

I post-conditions, skal systemet være klar til at vise brugeren de forskellige muligheder i kataloget.

I endnu et eksempel, har vi også lavet operationskontrakten OC01_OpretNytTilbud. Der vises der en operation hvor brugeren vælger at åbne programmets interface, for at trykke på lav et tilbud knappen.

I cross-reference, refererer vi til UC01, da denne operationskontrakt blev lavet før UC02.

I pre-conditions, skal Android telefonen være tændt for at kunne åbne applikationen og applikationen skal være korrekt installeret. Der står også at ”Brugeren er klar, til at designe”, men efterfølgende, virker det unødvendigt at skrive på pre-condition, da brugeren nok er klar til at designe en væg hvis de ønsker at åbne applikationen.

I post-conditions, skal systemet virke korrekt, og være klar til at vise en skitse.

OC01_opretNytTilbud :

Operation:

Åbner interface, så et tilbud kan registreres.

Cross-reference:

UC01 Lav et nyt tilbud.

Pre-conditions:

- Android telefonen er tændt.
- Applikationen er installeret, og startet.
- Brugeren er klar, til at designe.

Post-conditions:

- Systemet er klar, til at vise skitse.

Operationskontrakter – opret Nyt Tilbud

Hvorfor lave operations kontrakter?: Nogle gange giver use-casen dig alt den information, du har brug for til at lave dit design og program, i hvilket tilfælde operationskontrakter ikke vil hjælpe, og er derfor ligegyldige for at bygge din applikation.

Det der gør operationskontrakter nyttige, er når situationer opstår i dit program, som er for komplekse og detaljerede til at en use-case kan fange alt hvad der skal ske. Det er i sådanne situationer, at en operationskontrakt kan være et godt artefakt at bruge tid på at lave.

Hvis man laver en operationskontrakt for hvert enkelt system operation, så er det måske tilfældet at ens use-case ikke er lavet ordentligt, og hvis det er tilfældet, tvinger man kun ekstra arbejde på sig

Object-Oriented Design (OOD)(Mikkel)

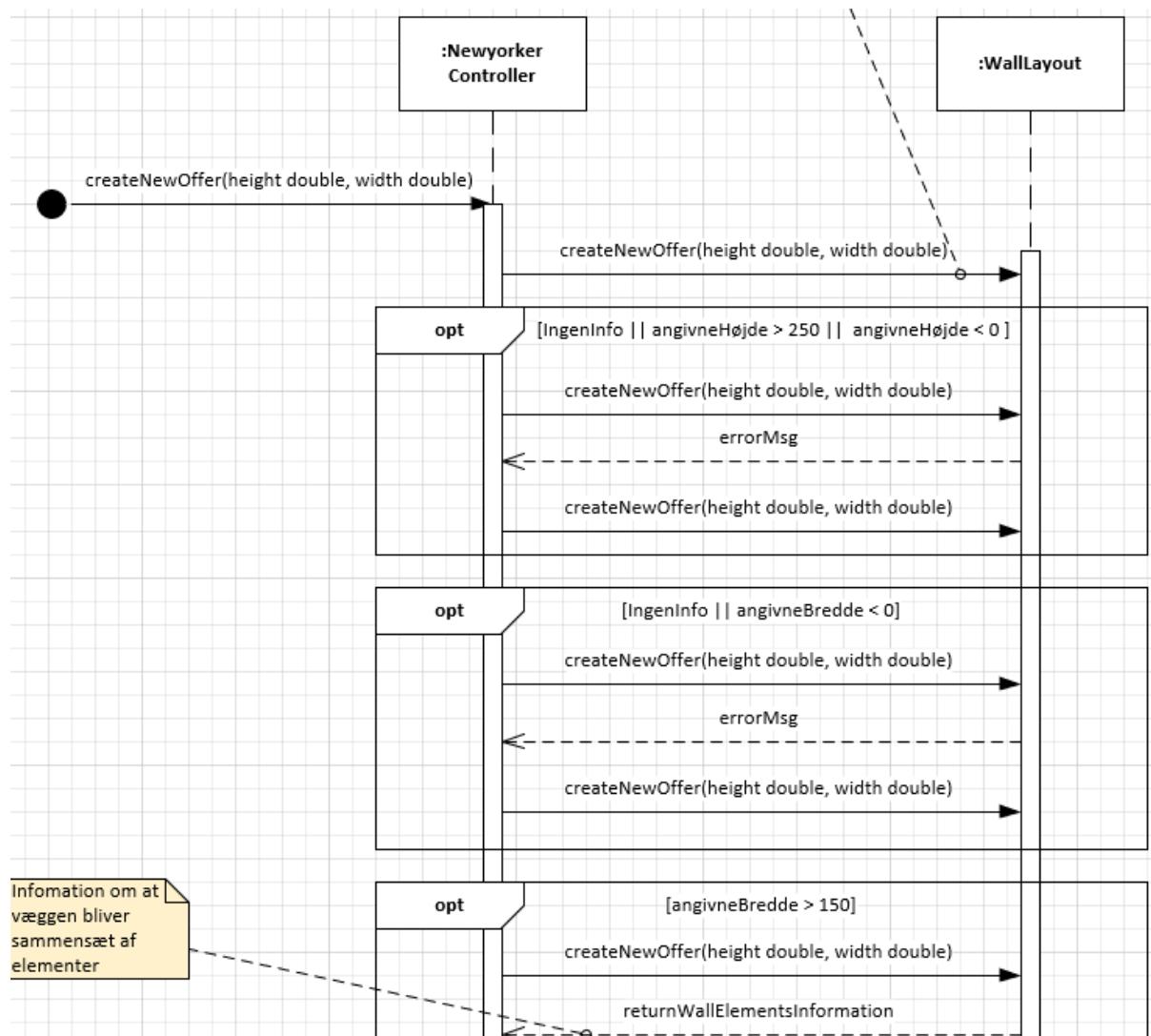
Sekvensdiagrammer (SD)

Sekvensdiagrammet¹⁵ viser i hvilken rækkefølge funktioner, og metoder, bliver eksekveret i programmet, og hvilke variabler der skal bruges, for at kunne fuldføre denne handling. Disse nødvendigheder kaldes også “Post-conditions”.

Selv navnet “sekvensdiagram” er med til at give en forståelse for hvad det indebærer, da ordet “sekvens” beskriver en bestemt ting der sker på et bestemt tidspunkt. Et eksempel på dette, kan ses på figuren nedenunder.

Her kan man se, at klassen “createNewOffer” er den første klasse, der sender data fra “NewyorkerController” til “WallLayout”. “height double”, og “width double” er så de variabler den sender med sig, hvor “double” beskriver at det er en datatype, der kan indeholde et kommatal, på 64 bit.

¹⁵ [Bilag 14](#): Sekvens diagram_iteration1 og iteration2



Dette er et eksempel på et Sekvensdiagram (SD)

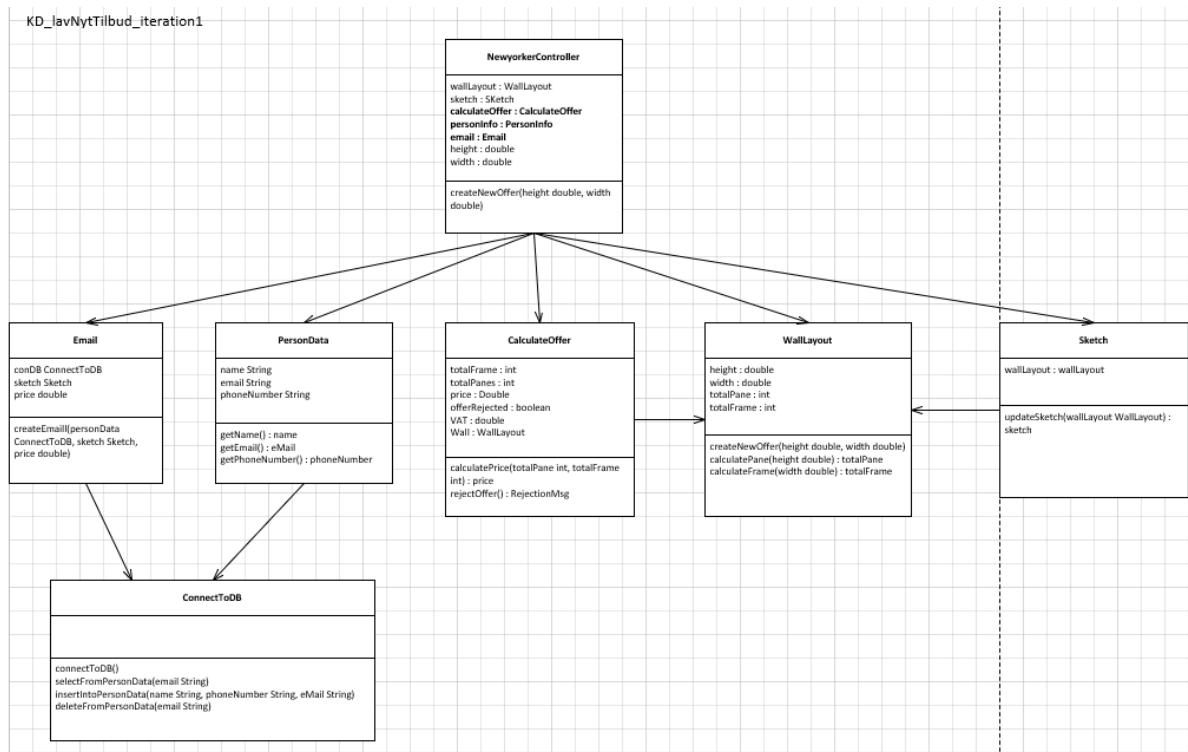
Klassediagrammer (KD)

Klassediagrammet¹⁶ viser, hvilket klasser der kommunikerer, men er også med til at give et overblik over hvilke funktioner, den specifikke klasse indeholder. Klassediagrammet er særligt vigtigt, da det også kan vise programmets kode, på mere brugervenligt niveau, selvom man ikke har været med, til kodningen af programmet. Dette gør det nemmere for folk der indirekte arbejder på programmet at danne sig et overblik over koden, så de bedre kan dokumentere koden.

Et klassediagram er sat op, så alle klasser er repræsenteret i en kasse med udadgående linjer, der viser klassens relation til andre klasser, samt hvor mange inputs den specifikke klasse, kan tage. Der kan

¹⁶ [Bilag 15](#): KlassDiagram_iteration1 og iteration2

også stå yderligere informationer, under klassens navn. Dette er funktioner, og variabler, som klassen kan indeholde.



Dette er et eksempel på, hvordan et klassediagram kan se ud.

Klasser (Mikkel)

Newyorker Controller

klassen Newyorker Controller, står for håndteringen af brugerens input som "height", og "width". Disse variable tager funktionen createNewOffer, og regner på. Resultat bliver derefter returneret, så andre klasser kan bruge udregningerne.

Email

Klassen Email, står for videregivningen af persondata, samt data. Den bruger så funktionen createEmail, der tager variablerne personData, connectToDB, sketch, og price, som den så bruger til generering af mål, til New-Yorker. Funktionen createEmail tager også connectToDB med som input, der den skal kunne opbevare oplysningerne på den database, der så kan tilgås af NewYorker.

Mailen der bliver sendt til New-yorker, indeholder så alle variablerne, der er sat op som en arbejdstegning, så New-yorker kan få lavet væggen, uden at skulle besvær på aflæsningen.

Vi valgte at lave en klasse, da vi gerne vil have en separat klasse, som kunne håndtere alt eksportering af data. Dette var hovedsageligt, for at have en ren kode overalt.

PersonData

Klassen PersonData, står for håndtering af kundens indtastede oplysninger om sig selv, som den så klargør til brug i andre klasser, ved at returnere dens udregninger.

PersonData er en ret interessant klasse, som vi valgte at tilføje. Dette var for at kunne håndtere persondata, og indsamle den i denne klasse. Men også for at nemt kunne ændre i alt gennemgående data om kunden, hvis kunden ønsker at fjerne sin persondata. Dette er også for at overholde GDPR reglerne

Vi bruger funktionerne getName, getEmail, og getPhoneNumber, til at få alt data fra brugerens input. Det fastgør vi så med sættere, i form af setName, setEmail, og setPhoneNumber. På denne måde kan vi gemme og håndtere, brugerens informationer.

```
1 package NewYorkerApp;
2
3 public class PersonData {
4
5     private String name;
6     private String email;
7     private String phoneNumber;
8
9
10    public String getName() {
11        //hente fra bruger input
12        return name;
13    }
14
15    public String getEmail() {
16        //hente fra bruger input
17        return email;
18    }
19
20    public String getPhoneNumber() {
21        //hente fra bruger input
22        return phoneNumber;
23    }
24
25    public void setName(String name) { this.name = name; }
26
27    public void setEmail(String email) { this.email = email; }
28
29    public void setPhoneNumber(String phoneNumber) { this.phoneNumber = phoneNumber; }
30
31
32
33
34
35
36
37
38 }
```

CalculateOffer

CalculateOffer er klassen, som står for beregningen af prisen på den væg, som kunden har designet.

Her er det calculatePrice funktionen, der tager totalPane, og totalFrame, og regner med det. Dette bliver så returneret, så andre klasser kan bruge dataen. Hvis funktionen rejectOffer bliver kaldt, bliver den udregning ikke brugt, og kan overskrives af den næste beregning. Denne klasse er en af de mest vigtige klasser, i vores program. CalculateOffer står for alt udregning, og er derfor meget vigtig at have i orden. Ellers vil resten af programmet, ikke få det korrekte information. Vi valgte at lave denne klasse, for at få alle udregninger samlet, og gjort klar, til næste klasse.

På linje 24: Funktionen calculatePrice, bruger vi til udregningen af totalPrice. Dette gør vi ved at tage tg, glassPanePrice, og VAT, som vi ganger med hinanden. Denne værdi returnerer vi så.

Vi har valgt at, have forskellige funktioner for udregning af glastyper, selvom de har samme måde at blive udregnet på. Et eksempel på dette er calculatesatinGlass, på linje 37, som ganger totalGlass, satinGlass, og VAT, med hinanden. Denne værdi bliver så returneret. Alle andre udregninger af glastyper, fungere på sammen måde, bare hvor satinGlass variablen, er en anden.

På linje 47: calculateDoor, er en funktion der udregner prisen på dørene. Her lytter den til hvad brugeren vælger af dørtyper, og vælger et regnestykke ud fra brugerens valg, hvor udregningen/prisen så bliver returneret. Hvis der ikke er noget valg, returnere den null. Det er særlig vigtigt at, alle funktioner returnere noget, ellers vil programmet crashe.

```
1 package NewYorkerApp;  
2  
3 public class CalculateOffer {  
4  
5     int totalFrame;  
6     int totalPane;  
7     double totalPrice;  
8     boolean offerRejected;  
9     double glassPanelPrice = 985;  
10    double satinGlass = 70;  
11    double lydGlass = 95;  
12    double lockCase = 500;  
13    double singleDoor = 2000;  
14    double doubleDoor = 4000;  
15    double singleSlideDoor = 2480;  
16    double doubleSlideDoor = 4960;  
17    double messing = 500;  
18    double blackHandle = 250;  
19  
20    final double VAT = 1.25;  
21    WallLayout wall = new WallLayout();  
22  
23  
24    public double calculatePrice(int tg) {  
25  
26        totalPrice = tg * glassPanelPrice * VAT;  
27        System.out.println("pris : " + totalPrice + "DKK.");  
28  
29        return totalPrice;  
30    }  
31  
32    public boolean rejectOffer() {  
33  
34        return offerRejected;  
35    }
```

```

35     }
36
37     public double calculateSatinGlass(int totalGlass) {
38
39         return totalGlass * satinGlass * VAT;
40     }
41
42     public double calculateLydGlass(int totalGlass) { return totalGlass * lydGlass * VAT; }
43
44     @
45     public double calculateDoor(int totalGlass, String doorType) {
46
47         if (doorType.equals("Enkeltdør uden låsekasse")) {
48             return singleDoor * VAT;
49
50         } else if (doorType.equals("Enkeltdør med låsekasse")) {
51             return (singleDoor + lockCase) * VAT;
52
53         } else if (doorType.equals("Dobbeltdør uden låsekasse")) {
54             return doubleDoor * VAT;
55
56         } else if (doorType.equals("Dobbeltdør med låsekasse")) {
57             return (doubleDoor + 2*lockCase) * VAT;
58
59         }
60
61         return 0;
62     }
63
64 }
65

```

WallLayout

WallLayout, er en klasse der står for fremvisningen af væggen, i form af et billede.

Den tager variablerne totalPane, og totalFrame, som den bruger til at finde et korrekt billede af kundens væg. Vi valgte at lave denne klasse, da New-Yorker ønskede at kunden skulle kunne se deres væg design. Det var tænkt, at væggen skulle genereres af programmet, men det var for tidskrævende, så vi valgte at tilføje billeder, som appen kunne håndplukke.

På linje 13: Funktionen createNewOffer, tager inputtet height, og width, som den så tjekker om de er større, eller mindre, end maks værdierne.

På linje 25: calculatePane, er en funktion der står for udregningen af antal glasruder, som skal bruges. Her bliver den ved med at sætte ruder ind i væggen, indtil regnestykket på linje 28, giver 60. Derefter stopper den looped.

På linje 32: calculateFrame, fungere på sammen måde som calculatePane, men i stedet regner på hvor mange rammer, der skal være.

```

7      public class WallLayout {
8          public int totalPane = 0;
9          public int totalFrame = 0;
10         public double height;
11         public double width;
12
13         public void createNewOffer(int height, int width) {
14             //TODO vildarer brugerinput
15             if (height > 250) {
16                 System.out.println("Max højden er 250cm, indtast venligst højden igen");
17             } else if (height < 0) {
18                 System.out.println("Højden er mindre end 0");
19             } else if (width > 150) {
20                 System.out.println("Bredden er mere end 150cm, væggen opbygges af flere elementer");
21             } else if(width < 0) {
22                 System.out.println("Bredden er mindre end 0");
23             }
24         }
25         public int calculatePane(int h) {
26             this.height = h;
27             for (int i = 1; i <= 100; i++) {
28                 if (h / i <= 60) { totalPane = i; break; }
29             }
30             return totalPane;
31         }
32         public int calculateFrame(int width) {
33             for (int i = 1; i < 100; i++) {
34                 if (width / i <= 45) { totalFrame = i; break; }
35             }
36         }
37     }
38 }
```

Sketch

updateSketch funktionens opgave, er at generere et billede af væggen, samt en udgave af dette billede, med mål, og materialer taget i brug. Dette bliver så returneret, så andre klasser kan bruge tegningerne.

Denne klasse var særlig vigtig at have, da New-Yorker ønskede at kunne få målene på de vægge, som kunderne bestilte. Så ligesom klassen WallLayout, tager denne klasse nogle billeder fra dens hukommelse, og putter mål på dem.

ConnectToDB

Klassen ConnectToDB, står for at lave en forbindelse til den valgte database, hvor klassen PersonData så kan kommunikere med databasen. Klassen indeholder de forskellige metoder, som er nødvendige for at kunne hente, indsætte eller slette information i databasen. Klassen kan løbende udvides som behov opstår for at gemme yderligere typer informationer i databasen. Denne klasse står for alt forberedelse af Databaser. Dette er så i form af, den laver en forbindelse, som Email klassen kan sende data på.

På linje 11: Funktionen connectToDB, står for oprettelsen af nødvendige variabler, som der skal bruges til oprettelse af en synkron forbindelse, med den valgte database.

På linje 27: selectFromPersonData, tager brugerens indtastede email adresse, og søger efter brugbar information, der er tilknyttet til denne specifikke email adresse. Hvis der ikke er noget information, lukker den forbindelserne, så der ikke bruges unødvendig regnekraft, på det. Dette kunne resultere i at Appen kører langsomt, eller brugerens telefon cracher.

På linje 62: Funktionen insertIntoPersonData, gemmer brugerens indtastede oplysninger, i databasen. Dette gør det muligt for programmet at opbevare data som brugeren, som brugeren tidligere har indtastet. Dette er særligt vigtigt, hvis man skal have en mere flydende brugeroplevelse, men gør det også nemmere for New-Yorker, at se hvem deres kunde er.

På linje 93: deleteFromPersonData, er en yderst vigtig funktion, når man bor i EU. Denne funktion gør det muligt for systemet, at slette brugere, og deres oplysninger. Dette frigør plads på databasen, men kan også redde New-Yorker, for en bøde for at bryde GDPR reglerne.

```
1 package NewYorkerApp;
2
3 import ...
4
5
6 public class ConnectToDB {
7
8     public static Connection connectToDB() {
9         Connection conn = null;
10        try {
11            // db parameters
12            String url = "jdbc:sqlite:Users/jennapetersen/Desktop/NewYorker.db";
13            // create a connection to the database
14            Class.forName("org.sqlite.JDBC");
15
16            conn = DriverManager.getConnection(url);
17
18        } catch (SQLException | ClassNotFoundException e) {
19            System.out.println(e.getMessage());
20        }
21    }
22    return conn;
23 }
24
25 }
```

```

27     public static void selectFromPersonData(String email) {
28         Connection conn = null;
29         try {
30             // create a connection to the database
31             conn = connectToDB();
32
33             String sqlPersonInfo = "SELECT * FROM PersonData WHERE email = '" + email + "'";
34
35
36             Statement stmt = conn.createStatement();
37             ResultSet rs = stmt.executeQuery(sqlPersonInfo);
38
39             System.out.println(sqlPersonInfo);
40             while (rs.next()) {
41                 System.out.println(rs.getInt( columnIndex: 1 ) + "\n" +
42                     rs.getString( columnIndex: 2 ) + "\n" +
43                     rs.getString( columnIndex: 3 ) + "\n" +
44                     rs.getString( columnIndex: 4 ) + "\n"
45             );
46         }
47
48     } catch (SQLException e) {
49         System.out.println(e.getMessage());
50     } finally {
51         try {
52             if (conn != null) {
53
54                 conn.close();
55             }
56         } catch (SQLException ex) {
57             System.out.println(ex.getMessage());
58         }
59     }
60 }

```

```

62     public static void insertIntoPersonData(String name, String phonenumber, String email) {
63         Connection conn = null;
64         try {
65             // db parameters
66             //String url = "/Users/jennapetersen/AndroidStudioProjects/NewYorker-UI_Iteration2_1/05 Implementation/New-Yorker designer app/app/src/main/java";
67             // create a connection to the database
68             conn = connectToDB();
69
70             String sqlPersonInfo = "Insert into PersonData (name, phonenumber, email) values ('" + name + "', '" + phonenumber + "', '" + email + "') ";
71
72
73             Statement stmt = conn.createStatement();
74             stmt.executeQuery(sqlPersonInfo);
75
76
77         } catch (SQLException e) {
78             System.out.println(e.getMessage());
79         } finally {
80             try {
81                 if (conn != null) {
82
83                     conn.close();
84                 }
85             } catch (SQLException ex) {
86                 System.out.println(ex.getMessage());
87             }
88         }
89
90
91     }
92

```

```

93     public static void deleteFromPersonData(String email) {
94         Connection conn = null;
95         try {
96             conn = connectToDB();
97
98             String sqlPersonInfo = "Delete From PersonData Where email = '" + email + "'";
99
100
101            Statement stmt = conn.createStatement();
102            stmt.executeQuery(sqlPersonInfo);
103
104
105        } catch (SQLException e) {
106            System.out.println(e.getMessage());
107        } finally {
108            try {
109                if (conn != null) {
110
111                    conn.close();
112                }
113            } catch (SQLException ex) {
114                System.out.println(ex.getMessage());
115            }
116        }
117    }

```

Kvalitetssikring (Mikkel)

- **Test af produkt**

Når vores kode skulle testes, startede vi med at køre koden, og lede efter bugs. Dette blev gjort i form af, at trykke på alt, og skrive forkerte værdier ind i alle felter. På denne måde, kunne vi finde en masse fejl, der fik programmet til at crashet, og vi kunne rette op på de fejl.

Når UI skulle testes, fik vi kunden til at teste programmet, i hvert kundemøde. Dette gjorde at kunden kunne se vores fremgang, samt vi kunne få en konstant feedback, på fejl, og mangler.

- **Review**

Vi valgte at alle skulle være med i reviews af kode, og designforslag. Dette betyder at vi har haft en ret dynamisk arbejdsproces, hvor alle har haft en finger med i alt, ligemeget hvilke emner der snakkes om.
Dette gør at alt vores erfaring, bliver samlet, og vi kan støtte hinanden i arbejdet. Så alle har hinandens ryg dækket.

- **Procesforbedring**

Da vi skulle designe programmet, og sætte det hele op, valgte vi at mødes kl 9:30 hver morgen, og fik snakket alt igennem. Dette var så igennem iteration 1, hvilke betyder at det meste af iteration 1, bestod af møder, samt mockups.

Det gode ved disse møder, var at alle kunne komme med deres viden, fra tidligere projekter, der hjalp med at få en bedre struktur, og orden.

- **Metrikker**

I gruppen kører vi et system, hvor vi arbejder i iterationer. For hver iteration, har vi lavet et møde med kunden, hvor vi har fået rettet fejl ud, og kommet med bedre løsninger.

Vi valgte at involvere kunden i denne process, der det er kunden der har det sidste ord, når det gælder design af UI, og funktioner.

I selve arbejdsprocessen, har vi valgt at godkende hinandens arbejde via siden Github. Så en har lavet noget til programmet, og de andre ser det igennem og accepterer de ændringer, den person har lavet.

Dette resulterer i et program, som vi alle har en forståelse for, hvordan fungere.

Udgivelse (Mikkel)

Programmet New-Yorker væg designer, bliver udgivet på programmet “Play store”.

Play store, er et program som Android anvender, til hentning af nye programmer til telefonen, samt opdateringer af firmware, og software.

Programmet vil blive lagt på op gratis, så alle kunder nemt og hurtigt, kan få designet, samt bestilt en væg.

Dog er der nogle ting, vi skal være opmærksomme på, når vi uploader.

App'en vil ikke komme direkte på Googles play store, der Google har en kvalitetskontrol af de apps, som folk ligger på. Dette betyder at, der kan gå et par dage, inden vores app bliver synlig.

Men det betyder også, at appen kan blive taget af Playstore, inden nogen når at downloade den.

Vi skal også have en “**Google Play Developer Account**”¹⁷ før vi kan uploadere nogle former for programmer. Dog tager dette ikke lang tid.

¹⁷ Guide til Playstore <https://bolter.com.au/launch-how-to-release-your-app-on-the-google-play-store/>

Konklusion

Vi startede opgaven med at opstille følgende problem i vores problemformulering:

New-Yorker.dk oplever stor efterspørgsel, på deres vægge. Dette presser dem, da de både skal tage imod kundeforespørgsler, udarbejde tilbud og designe hver enkelt skillevæg.

Vi vil derfor undersøge hvordan vi kan udvikle en app som kan tage imod målene på en væg og derudfra udarbejde et tilbud og en tegning, som kan bruges til den videre produktion af væggen.

Det første vi gik i gang med var selve analysen af problemet. Vi fokuserede først på, at få afklaret de krav som kunden havde til applikationen og at få lavet en mock-up, så vi kunne vise vores tanker til kunden og fra starten være sikre på at vi forstod hinanden. Det har været en god proces hvor vi løbende har kunnet få feedback på det udviklede og sørge for at holde os på rette spor.

Dernæst kiggede vi på det forretningsmæssige aspekt, hvor vi undersøgte hvordan andre lignende virksomheder lavede tilbud. Her kunne vi konkludere at mange stod i samme situation som New-Yorker.dk og at der derfor tilsyneladende er et marked for lignende applikationer.

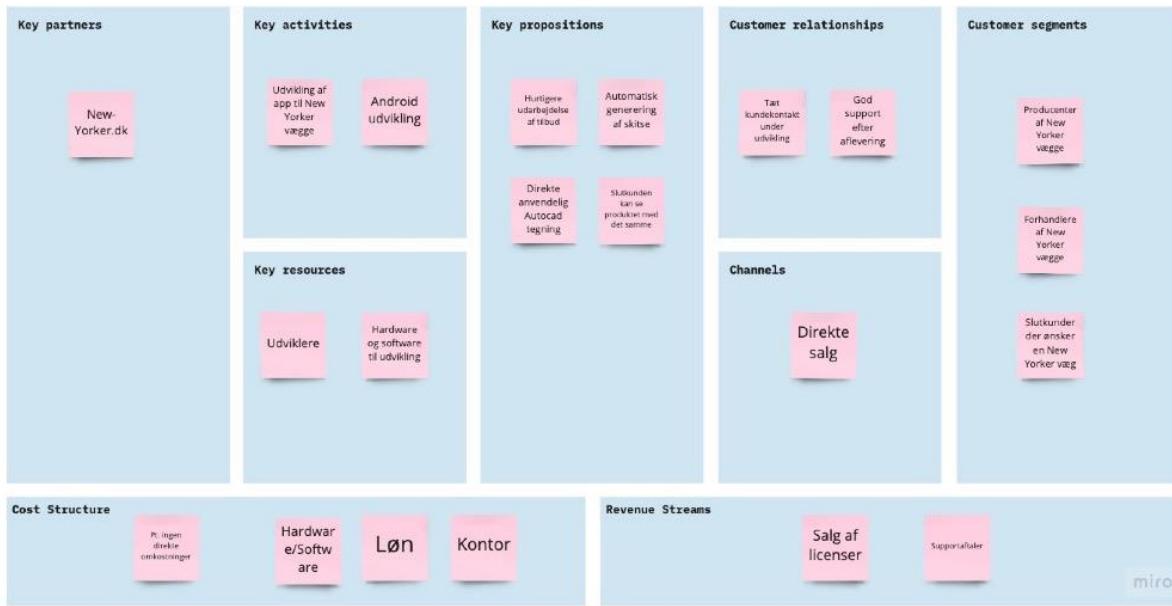
Så gik vi gang med selve design- og udviklingsfasen. Her har vores hovedfokus været på, at levere kerne funktionaliteten i app'en, nemlig at kunne levere et korrekt udregnet tilbud og præsentere det på en god og brugervenlig måde. Vi ville gerne have at brugeren fik en god oplevelse samtidig med, at vi fik implementeret så mange som muligt af kundens ønsker og vi føler vi er kommet godt i mål med det meste.

Der har også været udfordringer og vi har ikke nået alt det vi håbede vi kunne, men vi har lært en masse som vi kan tage med videre i det næste projekt. Vi føler at vi er vokset med opgaven og at vi står bedre rustet nu end vi gjorde for en måned siden og det mener vi er en god måde at afslutte et projekt på.

Bilag

Bilag 1: business Model canvas

The Business Model Canvas



miro

Bilag 2: Virksomhed finansiering excel ark

New Yorker.dk		Investeringsbudget									
Post	Enhed	2021		2022		2023		2024		2. halvår	
		1. halvår	2. halvår	1. halvår	2. halvår	1. halvår	2. halvår	1. halvår	2. halvår		
Softwarelicens Vedligeholdelsesaftale		0	50000	100000	150000	150000	200000	200000	250000	250000	
Brutto omsætning, p1	DKK	0	59000	118000	177000	186000	245000	254000	313000		
Samlet omsætning	DKK	0	59000	118000	177000	186000	245000	254000	313000		
Cost of goods	DKK	0	0	0	0	0	0	0	0	0	
Faste omkostninger											
Husleje (Kontorhotel plads)	DKK (1000)	0	0	0	0	10	10	10	10	10	
Bogholderi	DKK (1000)	2	2	2	2	5	5	5	5	5	
Rejseudgifter	DKK (1000)	0	6	6	10	10	15	15	15	6	
Diverse	DKK (1000)	2	5	5	10	10	10	10	10	6	
Kapacitetsomkostninger											
Adm ex i øn	DKK (1000)	0	3	3	6	6	6	6	6	6	
Adm i øn	DKK (1000)	0	0	0	0	90	90	90	90	50	
Indtægter, i alt	DKK (1000)	0	59	118	177	186	245	254	313		
Udgifter i alt	DKK (1000)	4	16	16	28	131	136	136	136	83	
P&L	DKK (1000)	-4	43	102	149	55	109	118	118	230	
Profit	DKK (1000)	-4	43	102	149	55	109	118	118	230	
Investeringer Cash on the end of the Period	DKK (1000)	-4	15	15	30	30	30	30	30	30	
	DKK (1000)		28	87	119	25	79	88	88	200	

Bilag 3: Projektplan excel ark

Tidsregistrering_gruppe5_Simon, Mikkel og Jenna												
TODO	Memo	Opfølgning	Dato	Simon	Mikkel	Jenna	Estimeret antal timer	Estimeret startdato	Faktisk sluttid	Faktisk antal timer	Faktisk startdato	Faktisk sluttid
Iteration 1 17/05-21/05												
Risikobørrelister							17/05	17/05				
Visiondokumentet							17/05	17/05				
Use case-diagrammer							17/05	17/05				
Causal/Usecase							17/05	17/05				
Formelle use cases 1, 2							17/05	17/05				
Aktivitetsdiagrammer 1,2							18/05	18/05				
FURPS							18/05	18/05				
Diagramme							18/05	18/05				
Domanemodel 1,2							19/05	19/05				
Systemsekvensdiagrammer 1,2							19/05	19/05				
Operationskontrakter 1,2							19/05	19/05				
Sekvensdiagrammer 1, 2							20/05	20/05				
Klassediagrammer 1, 2							20/05	20/05				
kode 1, 2							21/05	21/05				
Test 1,2							21/05	21/05				
Rapport												
New iteration 1 20/05-28/05												
Causal/Usecase				Mikkel	Jenna	1	20/05	20/05	1	20/05	20/05	
Formelle use cases				Mikkel	Jenna	1	20/05	20/05	1	20/05	20/05	
Aktivitetsdiagrammer					Jenna	1	20/05	20/05	1	20/05	20/05	
Domanemodel					Jenna	1	20/05	20/05	1	20/05	20/05	
Systemsekvensdiagrammer					Jenna	1	20/05	20/05	1	20/05	20/05	
Operationskontrakter					Jenna	3	25/05	25/05	3	25/05	25/05	
Sekvensdiagrammer					Jenna	1	20/05	20/05	2	20/05	21/05	
Klassediagrammer					Jenna	1	24/05	25/05	1	24/05	25/05	
Dataordbog							25/05					
kode ud fra KO				Simon	Jenna	16	25/05	26/05	16	25/05	28/05	
Test				Simon	Mikkel	4	28/05	28/05				
UI Kode					Mikkel	30	25/05	28/05	30	25/05	28/05	
Rapport							28/05					
Iteration 2 31/05-11/06												
rette Mockup					Jenna	1	27/05	27/05				
Dataordbog					Mikkel	1	01/06	01/06	2	01/06	01/06	
Formelle use cases				Simon	Mikkel	Jenna	1	31/05	31/05	3	01/06	01/06
Aktivitetsdiagrammer					Jenna	1	31/05	31/05	1	01/06	01/06	
Domanemodel				Simon	Mikkel	Jenna	1	01/06	01/06	1	01/06	01/06
Systemsekvensdiagrammer				Simon	Mikkel	Jenna	1	01/06	01/06	1	01/06	01/06
Operationskontrakter							3	01/06	01/06	3	01/06	01/06
Sekvensdiagrammer					Jenna	2	02/06	02/06	3	02/06	02/06	
Klassediagrammer					Jenna	2	02/06	02/06				
kode ud fra KO					Mikkel	Jenna	4	02/06	06/06			
Test						Jenna	2	07/06	07/06	07/06	07/06	
UI Kode						Mikkel	Jenna	30	02/06	06/06	02/06	
Rapport							04/06					
Iteration 3												
Formelle use cases												
Aktivitetsdiagrammer												
Domanemodel												
Systemsekvensdiagrammer												
Operationskontrakter												
Sekvensdiagrammer												
Klassediagrammer												
kode ud fra KO												
Test												
UI Kode												
Samlet rapport												

Bilag 4: Risikoanalyse**Risiko analyse**

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Sygdom blandt udviklerne	4/10	Tidstab til review af input(4/10)	lav	1/10	Stadig tidstab, men bedre oversigt (3/10)

Imødekommes strategi:

Der skal tages højde for et vist antal sygedage i tidsplanen, et normalt estimat er 5% fravær. Planen er lavet så der er taget højde for sygdom og vil derfor kunne gennemføres hurtigere uden sygdom.

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Fejlestimer	6/10	Vi når ikke i mål med projektet. (4/10)	Høj	2/10	Tidsplanen justeres løbende og giver bedre chance for at vi kan levere (1/10)

Imødekommes strategi:

Der skal laves løbende opfølgning på tidsplanen og estimererne samt planen skal tilrettes når der er afvigelser.

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Der opsamles persondata	10/10	Persondata ikke opbevaret korrekt(10/10)	Høj	3/10	Kryptering samt oversigt over gemte data sikrer at GDPR er overholdt(1/10)

Imødekommes strategi:

Ved opsamling af personlige data skal det sikres at GDPR overholdes. Persondata skal krypteres når det ligger gemt/samt når det flyttes. Derudover bør virksomheden etablere et datalogatalog hvor det registreres at der findes persondata i denne applikation

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Fejl ved gemning af tilbudsfil	2/10	Filen kan ikke åbnes.(10/10)	Lav	1/10	Filformatet kontrolleres altid og dermed sikres filens integritet.(1/10)

Imødekommes strategi:

Filformatet bliver valideret af systemet efter det er gemt.

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Filen kan ikke åbnes i Autocad.	5/10	Vi kan ikke imødekomme kundens ønske(8/10)	Høj	3/10	Selv med grundig test og undersøgelse er det stadig usikkert om filen kan gemmes i DWG format.(3/10)

Imødekommes strategi:

Det er usikkert hvordan tegningen skal genereres i korrekt format, det skal derfor undersøges hvordan vi gør det rent teknisk og det skal testes at det er muligt for os at gemme i det format.

Risiko	Sandsynlighed	Konsekvens	Prioritet	Revideret sandsynlighed	Revideret konsekvens
Udsendelse af email til forkert adresse	5/10	Der er en risiko for, at vi får sendt persondata til en forkert email adresse.(5/10)	Mellem	1/10	Der er stadig en lav chance for at brugeren blot bekræfter adressen uden at kontrollere den, men det er mere sikkert.(1/10)

Imødekommes strategi:

Modtageremail tages fra en fast liste hvor producent/sælger/tømrer findes og kundens mailadresse skal bekræftes inden mailen sendes.

Bilag 5: Visionsdokument

Visions Dokument

Vision

Produktet henvender sig til New-Yorker.dk, som er en virksomhed der producerer New Yorker skillevægge. De ønsker sig en app som kan hjælpe dem med hurtigt og let at lave et tilbud på en skillevæg, blot ud fra indtastning af størrelse samt valg af tillæg. Formålet er, at lette processen både for forhandleren, som let og elegant kan lave et tilbud og vise kunden med det samme, samt for producenten, som vil få tilsendt et diagram over hvordan væggen ser ud dermed hurtigt kan komme i gang med produktionen.

Interessentanalyse

Brugere:

- **Producenten** ønsker en let måde selv at lave tilbud på, samt at modtage færdige tilbud fra deres forhandlere så de hurtigt kan komme i gang med produktionen.
- **Forhandleren** ønsker let og elegant at kunne vise kunden hvordan væggen ser ud samt at kunne give et færdigt tilbud.

Featureliste:

- Oprettelse af tilbud ud fra de angivne mål
- Tilpasning af væggen med forskellige materialer
- Visning af en færdig skitse af væggen ud fra mål og materialevalg
- Generering af skitse i autocad format
- Afsendelse af tilbud til producenten, forhandleren og kunden via mail

Bilag 6: Casual Usecase

Casual Usecase

UC1: Lav et tilbud (iteration 1)

- Brugeren starter oprettelse af nyt tilbud
- Systemet viser indtastnings mulighederne for en ny væg
- Brugeren indtaster væggens højde og bredde.
- Systemet beregner antallet af lodrette ruder ud fra at højde skal ligge så tæt på 60 cm som muligt
- Systemet beregner antallet af fag, ud fra at bredden på ruderne skal ligge så tæt på 45 cm som muligt
- Systemet genererer skitse med det beregnede antal ruder og fag
- Systemet viser skitse.
- Systemet opdaterer prisen ud fra den indtastede højde/bredde og de beregnede antal ruder og fag
- Systemet viser tilbudsprisen
- Brugeren bekræfter den viste skitse og tilbudsprisen
- Systemet viser kontaktformularen til indtastning af kundeinformation
- Brugeren angiver navn, email og telefonnummer
- Systemet laver e mail og sender med vedhæftede tilbud samt skitse

Bilag 7: Formelle Usecase_iteration1 og iteration2**UC_Lav nyt tilbud (Iteration 1)**

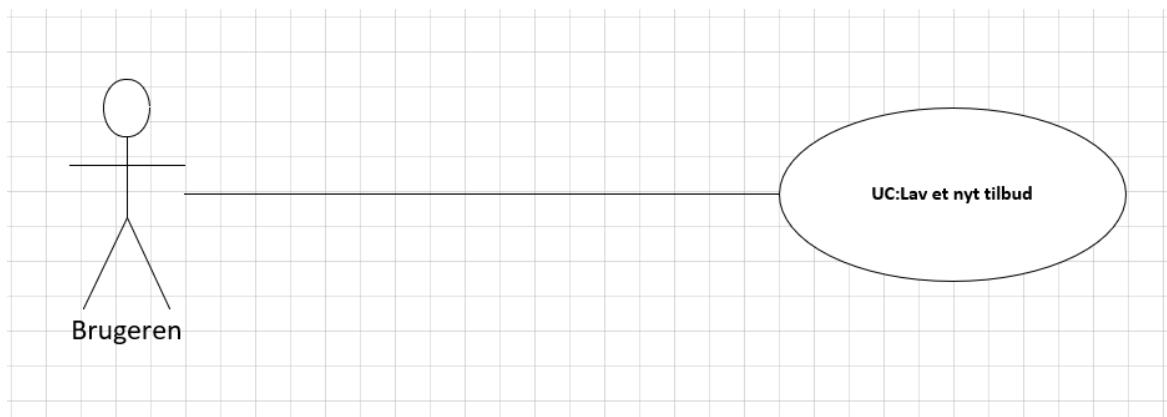
UseCase navn	UC Lav nyt tilbud
Scope	New Yorker
Primary Actor	Tømrer/Kunder
Level	User-goal
Description	Systemet skal kunne modtage angivelse af størrelse på den ønskede væg, udregne og præsentere en skitse, beregne et færdigt tilbud på væggen samt sende disse til kunden og producenten på email.
Stakeholders and Interests	Byens Vinduer/New-Yorker.dk
Preconditions	Den android enhed er tændt. Applikationen er installeret og klar til at bruge.
Succes Guarantee	Der er oprettet et gyldigt tilbud og det er sendt til kunden og producenten
Main Succes Scenario	<p>1. Brugeren starter oprettelse af nyt tilbud. 2. Systemet viser indtastnings mulighederne for en ny væg. 3. Brugeren indtaster væggens højde og bredde. 4. Systemet beregner antallet af lodrette ruder ud fra at højde skal ligge så tæt på 60 cm som muligt. 5. Systemet beregner antallet af fag, ud fra at bredden på ruderne skal ligge så tæt på 45 cm som muligt. 6. Systemet genererer skitsen med det beregnede antal ruder og fag. 7. Systemet opdaterer prisen ud fra den indtastede højde/bredde og de beregnede antal ruder og fag. 8. Systemet viser væg design og tilbudsprisen. 9. Brugeren bekræfter den viste væg design og tilbudsprisen. 10. Systemet viser kontaktformular til indtastning af kundeinformation. 11. Brugeren angiver navn, email og telefonnummer. 12. Systemet laver e mail og sender med vedhæftede tilbud samt skitse.</p>
Extensions	<p>3a. Mangler at udfylde information om højde. 1. System oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 3.</p> <p>3b. Hvis brugeren angiver negativ tal om højde. 1. System oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 3.</p> <p>3c. Hvis brugeren angiver højden mere end 250 cm. 1. Systemet oplyser at max højde er 250 cm og afventer brugerinput. Processen fortsætter ved punkt 3.</p> <p>5a. Mangler at udfylde information om bredde. 1. System oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 5.</p> <p>5b. Hvis brugeren angiver negativ tal om bredde. 1. System oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 5.</p> <p>5c. Hvis brugeren angiver bredden mere end 150 cm. 1. Systemet oplyser at væggen vil blive sammensat af flere elementer. Processen fortsætter ved punkt 5.</p> <p>9a. Hvis brugeren ikke bekræfter tilbuddet. 1. Forløbet starter forfra ved punkt 1.</p>
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	Use casen forekommer ved oprettelse af en designe væg
Miscellaneous	-

UC_Lav nyt tilbud (Iteration 2)

UseCase navn	UC Lav nyt tilbud
Scope	New Yorker
Primary Actor	Tømrer/Kunder
Level	User-goal
Description	Systemet skal kunne modtage angivelse af størrelse på den ønskede væg, udregne og præsentere en skitse, beregne et færdigt tilbud på væggen samt sende disse til kunden og producenten på email.
Stakeholders and Interests	Byens Vinduer/New-Yorker.dk
Preconditions	Den android enhed er tændt. Applikationen er installeret og klar til at bruge.
Succes Guarantee	Der er oprettet et gyldigt tilbud og det er sendt til kunden og producenten
Main Succes Scenario	<ol style="list-style-type: none"> 1. Brugeren starter oprettelse af nyt tilbud. 2. Systemet viser indtastnings mulighederne for en ny væg. 3. Brugeren indtaster væggens højde og bredde. 4. Systemet beregner antallet af lodrette ruder ud fra at højde skal ligge så tæt på 60 cm som muligt. 5. Systemet beregner antallet af fag, ud fra at bredden på ruderne skal ligge så tæt på 45 cm som muligt. 6. Systemet genererer skitsen med det beregnede antal ruder og fag. 7. Systemet opdaterer prisen ud fra den indtastede højde/bredde og de beregnede antal ruder og fag. 8. Systemet viser væg design og tilbudsprisen. 9. Brugeren bekræfter den viste væg design og tilbudsprisen. 10. Systemet viser kontaktformularen til indtastning af kundeinformation. 11. Brugeren angiver navn, email og telefonnummer. 12. Systemet laver e mail og sender med vedhæftede tilbud samt skitse.

UseCase navn	UC Lav nyt tilbud
	<p>1a. Hvis brugeren åbner "Kataloget"</p> <ol style="list-style-type: none"> Systemet viser en oversigt over standard væg opsætninger. Brugeren vælger en væg opsætning uden dør. Systemet viser min og maks bredde/højde på den væg opsætning der er valgt. <p>Processen fortsætter ved punkt 3.</p> <p>1a 2a. Brugeren ønsker at vælge væg med dør.</p> <ol style="list-style-type: none"> Brugeren vælger en standard opsætning med dør. Systemet viser min og maks bredde/højde på den væg opsætning der er valgt. Brugeren angiver væggens højde og bredde. Systemet viser en oversigt over greb typer. Brugeren vælger en greb type. Processen fortsætter ved punkt 4. <p>3a. Mangler at udfylde information om højde.</p> <ol style="list-style-type: none"> Systemet oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 3. <p>3b. Hvis brugeren angiver negativ tal om højde.</p> <ol style="list-style-type: none"> Systemet oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 3. <p>3c. Hvis brugeren angiver højden mere end 250 cm.</p> <ol style="list-style-type: none"> Systemet oplyser at max højde er 250 cm og afventer brugerinput. Processen fortsætter ved punkt 3. <p>5a. Mangler at udfylde information om bredde.</p> <ol style="list-style-type: none"> Systemet oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 5. <p>5b. Hvis brugeren angiver negativ tal om bredde.</p> <ol style="list-style-type: none"> Systemet oplyser om fejl og afventer brugerinput. Processen fortsætter ved punkt 5. <p>5c. Hvis brugeren angiver bredden mere end 150 cm.</p> <ol style="list-style-type: none"> Systemet oplyser at væggen vil blive sammensat af flere elementer. Processen fortsætter ved punkt 5. <p>9a. Hvis brugeren ønsker at tilpasse designet (speciel type glas)</p> <ol style="list-style-type: none"> Systemet viser en oversigt over mulige glastyper. Brugeren vælger en glastype. Processen fortsætter fra punkt 8. <p>9b. Hvis brugeren ønsker at tilpasse designet (tilvalg af dør).</p> <ol style="list-style-type: none"> Systemet viser en oversigt over mulige dørtyper. Brugeren vælger en dør. Systemet viser en oversigt over greb typer. Brugeren vælger en greb type. Processen fortsætter fra punkt 8. <p>9c. Hvis brugeren ikke bekræfter tilbuddet.</p> <ol style="list-style-type: none"> Forløbet starter forfra ved punkt 1.
Extensions	
Special Requirements	-
Technology and Data Variations List	-
Frequency of Occurrence	Use casen forekommer ved oprettelse af en designe væg
Miscellaneous	-

Bilag 8: Usecase diagram



Bilag 9: FURPS+

FURPS+

Functionality

- Skal hjælpe brugeren, med at lave et tilbud.
- Skal have et design som følger sammen tema, som det der brugt på New-Yorker.dk's hjemmeside.
- Skal hjælpe producenten, med at lave en produktionsklar tegning.

Usability

- Applikationen skal være let at benytte, og skal støtte brugeren mest muligt, ved at præsentere alle tilvalg overskueligt.

Reliability

- Alle beregninger, og skitsering, foregår lokalt på telefonen for at sikre hurtig behandling.

Performance

- Udfregninger af tilbud, samt udfærdigelse af skitse, skal ske hurtigst muligt, så brugeren ikke oplever ventetid.

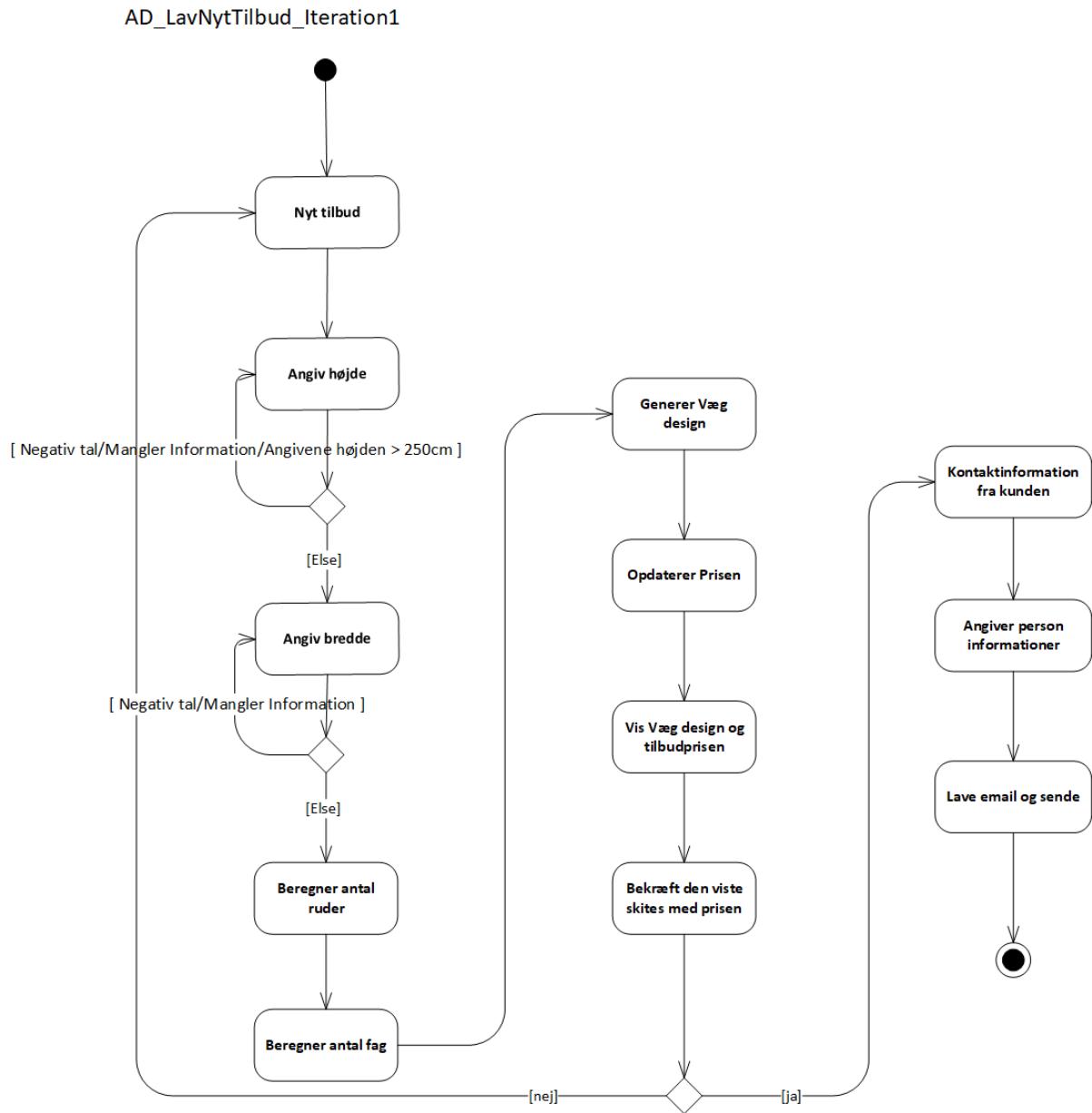
Supportability

- Applikationen skal være understøttet på Android, som udgangspunkt. Det er dog muligt, at iOS skal understøttes på et senere tidspunkt.

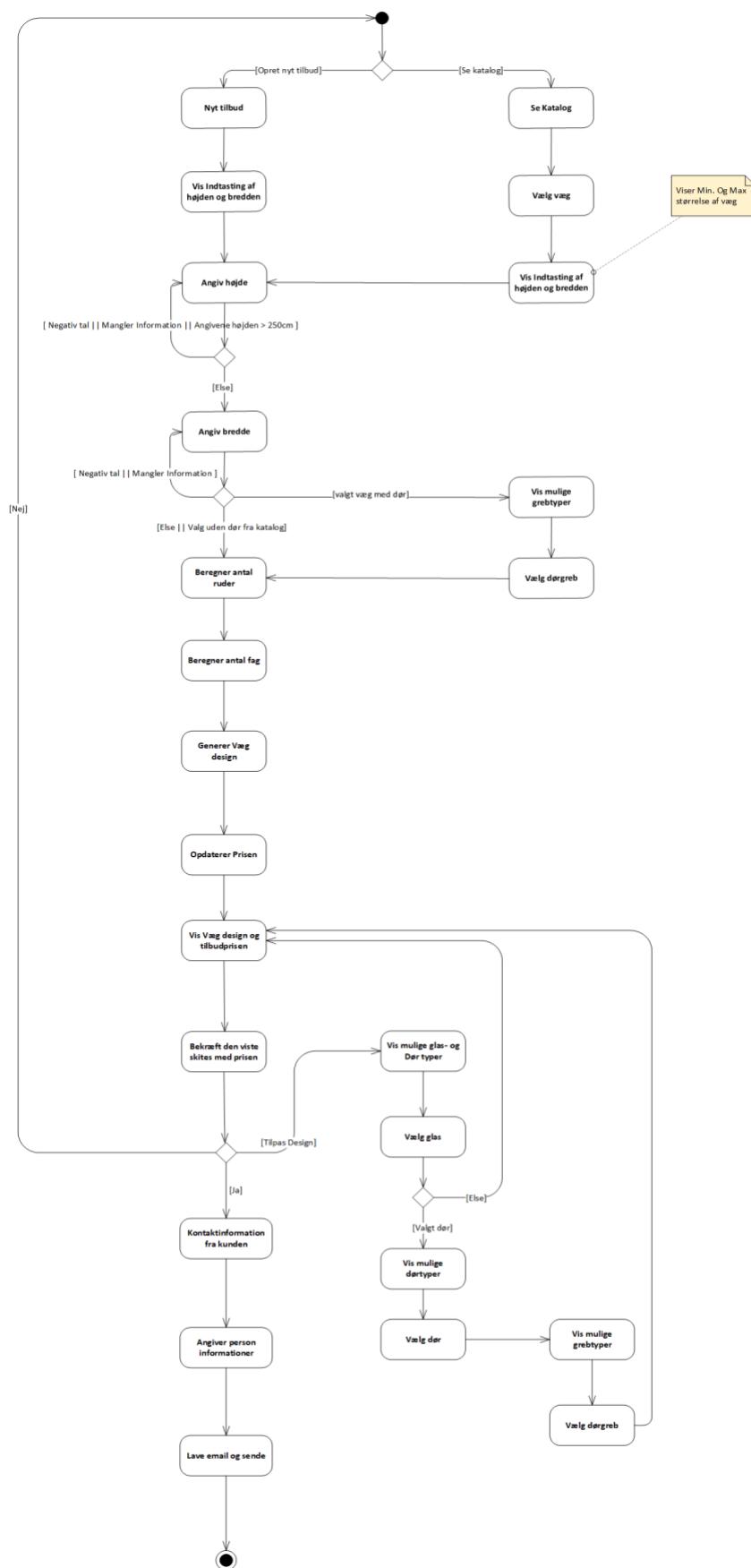
Precision

- Alle målangivelser på skitserne skal være korrekt detaljerede, så de kan anvendes i produktionen.

Bilag 10: Aktivitetsdiagram_iteration1 og iteration2

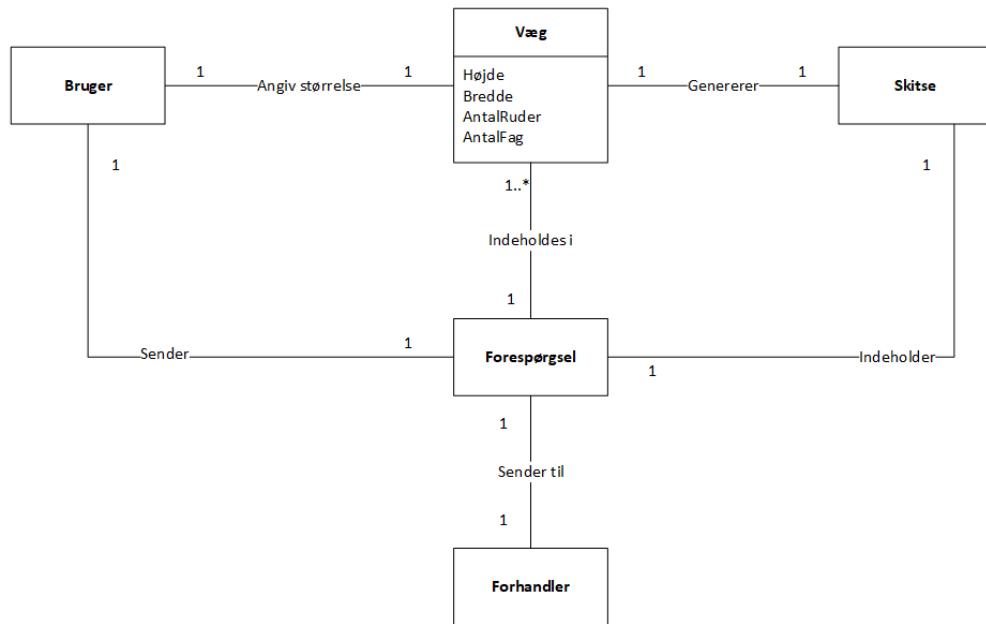


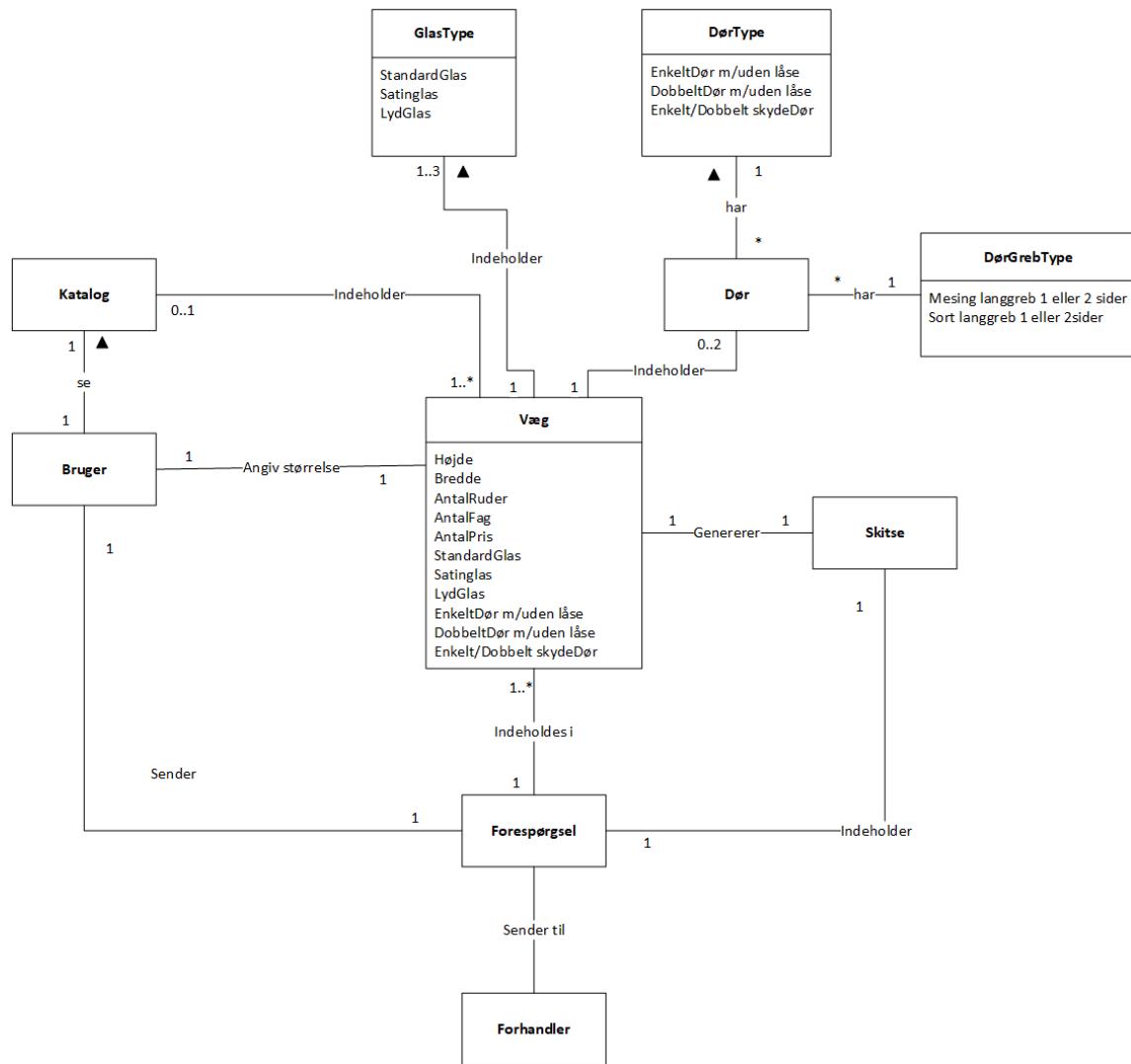
AD_LavNytTilbud_Iteration2



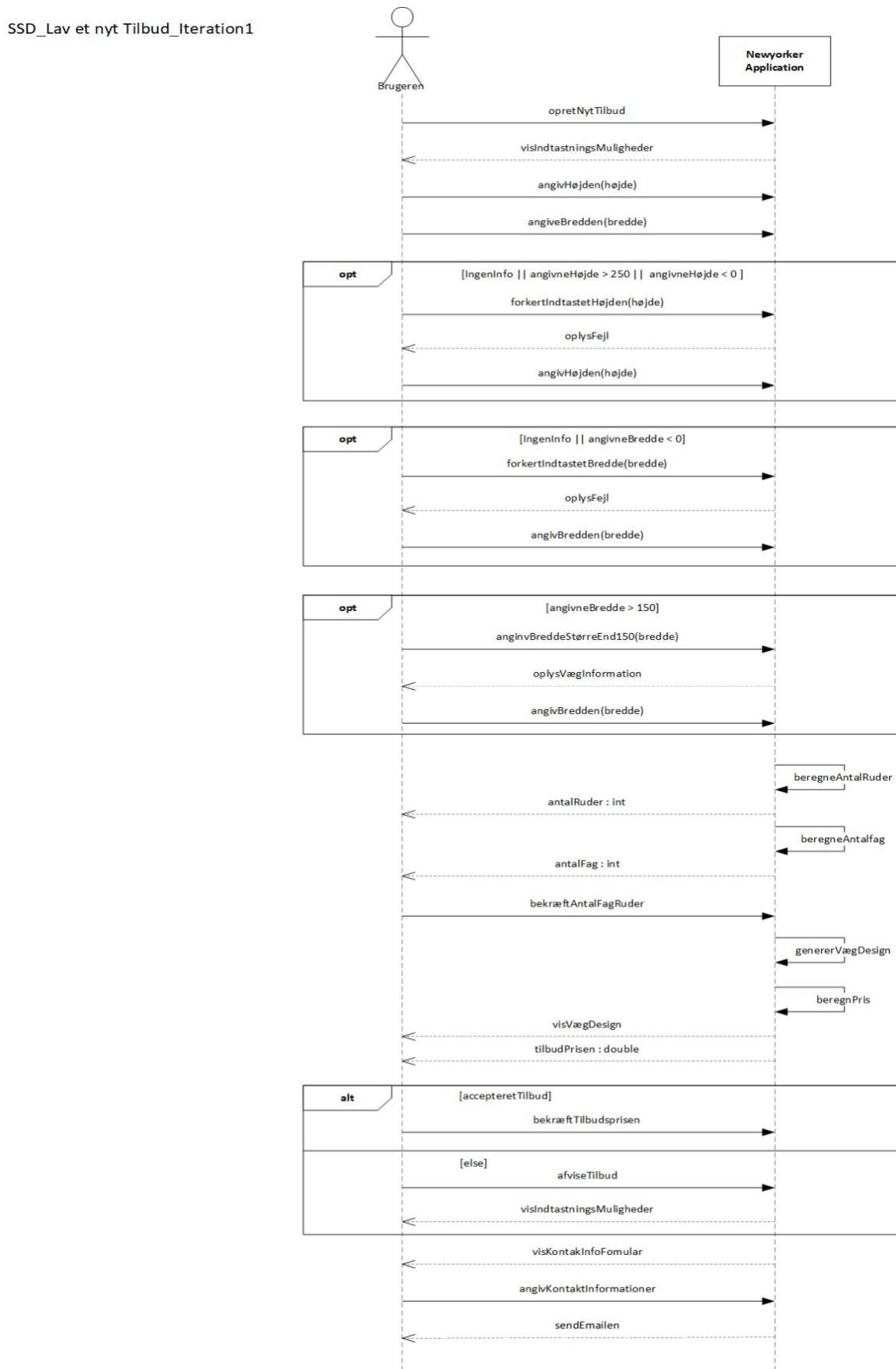
Bilag 11: Domænemodel_Iteration 1 og iteration2

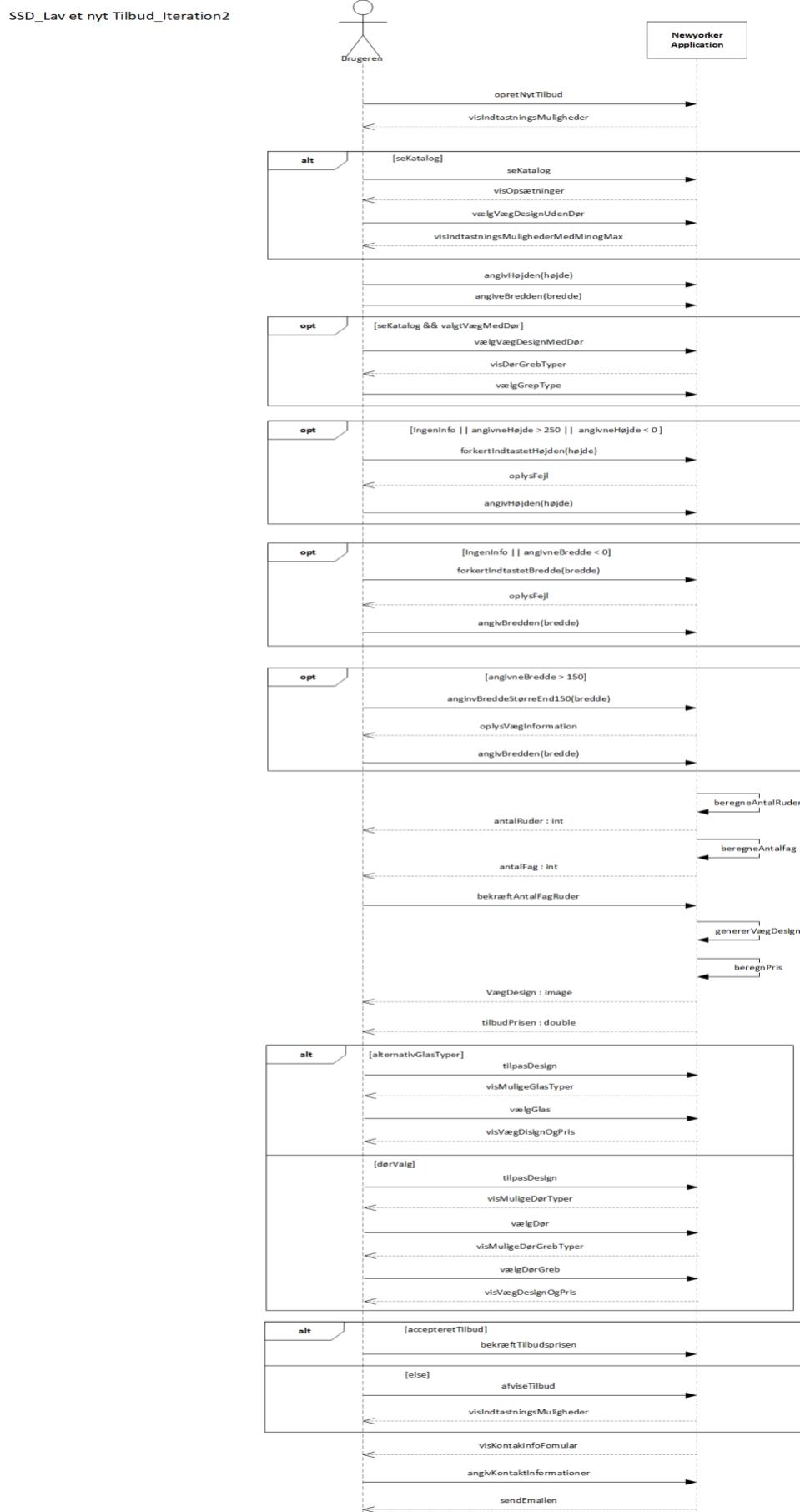
DM_LavNytTilbud_iteration1





Bilag 12: System sekvens diagram_iteration1 og iteration2





Bilag 13: Operationskontrakt_iteration1 og iteration2

OC01_opretNytTilbud

Operation:

Åbner interface, så et tilbud kan registreres.

Cross-reference:

UC_Lav nyt tilbud(Iteration1)

Pre-conditions:

- Android telefonen er tændt.
- Applikationen er installeret, og startet.
- Brugeren er klar, til at designe.

Post-conditions:

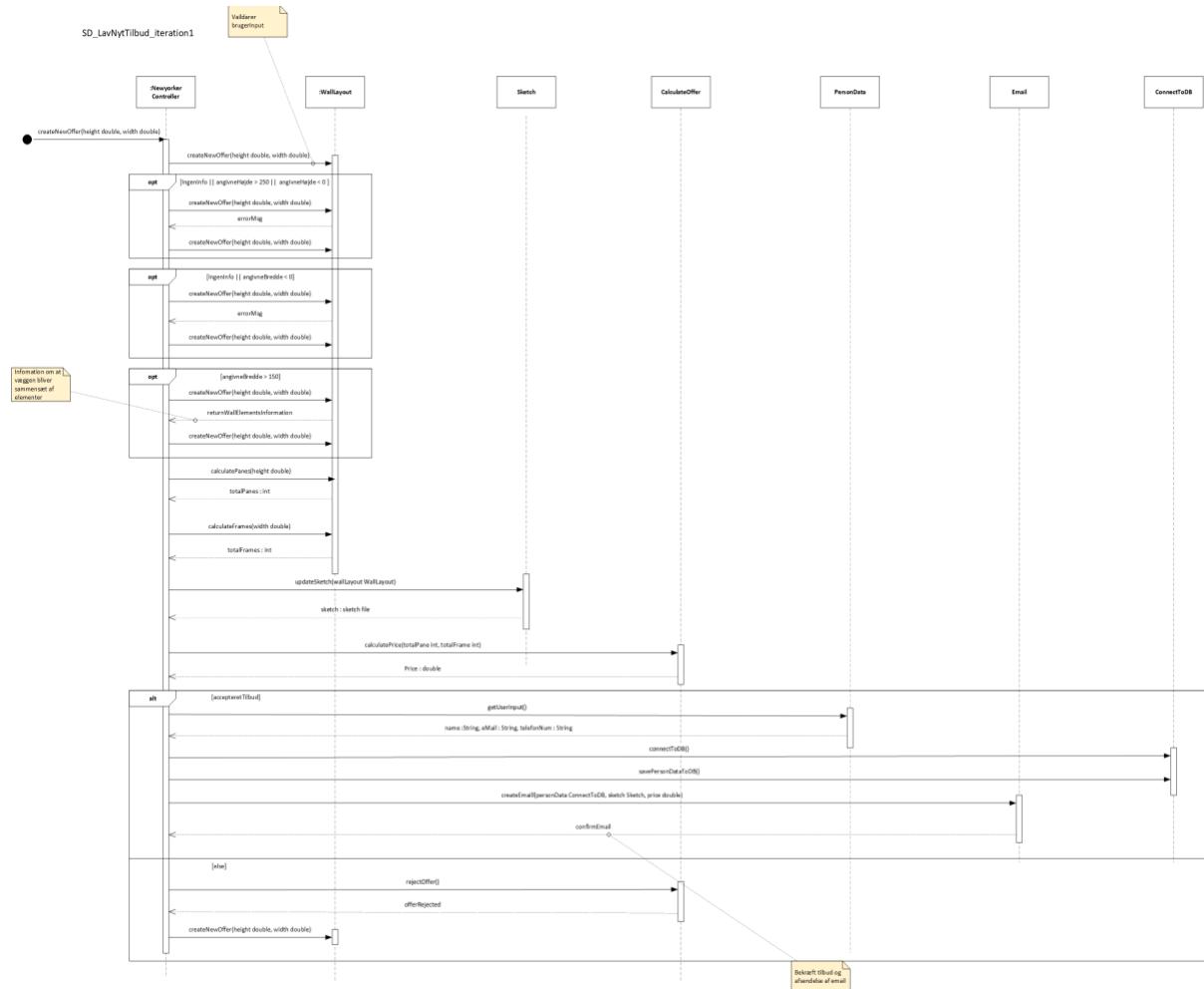
- Systemet er klar til at modtage input om højde og bredde

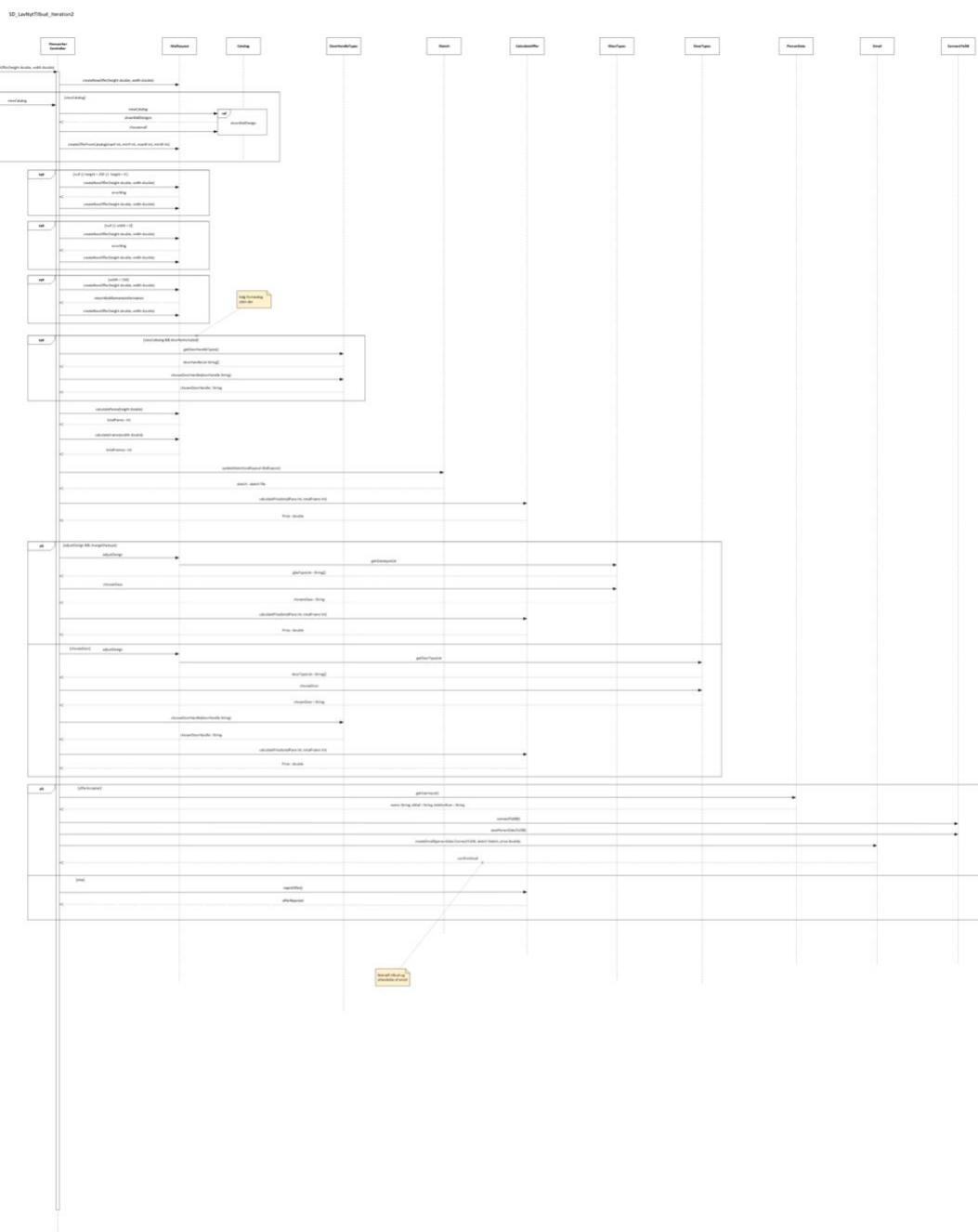
De resterende operationskontrakter kan findes på nedenstående link

<https://github.com/Jenna-P/NewYorker/tree/main/03%20OOA/Operationskontrakter>

Bilag 14: Sekvensdiagram _iteration1 og iteration2

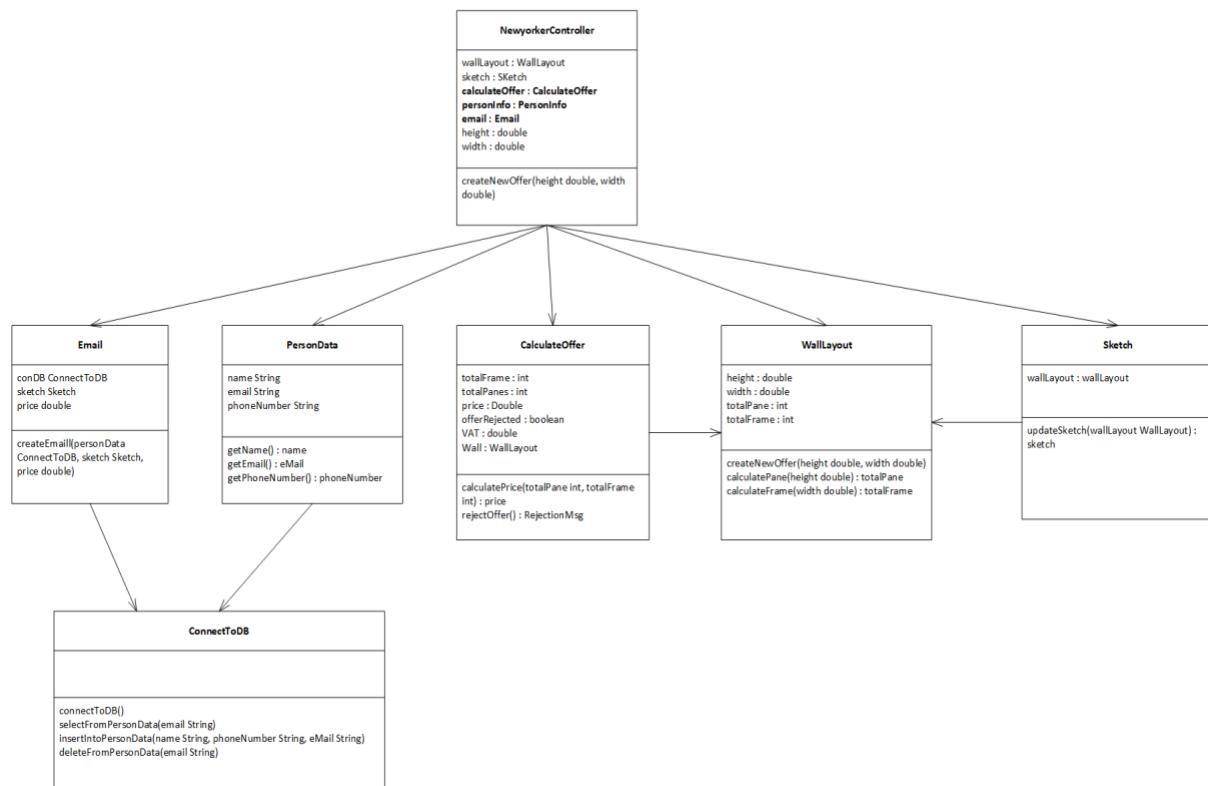
iteration1

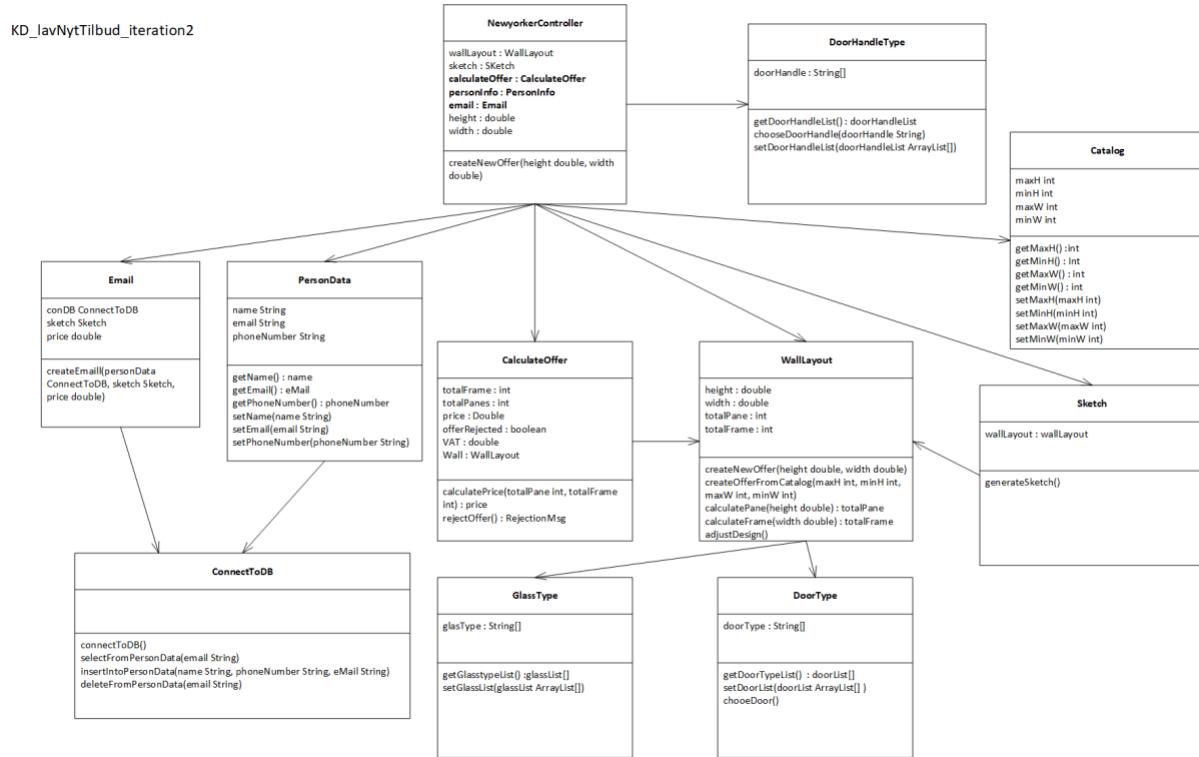




Bilag 15: KlassDiagram_iteration1 og iteration2

KD_lavNytTilbud_iteration1





Link

Guide til Playstore

<https://bolter.com.au/launch-how-to-release-your-app-on-the-google-play-store/>

GitHub

<https://github.com/Jenna-P/NewYorker.git>