

17-12-2021

Projekt Kalkuleringsværktøj

2. Semester eksamensprojekt. KEA –
Københavns erhvervsakademi

KRAM

Kamille Annemone Nikolajsen: 17-02-1995

[@KamilleNikolajsen](<https://github.com/KamilleNikolajsen>)

Rasmus Lundberg Kibshede: 03-02-1993

[@Rasmus-Kibshede](<https://github.com/Rasmus-Kibshede>)

August Hauerslev: 18-05-1998

[@Theecapain](<https://github.com/TheeCapain>)

Michala Nim-Melchiorson Nybroe: 18-06-1997

[@MichalaNybroe](<https://github.com/MichalaNybroe>)

Heroku og github:

<https://projectcalculationtool.herokuapp.com/>

<https://github.com/KamilleNikolajsen/ProjectCalculationTool>

Indholdsfortegnelse

Introduktion.....	4
Problemstilling.....	4
Projektafgrænsning	4
Projektets forudsætninger	5
Teknologivalg.....	5
IntelliJ IDEA - version 2021.2.3(Ultimate Edition)	5
MySQL Workbench - version 8.0.22	6
Spring Boot - version 2.6.0	6
Thymeleaf - version 3.0.14	6
Apache Maven - version 4.0.0.....	6
JDBC - version 4.3	6
Selenium - version 3.9.1	6
Værktøjer.....	7
Github.....	7
Heroku	7
Simply	7
Google Drive	7
Visual Paradigm	7
Virksomhedsanalyse.....	8
Interessentanalyse.....	8
Interessent Akse	8
Udvidet risikoanalyse	9
Gennemførligheds overvejelser	10
Krav.....	10
Use cases	11
Use case: Register project	12
Use case: Maintain project.....	15
Use case: Login og se projektoversigt	18
Domæne model.....	18
FURPS+.....	19
Arbejdsprocess	20
Gruppedynamik.....	21
Rollefordeling ved Belbin.....	21

Arbejdstilgang.....	22
Unified Process	22
Iterationsplan	22
Brug af kundeproxy	23
Par programmering	23
Koordinering	24
Planlægning	24
Github	25
Analyse og design	26
Brugerfladedesign	27
Prototype	27
Diagrammer	27
Start klassediagram	27
System oversigt	28
Patterns	33
MVC	34
GRASP	34
Singleton	36
Implementering	36
Programstruktur	36
Overholdelse af domæne struktur	37
Forbindelsen til databasen	39
Særlige forhold	41
Validering	41
Exceptions.....	42
Session og sikkerhed.....	43
Web	44
De otte gyldne regler	44
Gestaltlovene.....	45
Fontawesome	46
HTML/CSS	48
Spring Boot	50
Thymeleaf	52
Database.....	53

SQL.....	54
Cascade.....	55
Test	55
JUnit test.....	55
Integration test.....	57
Ui test - Selenium	57
Konklusion	58
Mangler/refleksion - Hvor langt nåede vi?.....	59
Videreudvikling.....	59
Litteraturliste.....	60
Bilag	62

Introduktion

I denne rapport fremføres fire studerendes projektudvikling af et projekt kalkuleringsværktøj for Alpha Solutions. Kunden Alpha Solutions fremførte i begyndelsen af projektet et oplæg om ønskede system med Klaus Petersen som repræsentant. Alpha Solutions er en it- og e-commerce virksomhed der skaber digitale løsninger for deres kunder. Deres kunder er virksomheder, som har behov for rådgivning indenfor e-handel og tekniske implementeringer (<https://www.alpha-solutions.com/da/om-os> 03/12-21). Som agerende kunde gennem projektforsløbet har gruppens undervisere stået til rådighed.

I rapporten fremlægges gruppens arbejdsprocess fra projektforsståelse og projektafgrænsning til implementering af problemstilling.

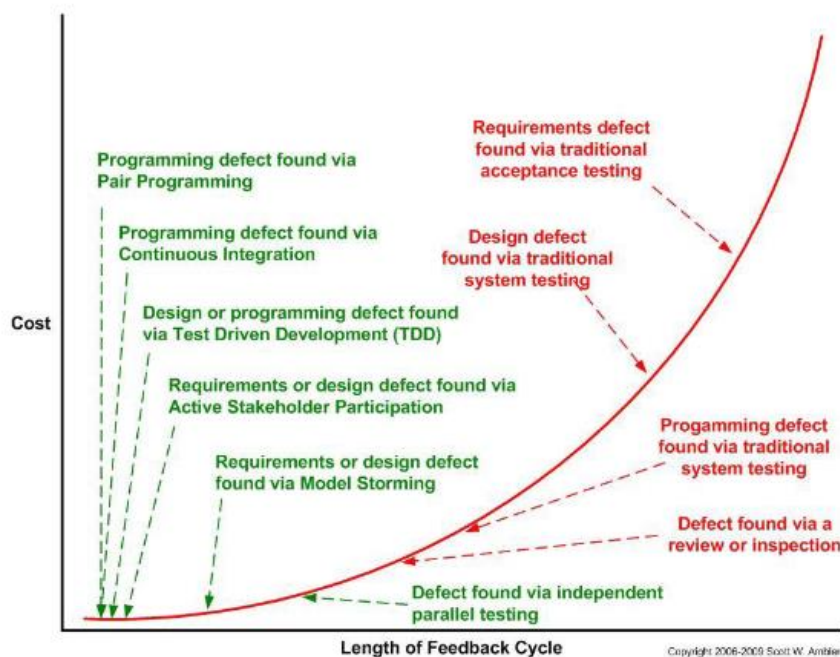
I starten af projektstart vedtog gruppen i samtale med vejleder at ingen navne på rapport eller program skulle skrives. Alle i gruppen står inde for alle dele, af både system, diagrammer, analyser og rapport.

Problemstilling

Ud fra oplægget fremført af Alpha Solutions har gruppen forstået projekt kravet til at være et projekt kalkulationsværktøj, hvormed systemet skulle være et webbaseret værktøj til registrering af projekter. Her var kerneopgaven at et projekt skulle kunne nedbrydes til mindre dele samt kunne registrere tidsforbrug til senere kalkulerings af arbejdsdage og løbende kunne vedligeholdes. På sigt blev der opfordret til flere styringsfeatures som kunne tilføjes til systemet.

Projektafgrænsning

Alpha Solutions kom med mange ideer til udførelsen af deres kalkulationsværktøj, hvor gruppen har foretaget en afgrænsning, således systemet kunne leve op til minimal viable product (MVP) og herfra videreudvikles. Det blev vurderet ud fra projektets tidsramme at kerneopgaven ville være et godt udgangspunkt for videre udvikling af flere features for systemet.



Ovenstående ses grafen for teknisk gæld, denne omhandler en evige dualitet i prioritering mellem højere omkostninger og hvornår man modtager feedback. Som grafen viser kan man optjene teknisk gæld ved at negligere løbende feedback, hvor gruppen har benyttet sig af nogle af disse tilgange som vil fremsættes i “Arbejdsproces” afsnittet.

Med udgangspunkt i grafen fokuseres der på at lave kerneopgaven robust frem for at tilføje flere features. Således ønskes prisen af systemet at holdes nede særligt da perioden på fire uger ikke udelukkende måtte bruges på systemets udvikling. Desuden har gruppen valgt ikke at implementere et registreringssystem af ansatte, da systemet forventes integreret i Alpha Solutions eget interne system.

Projektets forudsætninger

Her fremsættes teknologivalg og værktøjer, som gøres brug af i projektet. Dette gøres med henblik på et hurtigt overblik og ikke en detaljeret indføring i de forskellige elementer.

Teknologivalg

IntelliJ IDEA - version 2021.2.3(Ultimate Edition)

Til udførelsen af projektet har gruppen gjort brug af udviklingsmiljøet IntelliJ IDEA version 2021.2.3 udviklet af JetBrains til udvikling af computersoftware. Brugen af IntelliJ IDEA gør det særligt nemt at teste sit program via Junit testing og code coverage samt hurtigt at finde fejl via debugging.

MySQL Workbench - version 8.0.22

Til at håndtere data oprettes en connection gennem MySQL Workbench til projektets database, hvor systemets lagring af data er.

Spring Boot - version 2.6.0

Gruppen gør brug af Spring Boot, et java framework, til webudvikling.

Thymeleaf - version 3.0.14

Thymeleaf er et bibliotek gruppen gør brug af til dynamisk webudvikling(<https://www.baeldung.com/thymeleaf-in-spring-mvc>, 07-12-2021).

Apache Maven - version 4.0.0

Maven er et redskab der bruges til at bygge og vedligeholde java baserede projekter (<https://maven.apache.org/what-is-maven.html> 1/12-21). I ens projekt, der gør brug af maven, anvendes denne gennem en POM (project object model) fil til at styre projektets opbygning, rapportering og dokumentation(<https://maven.apache.org/> 1/12-21). Maven er således ansvarlig for at holde styr på projektets afhængigheder og integration med andre frameworks og biblioteker som Spring Boot og Thymeleaf gennem dets pom.xml fil.

JDBC - version 4.3

Java database connectivity(JDBC) er et java API(Application Programming Interface) der håndtere forbindelsen til databasen, via queries og kommandoer(<https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html>, 11-12-2021).

Selenium - version 3.9.1

Selenium bruges til at automatisere webapplikationer til testning ved emulering af webbrowser (<https://www.selenium.dev/> 4/12-21). Hermed kan selenium bruges til at user interface (UI) teste programmet ved at simulere brugeradfærden.

Værktøjer

Gennem projektet har gruppen gjort brug af værktøjer til; at samarbejde på projektet, gøre det muligt at hoste både databasen og webapplikationen samt at udføre UML diagrammer.

Github

Gennem udviklingen af dette Kalkulationssystem har gruppen gjort brug af github.com til lagring og deling af programmet. Github er en online platform for lagring af softwareprojekter, der gør brug af “distributed workflows”. Dette gør det fleksibelt for samarbejde via git ved at hver udvikler kan både opretholde et repository og udvikle på samme([Distributed Workflows](#) 30/11-21).

Heroku

Heroku er en webhost, som understøtter java og kan håndtere forskellige databaser. Systemet er deployet på Heroku, så andre kan tilgå den via en webbrowser.

Simply

Gruppen har i projektet gjort brug af en MySQL database, version 5.7.35-38 se bilag 1, som er hostet hos webhosten Simply.com. Grunden til der er gjort brug af en database host hos Simply og ikke Heroku, er tidligere erfaringer ved brug af gratis hosting, hvor data pludseligt gik tabt, hvorfor gruppen valgte at fortsætte hos Simply fra forrige projekt. Databasen er sat op til en online server, hvilket gør det muligt for hele gruppen at bruge den.

Google Drive

Google Drive har fungeret som et samlingspunkt for gruppens arbejde. Dette har været med til at give et overblik over tidsplanen, samt de forskellige diagrammer og skriftlige delelementer projektet indeholder.

Visual Paradigm

Til visualisering og planlægning af programmet, er flere diagrammer udviklet i visual paradigm, et program specifikt designet til at lave modeller og diagrammer i UML(Unified Modelling Language).

Virksomhedsanalyse

Her fremsættes projektets virksomhedsanalyse bestående af interessentanalyse, udvidet risikoanalyse, og vurdering om gennemførlighed.

Interessentanalyse

Herunder vises de fem interessenter.

KEA ledelse:

KEA ledelse vurderes til at være en del af interessenterne, da de sætter rammerne for projektets form. Deres indvirkning er dog primært i projektets opstartsfase. Desuden er dette en interessant for projektarbejdsvilkårne med henblik på coronapas.

Alpha Solutions:

Alpha Solutions sætter rammerne for projektet og sætter specifikt kravene, som gruppen skal opnå. Der bliver ikke forventet at de har en yderligere indvirkning på projektet, men dog at de har en interesse i resultatet af projektet med henblik på eventuelle ansættelser i fremtiden.

Alpha Solutions kunder:

Hvis dette system bliver taget i brug vil Alpha Solutions kunder ses som en interessant. Hvormed de vil få et estimat på tidsforbrug nødvendig for projektet de ønsker rådgivning omkring.

Alpha Solutions konkurrenter:

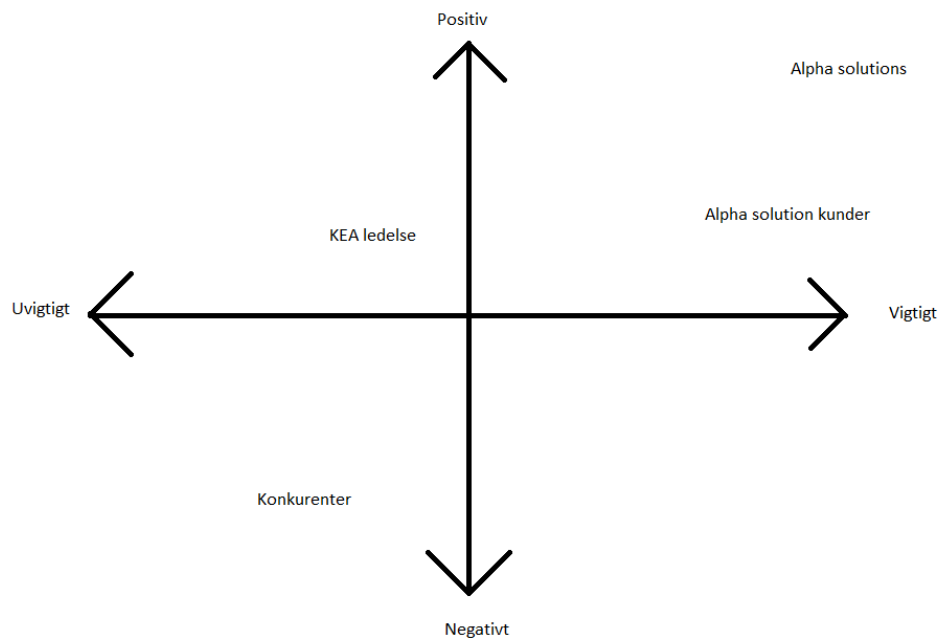
Alpha Solutions konkurrenter forventes at bruge lignende kalkulationsværktøjer, og vurderes til at have interesse i hvad Alpha Solutions gør brug af. Dog forventes de ikke til at have nogen indvirkning på dette projekt.

Vejledere:

Vejlederne agere proxy for Alpha Solutions, hvormed de rådgiver gruppen om produkt forventninger og om kravene mødes. Desuden justere vejlederne kravene med henblik på eksamen.

Interessent Akse

Her ses interessant aksen, hvor de forskellige interessenter er plottet ind, med henblik på deres holdninger og vigtigheden heraf i forhold til projektet.



Idet overordnede analyse, er lavet uden adgang til slutkunden arbejdes der med et konstrueret projekt, hvor interessant analysen ikke forventes at have effekt på projektet.

Udvidet risikoanalyse

Den udvidede risikoanalyse, blev udført med ønsket om på forhånd at håndtere de problematikker som gennem projektforsløbet kunne opstå via forudbestemte løsningsforslag.

Udvidet risikotabel for IT-udviklingsprojekt							
Risikomoment	Sandsynlighed (1-5)	Konsekvens (1,3,7,10)	Produkt (s * k)	Præventive tiltag	Ansvarlig	Løsningsforslag	Ansvarlig
Gruppemedlem bliver syg fx Corona	3	7	21	Vær opdateret på hinandens arbejde, så koden er let tilgængelig for andre gruppemedlemmer.	Michala	Hjemsende syge gruppemedlem, så de kommer hurtigt i bedring, og ekskludere ham/hende fra to do liste indtil de igen kan arbejde.	Gruppen
Mister adgang til database kort før aflevering	1	10	10	At holde vores script opdateret, også med test data.	Kamille	Kontakt udbyder support	Rasmus
Kunde ændrer kravspecifikationer	1	10	10	Spørge vejleder med henblik på problemforståelse og problemløsnings tiltag. Gøre koden nem at vedligeholde/ændre ved at holde lagdelt arkitektur og overholde low coupling og high cohesion.	Gruppen	Tilpasse løsnings tiltag med nye krav.	Gruppen
Deployment til Heroku går galt	2	10	20	Slut deploy i god tid, for at have tid til at løse eventuelle problemer.	Rasmus	Kontakte Claus, for løsningsvejledning	August

Ovenstående skema viser de risici med et produkt på 10 eller over, hvor et præventivt tiltag med en ansvarlig person er fremskrevet og hertil yderligere et løsningsforslag med den ansvarlige for dette.

Gennemførligheds overvejelser

Det blev af gruppen vurderet ikke nødvendigt at udføre et feasibility studie med henblik på projektet, idet Alpha Solutions og KEA satte rammen at det skal udføres og gruppen ingen indvirkning havde på dette. Supplerende til dette var gruppen klar over at Alpha Solutions allerede ligger inde med et sådant kalkulationssystem, samt at andre systemer, udviklet med flere ressourcer, allerede er på markedet. For eksempel er projectmanager.com udviklet med mulighed for statistikker, hvor man kan afkrydse, hvad som er lavet (<https://www.projectmanager.com/software> 12/12-21).

På baggrund af ovenstående stiller gruppen sig kritisk overfor om det er gennemførligt, men projektet erklæres gennemførligt uden videre analyse.

Krav

Systemet skal gøre det muligt for Alpha Solutions at nedbryde de opgaver de stilles af deres kunder, således de gennem denne nedbrydning kan se en opsummering af projektets tidsforbrug. Dette skal sikre at de selv kan leve op til de tidsrammer der stilles og kunne give kunderne en vurdering af hvor lang tid projektet vil tage.

I begyndelsen af projektet udførte gruppen en ordliste over problemstillingen til værktøj for klargørelse af use cases og til udvikling af UML diagrammer.

UML Diagrammer og Analyser kan findes i bedre opløsning på github(<https://github.com/KamilleNikolajsen/ProjectCalculationTool>).

Ordliste

Kerneopgave

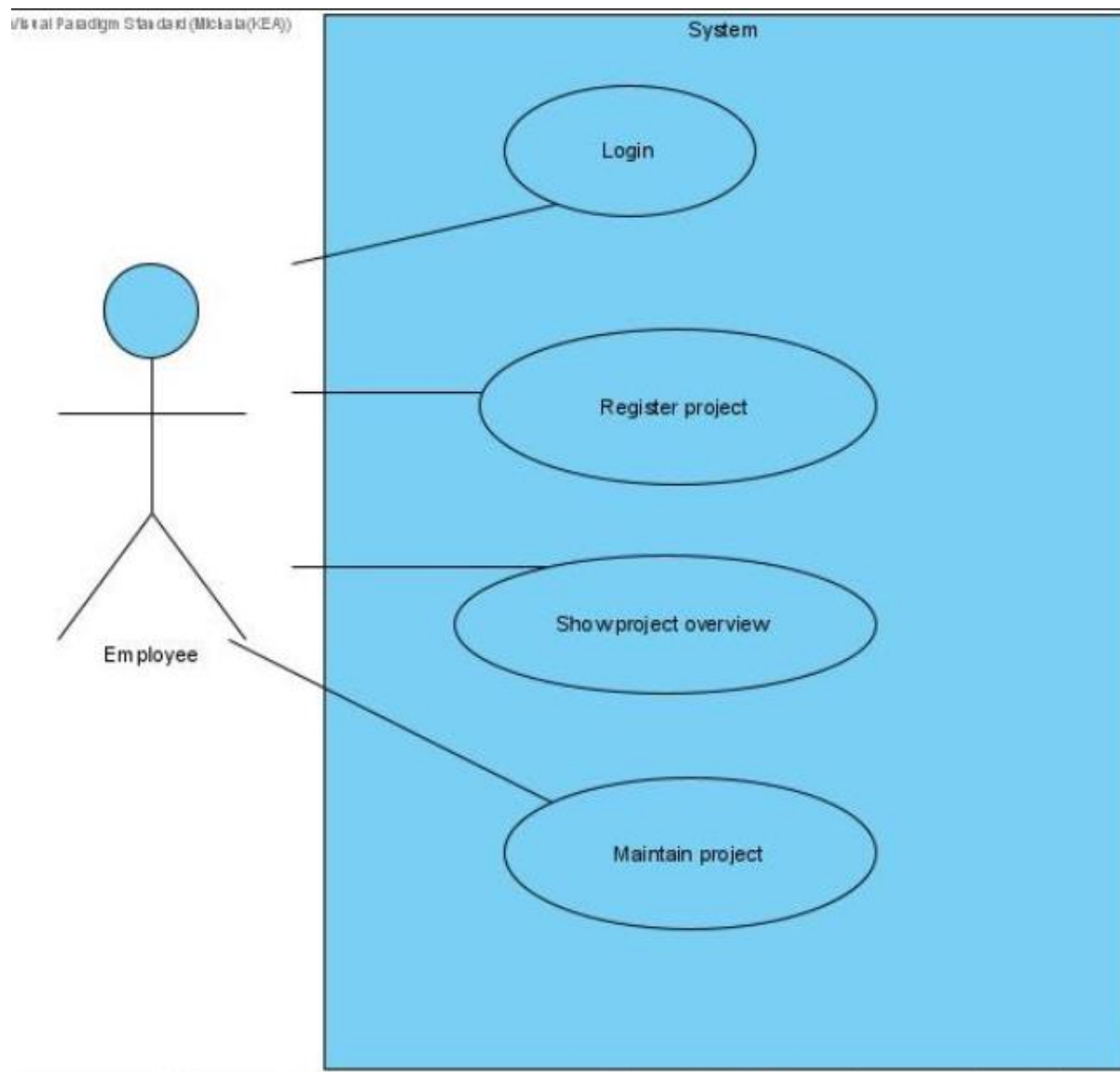
- Nedbrydning af projekt
- hjælp til kalkulation
- datamodel
 - tidsforbrug
 - projekt
 - delprojekt
 - opgaver
 - deadlines
- brugergrænseflade
- oprettelse og vedligeholdelse af projekt
- tidsmæssig nedbrydning,
- Time in days
- Time
- overblik

Supplerende features

- tidsforbrug på arbejdsdage - færdigt til tiden
 - deadline
- Resource/kompetencer
- ressource(type) og kompetence tidsforbrug
- belastning af ressourcer på arbejdsdag

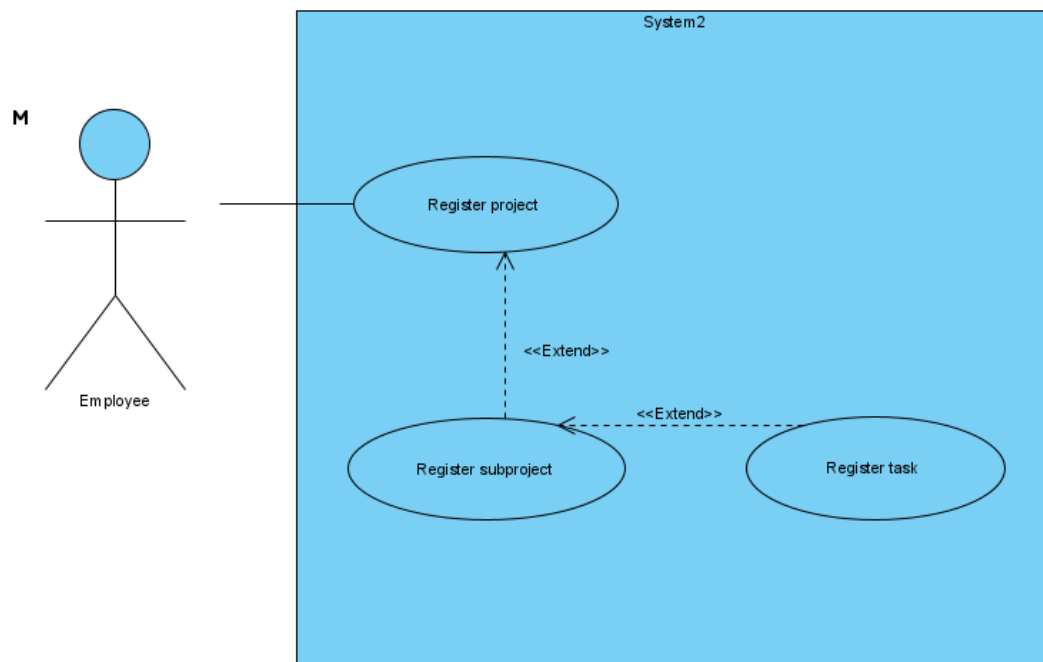
Ordlisten ovenfor er forsøgt opdelt i krav til systemet. ordlisten har fungeret som start værktøj til den videre fremskrivelse af use case og system klasser.

Use cases



Ovenfor ses en oversigt over projektets use cases, fundet på baggrund af ordlisten. At kunne registrere et projekt og nedbryde denne i mindre dele vurderes som den primære use case. Den sekundære use case er at kunne vedligeholde projektet. Udover disse findes to yderligere; at aktøren - den ansatte, kan se en projektoversigt, samt at den ansatte skal være en gyldig bruger i programmet forinden, at kunne se, oprette, slette eller redigere projekter.

Use case: Register project



Ovenfor ses modellen over første use case, hvordan den ansatte vil kunne registrere et projekt og der i forlængelse af dette kan registreres et delprojekt, som igen forlænges i registrering af en opgave. Herunder præsenteres beskrivelsen af den fully dressed use case over registrering af projekt.

Use case name	UC1 Register Project
Scope	Alpha Solutions projectCalculationTool
Primary actor	Alpha Solutions employee
Stakeholders and interests	<ul style="list-style-type: none"> Alpha Solutions employee: Wants easy to use, high quality project calculation tool with good time estimate function as inefficient time estimate will affect customer relations. Alpha Solutions customer: wants accurate information from hired company.

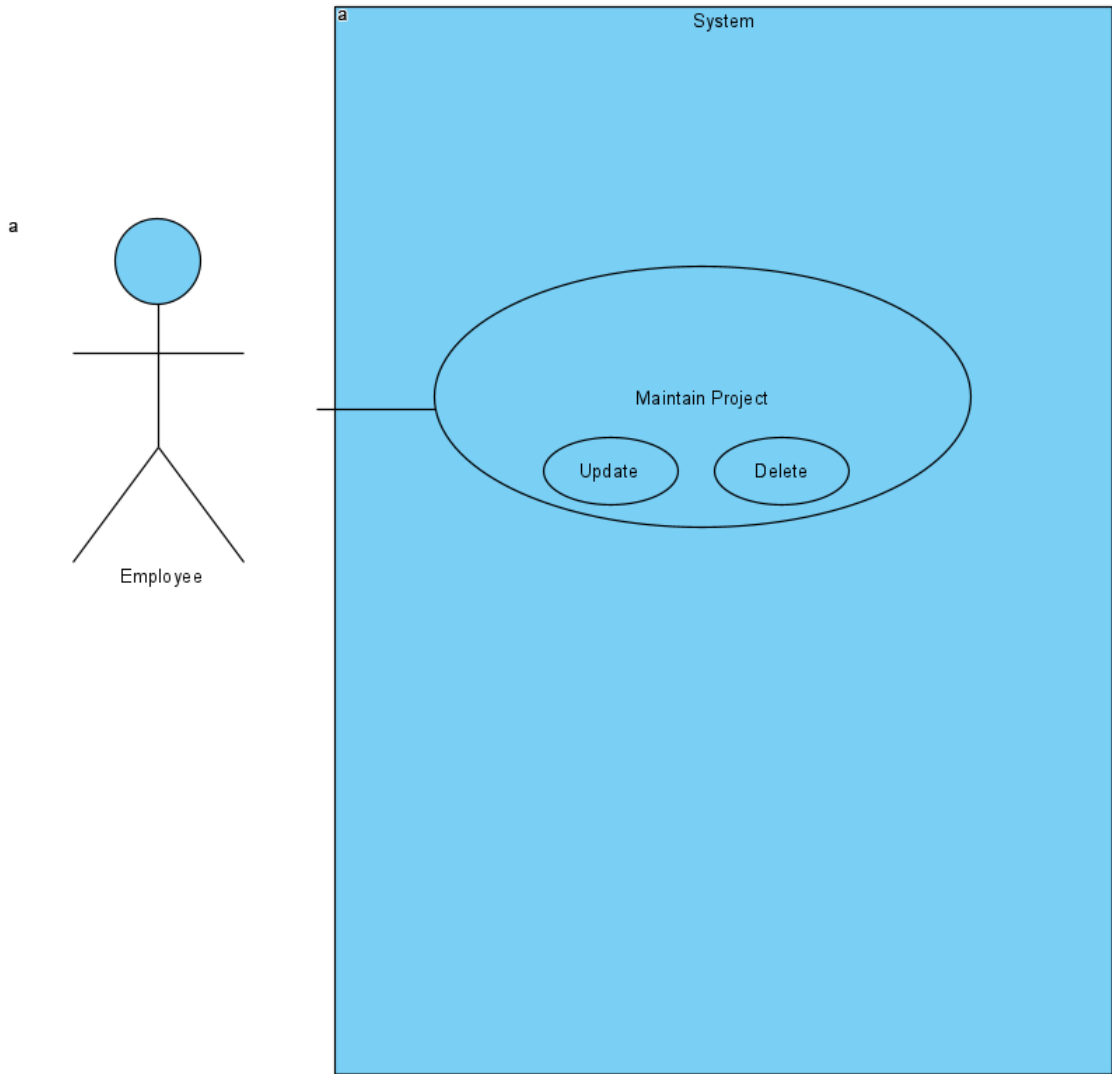
pre-conditions	Employee is identified via login and then authorized to create project.
Success guarantee	project is registered and projectTime is correctly calculated into days.
main success scenario	<ol style="list-style-type: none"> 1. Employee starts new project 2. System records project and presents project 3. Employee registers subproject 4. System displays registered subproject 5. Employee registers tasks 6. System displays registered tasks and calculated time for tasks, workdays for project and subproject. 7. Employee finishes project information and goes to the overview page to view summarized project information
Extensions	<ol style="list-style-type: none"> 1. Employee attempts to register project/subproject/task without a name <ol style="list-style-type: none"> 1. system signals error message to employee 2. Employee registers project/subproject/task with a name 2. employee attempts to register task without time <ol style="list-style-type: none"> 1. system signals error message to employee 2. employee registers task with time 3. employee attempts to register task time field with invalid input <ol style="list-style-type: none"> 1. system signals error message to employee 2. employee puts in valid input 4. employee attempts to register project/subproject/task with name longer than 45 characters <ol style="list-style-type: none"> 1. system signals error message to employee 2. employee enters valid input

Technology and Data variations list	Employee has to have a login to use system.
Frequency of occurrence	1-2 projects a week
Misc.	<ul style="list-style-type: none">-What resources are needed if given a timeframe for project.-What different types of employees are needed for different projects?-does every employee have same access to project registering?-explore accountability for vacation- and sick days
Special Requirements	none

Use case: Maintain project

Anden use case omhandler hvordan brugeren skal kunne vedligeholde projektet.

Nedenunder fremsættes modellen over anden use case, som viser, hvordan den ansatte skal kunne vedligeholde projektet gennem underliggende use cases, at kunne opdatere og slette.



Herunder præsenteres beskrivelsen af den fully dressed use case over vedligeholdelse af projekt.

Use case name	UC1 Maintain project
Scope	Alpha Solutions projectCalculationTool
Level	employee goal

Primary actor	<ul style="list-style-type: none">Alpha Solutions employee
Stakeholders and interests	<ul style="list-style-type: none">Alpha Solutions employee: wants to maintain an existing project.Alpha Solutions customer: wants to be able to review requirements and possibly change direction of project
pre-conditions	<ul style="list-style-type: none">Employee is identified via login and then authorized to create project.A project is in existence
Success guarantee	System can maintain projects via registering changes(updating and deleting).
main success scenario	<ol style="list-style-type: none">Employee clicks on a project to maintain.System displays project page.Employee clicks on edit element icon.System displays edit page for chosen element.Employee submits updated element information.System returns to project page - now updatedEmployee clicks delete icon on elementSystem displays project page -now having removed an element.
Extensions	<ol style="list-style-type: none">Employee attempts to update element with invalid data<ol style="list-style-type: none">System signals error message to employeeemployee either cancels or enters correct input

Technology and Data variations list	Employee has to have a login to use system.
Frequency of occurrence	often
Misc.	<ul style="list-style-type: none">• If implementation of resources - then how to maintain these.• Does every employee have the same access to project maintaining?
Special Requirements	none

Som beskrivelsen ovenfor viser, kan den ansatte redigere projekt, delprojekt og opgave navn. Desuden kan den ansatte også slette alle de forskellige elementer; altså en opgave, et delprojekt og selve projektet.

Use case: Login og se projektoversigt

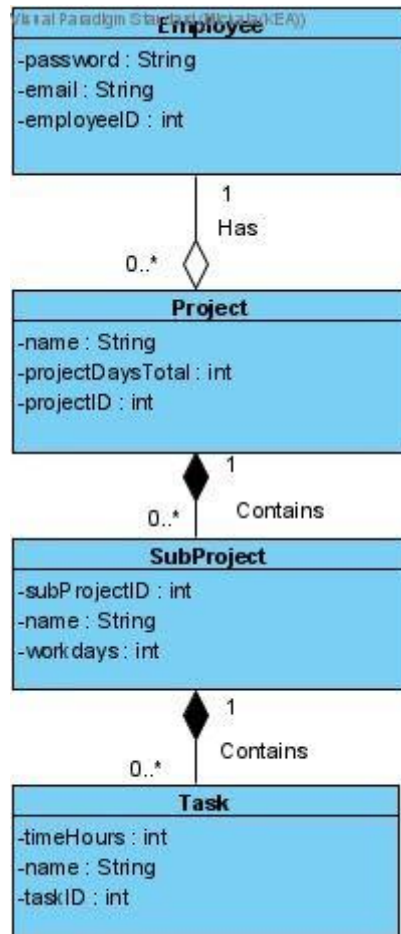
Udover de to fully dressed use cases, fremføres her de to resterende use cases i et kort format.

Login er tænkt til at skulle integreres med internt system i Alpha Solutions, således forventes det at dette projekt ikke skal kunne registrere brugere. Men det vurderes at sikkerhed omkring at være logget ind for at kunne tilgå de forskellige projekter, er essentielt. Derfor lyder login use casen på at en ansat skal møde en login side og herfra komme til resten af systemet.

Use casen's projektoversigt skal gøre det nemt for den ansatte at se igangværende projekt og detaljer vedrørende dette. Derfor lyder use casen at den ansatte skal kunne klikke på projektoversigt fra profil siden.

Domæne model

Domæne modellen fungerer som et overblik over de konceptuelle klasser. Den har været et værktøj for den videre kodning af modeller i systemet.



De konceptuelle klasser som set ovenfor er; en ansat, et projekt, et delprojekt og en opgave. Som ses på billedet er det alle projektdelte som består af en attribut med tid, en med navn og et id. Den sidste klasse, en ansat, består af et id, en adgangskode og en email.

FURPS+

Her fremsættes de funktionelle og ikke-funktionelle krav (Larman, 2015, s 57, 107), der gør sig gældende over flere af use casene.

Funktionalitet

Den eneste måde at bruge projektets webapplikation på, er ved at have adgang til internettet. Desuden kan kræves visse programmer på ens computer. Det er desuden et behov at have en allerede eksisterende bruger for at kunne gøre brug af systemet.

Anvendelighed

Til at bruge systemet, er forhåbningen, at man ikke skal kende til brug af denne typer af systemer forinden. Det har været meningen at gøre det nemt at bruge, således det er intuitivt at navigere.

Reliabilitet

Gruppen har forsøgt at håndtere alle fejl som måtte opstå og udskrive fejlbeskeder til brugeren af program. Den eneste kendte fejl, hvor en fejlkode ikke bliver sendt er hvis en ansat ikke har internetforbindelse, og de vil i dette tilfælde blive mødt af en HTTP fejlkode 500. Dette kan dog håndteres ved at oprette forbindelse til internettet og genindlæse hjemmesiden.

Ydeevne

Hvert repository i webapplikationen forbinder til den online database, så flere forbindelser til den online database skabes, hvilket nedsætter responstiden. En mulig løsning ville være at bruge joins i et sql forbindelse til databasen, som der så ville håndtere et resultset i de forskellige repository lag og ikke forbinde flere gange. Det er blevet fra prioriteret på grund af omlægning af brugergrænsefladen og der ses ikke et behov for optimering af responstid.

Støtte/vedligeholdelses

Da fokus har været at kode et kerneprodukt med lille teknisk gæld bør det være muligt at videreudvikle på systemet herunder at tilføje flere funktioner til det. Gruppen har desuden forsøgt at gøre koden så letlæselig som muligt, for at gøre det lettere for andre at sætte sig ind i koden og vedligeholde den.

Implementering

I udarbejdelsen af dette projekt kalkulationsværktøj, har det været en forudsætning, at gøre brug af java, jdbc, maven, mysql, html, css, thymeleaf, spring boot og at deploye webapplikationen på heroku.com. Dette medfører at man gør brug af en compiler der understøtter det, hvor gruppen derfor gjorde brug af IntelliJ IDEA. Det er tænkt at projekt kalkuleringsværktøjet integreres i et allerede eksisterende internt system hos Alpha Solutions. Herudover skal Alpha Solutions ansatte have adgang til internet og web browser, som systemet understøtter.

Fordelingen af arbejdsressourcer er en uge på forståelse og planlægning af projektet, to uger på udarbejdelse af system og en uge på rapport.

Arbejdsprocess

I dette afsnit fremsættes gruppens arbejdsprocess. Det vil foregå i tre dele, en indføring i gruppens dynamik og håndtering heraf ved brug af Belbins ni teamroller, gruppens brug af Unified Process, i forlængelse af gruppens arbejdstilgang med tilhørende iterationsplan, og gruppens koordinering.

Gruppedynamik

Rollefordeling ved Belbin

Til at sikre et godt projektforsløb, forsøgte gruppen at finde ud af rolle dynamikken i gruppen. Dette blev gjort med udgangspunkt i Belbins grupperoller, hvor en beskrivelse af hvert gruppemedlems funktioner, i gruppen blev fundet frem til. Blikket for hvilke roller gruppen besad, gav indsigt i nogle af de “mangler”, gruppen havde og ligeledes, hvor projektet var godt dækket ind på baggrund af de fire medlemmere. Særligt to uhensigtsmæssige karakteristika var tydelige for alle medlemmerne af gruppen; konfliktskyhed og undvigelse af upopulære valg. Til at imødegå disse svagheder blev nogle gruppemedlemmerne givet roller i håb om at reducere disse karaktertræk.

Den naturlige dynamik i gruppen, var gennem rollerne let at finde, hvorfor man kunne se at visse af gruppemedlemmerne varetog visse opgaver i højere grad end andre. På baggrund af dette indblik, forsøgte gruppen at rammesætte hvert gruppemedlems funktion i gruppen via en rollefordeling.

Dette fungerede ikke optimalt og måtte gennem projektet revurderes. Hermed forsøgte gruppen at finde en balance, som fungerede. Gruppen har en stor dedikation til arbejdet, som også betyder at der er stor ivrighed til at lave noget. Ud fra dette kunne det være svært at udvikle, idet alle havde en mening om fx design og arkitektur.

På trods af forsøg med uddelegering af roller blev det tydeligt gennem projektet at følgende roller gjorde sig gældende;

- Rasmus - specialist og idemand, hermed den som løste de svære problemstillinger i projektet, når andre sad fast og en force, hvad angår at komme på nye ideer. Desuden undersøgte Rasmus nye koncepter, som skulle gøres brug af i projektet.
- August - multirolle og idemand, hermed trådte August til i de situationer, hvor behov var og stod som primær udvikler for brugeroplevelsen. Desuden har August forsøgt at få velvære i gruppen til at fungere ved at tjekke ind med de andre gruppemedlemmer.
- Kamille - organisator, hvormed Kamille har forsøgt at holde tidsplaner og sikre gruppen fokusere på at arbejde. Hun sætter nemt handling bag ting som skal udføres.
- Michala - Trådte hurtig til i en lederrolle, som sørgede for et generelt overblik og en plan for projektet. Hun uddelegerede opgaver efter behov af de mål, som der bliver

sat og havde en meget analytisk tilgang til problemløsning. Desuden havde Michala en meget detaljeorienteret sans i forhold til afslutning af projekt.

Arbejdstilgang

Unified Process

Gruppen har arbejdet ud fra Unified Process, som indeholder fire faser; forberedelse, etablering, konstruktion og overdragelse. Når man arbejder med unified process, arbejder man i iterationer. Normalt vil et projektforsløb bestå af flere iterationer og nå igennem alle fire faser.

En kort opsummering af faserne lyder således: Forberedelsesfasen er en kort fase, hvor man bruger tid på at få et overblik over projektets krav og rammer, samt kundens behov. Etableringsfasen, som er den længste fase, er hvor skelettet og den grundlæggende arkitektur der skal skabe fundament for programmet bliver skabt. Det der er startet i forberedelsesfasen bliver i etableringsfasen raffineret. Konstruktionsfasen bruges på at udfylde det skelet der blev udviklet i etableringsfasen og finpudse systemet, så det er klar til overdragelse. Overdragelsesfasen er der hvor produktet bliver udgivet.

Fra start var det ikke forventet at projektet ville nå konstruktion og overdragelsesfasen, herunder fremsættes iterationsplanen for projektet.

Iterationsplan

Vedlagt i bilag 2, ses gruppens iterationsplan udarbejdet over projektet med udgangspunkt i Larman(2015). En iterationsplan er en oversigt over de iterationer projektet gennemgår. Ved udvikling af iterationsplanen, er den fremført således, at der udarbejdes én iteration fremad af gangen, hvor der i projektstart blev lavet en plan for første iteration. Først da denne iteration påbegyndes planlægges næste iteration. Sådan følger udarbejdelse af resten af iterationsplanen(Larman, 2015, s. 32).

I arbejdet med iterationer handlede det om i planlægningen af en iteration at sætte start og slutdato på denne. Dette er hvorfor en iteration aldrig forlænges, men afsluttes, som planlagt. Ting man ikke når i iterationen må derfor videreføres, således sikre man at få udviklet noget.

Gruppens brug af unified process fungere ikke som en normalt process ville, idet iterationerne ville varer uger og alle faser ville nås. I dette projekt har gruppen forsøgt at arbejde inden for rammerne af unified process men med iterationer på nogle dage og maks op til en uge.

Efter påbegyndelse af fastlæggelsen af projektets rammer og krav, samt udarbejdelse af forberedende analyser i forberedelses fasens ene iteration, bevægede gruppen sig ind i første iteration i etableringsfasen, som i alt indeholdte tre iterationer. Her forsøgte gruppen at udarbejde det bredere skelet, af henholdsvis registrering af projekt use casen og at gøre det muligt for brugeren at logge ind. Inden kodning blev det udarbejdet i forberedelsesfasen raffineret og supplerende diagrammer udført, som domænemodel og start klassesdiagram.

I anden iteration, i etableringsfasen blev gruppen opmærksom på behov for omstrukturering af systemt, hvorfor der i denne iteration blev lavet større ændringer i kodestrukturen. Herudover blev der tilføjet use case om at kunne vedligeholde projekter samt validering og testning af systemet.

Sidste iteration i etableringsfasen er mindre klar, idet meget finpudsning af programmet fandt sted her. Denne iteration var meget præget af at udvikle på et eksamensprojekt, hvorfor rapportskrivning blev påbegyndt. Herudover fandt, sikring af korrekt krav forståelse sted og den visuelle styling af brugergrænsefladen, var særligt i fokus, således systemet kunne brugertestes af bekendt. På baggrund af udført brugertest fandt mere finpudsning sted(Bilag 3). Dette kunne argumenteres at være en del af konstruktionsfasen og ikke etableringsfasen, hvorfor at lade bruger testen ske på et mere ufærdigt produkt ville have fundet sted.

Allersidst i iterationsplanen(bilag 2) fremgår at slutiterationen ikke hørte under nogen af up faserne, idet denne omhandlede rapportskrivning og færdiggørelse til indlevering af eksamensprojekt.

Brug af kundeproxy

Da de interessenter med interesse i projektet ikke faktisk har nogen måde at kommentere på projektet, er der i stedet søgt til projektets vejleder, som proxy. Hermed har det ikke været muligt at gå til slutkunden Alpha Solutions og lave accepttest af systemet. Hvorfor vejleder er afsøgt til kravsafklaring mens en brugeroplevelses test er foretaget, i håb om indblik i brugerens oplevelse af brugerfladen dog ikke af en vejleder.

På baggrund af manglende input på projektets tilstand, blev der lavet en UX think aloud-test. Dette resulterede i at dele af webapplikationen skulle redesignes, for at gøre det mere intuitivt at brug og gav grundlag for yderligere funktionalitet.

Par programmering

Gruppen har i nogle tilfælde gjort brug af par programmering. Dette er gjort ved at to gruppemedlemmer i samarbejde har arbejdet på en use case eller metode, hvor den ene har ageret driver - den der skriver selve koden, og den anden har ageret navigator - den der siger hvad der skal skrives (<https://www.agilealliance.org/glossary/pairing/#q=~13/12-2021>). Dette er blevet brugt som et redskab til dels at have flere øjne på svære dele og dels været med til at styrke gruppens viden og overblik over programmet.

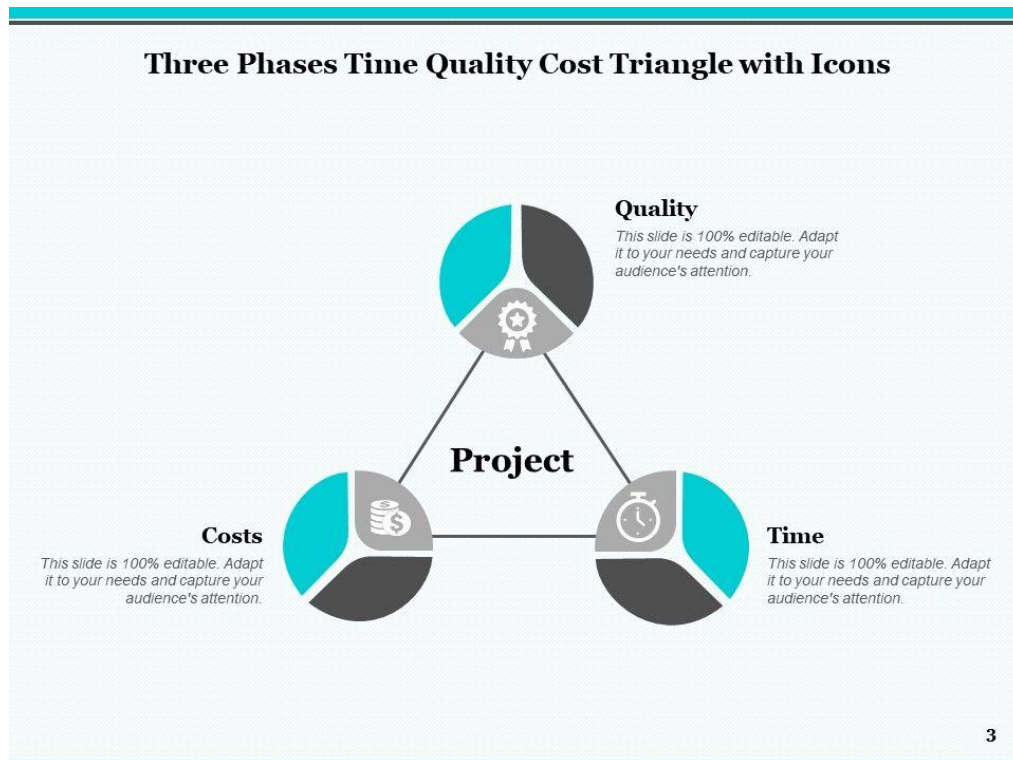
Koordinering

Planlægning

Forskellige redskaber blev gjort brug af til at give projektet et overblik samt at holde planen for gruppearbejdet. En fælles kalender blev oprettet, til at sikre hvem der arbejdede i hvilke tidsrum. Dette blev suppleret af en opgaveliste som løbende blev opdateret, med hvad gruppen skulle nå fra dag til dag. Her blev skrevet, hvem som var ansvarlig for opgaverne så man altid var sikker på, hvad folk lavede. Det blev gjort med henblik på at skabe et overblik over, hvad som skulle laves og hvilke opgaver andre allerede var igang med.

På baggrund af rollefordeling var nogen i gruppen i højere grad ansvarlig for denne del af gruppearbejdet, hvor de holdt overblik over planerne samt arbejdsfordeling. Gennem projektet blev det tydeligt at gruppen havde tendens til at overføre punkter fra dag til dag. Altså havde alle dage hængepartier, mængden blev dog ikke forøget som dagene skred frem men heller ikke korrigeret.

Gennem projektet oplevede gruppen at deres ressourcer blev formindsket, idet to gruppemedlemmer blev syge, heraf den ene med corona. Hertil måtte gruppen restrukturere både planer og forventninger til projektet, idet færre personer måtte bære arbejdsbyrden. Nedenfor ses en model over projektledelses begrænsninger (<https://www.slideteam.net/time-quality-cost-cost-resources-time-schedule-quality-scope.html> 14/12-21).



Med modellen vist ovenfor vises de tre begrænsninger involveret i projektledelse; tid, kvalitet og ressourcer. Denne er særligt kommet til udtryk i dette projekt, grundet sygdoms omstændighederne nævnt ovenover. Idet gruppen ikke kunne ændre på tidsrammen sat på fire uger, og ej heller på ressourcer involveret i form af gruppemedlemmer, måtte systemets kvalitet sænkes. Det kommer til udtryk i projektet ved fravælgelse af yderligere features ud over den vurderede kerneopgave. Med henblik på risikoanalysen foretaget under virksomhedsanalyse delen var forventning ligeledes at dette måtte skabe en del implikationer for gruppen. Dog har gruppemedlemmer grundet pres over projekt forsøgt at arbejde på trods af sygdom, men det har været tydeligt med nedsatte evner og energi.

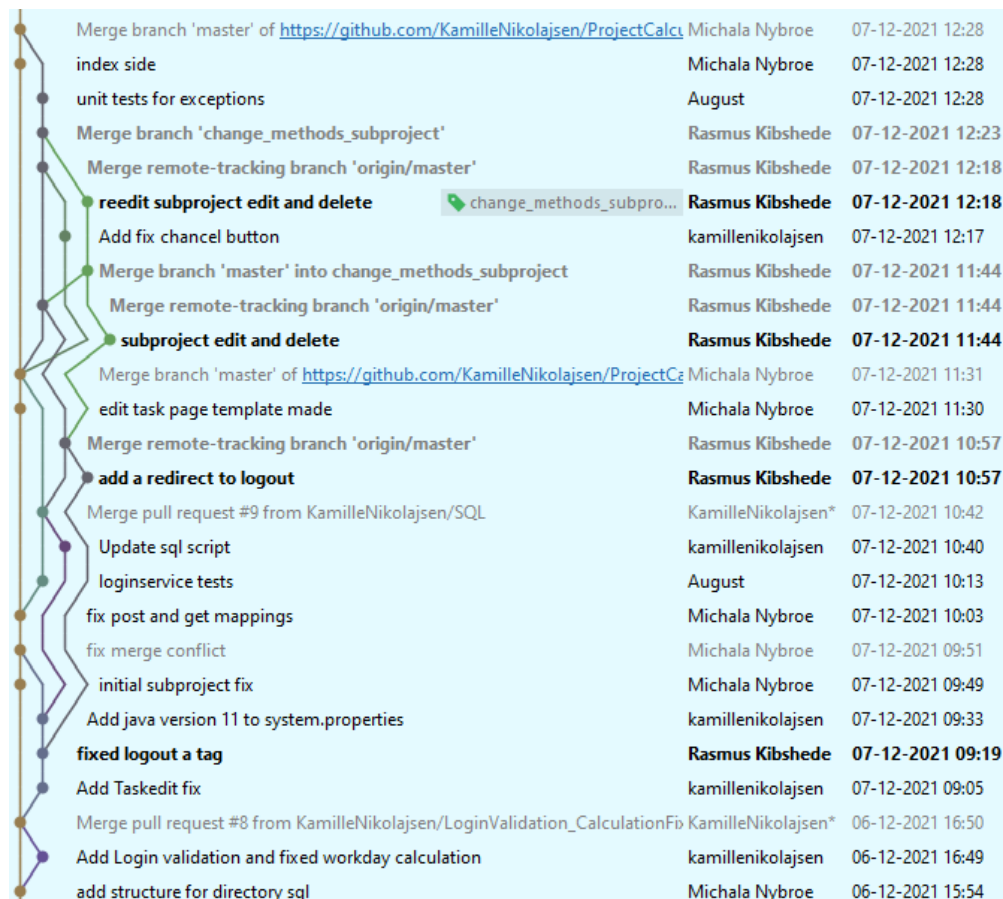
Endvidere har sygdom resulteret i at dette projekt i stor grad er lavet online, som har haft effekten at kommunikation i gruppen er blevet reduceret, og generel trivsel og social engagement har været lavere end typisk for gruppen.

Github

Igennem projektet har gruppen gjort brug af Centralized workflow, hvor alle gruppemedlemmer har haft adgang til et fælles repository. Det fælles repository har alle medlemmer en lokal klonet udgave af, som de arbejder på. Her foretages lokale ændringer og når disse ønskes overført til den fælles repository, tilføjes ændringerne igennem git-kommandoer, hvorefter disse

ændringer “commites” og herefter sendes til det originale repository på github([Distributed-Workflows](#) 30/11-21). Således kan alle i gruppen arbejde sammen om det repository som er på git og ved at “pulle”, kan de ligeledes videreudvikle på den nyeste version uden at nogens arbejde overskrives. Desuden gør sigende commit beskeder det nemt at lokalisere fejl som måtte opstå.

I brugen af github har hvert gruppemedlem endvidere gjort brug af branches til udarbejdning af indhold til systemet. At arbejde i branches er en god måde at separere gruppemedlemmers individuelle arbejde, uden at eventuelle fejl og konflikter påvirker resten af projektet. Desuden hjælper det at arbejde i branches med at reducere antallet af merge conflicts, idet flere kan udvikle i samme klasser og committe disse uden, at det har indvirkning på de andres arbejde. Brugen af branches har derudover været en måde for gruppen at udvikle forskellige features, parallelt med hinanden. Dette har gjort at der frit kan committe til branches uden, frygt for at et gruppemedlems arbejde ville få hele programmet til at gå ned.



Analyse og design

I denne del vises gruppens analyse og design af arkitektur for systemet.

Brugerfladedesign

I starten af projektet var et af de valg gruppen foretog sig, at lade sig inspirere og tilmed låne dele af Alpha Solutions egen hjemmeside. Her blev hentet Alpha Solutions Header og Footer til brug på systemets sider. Hermed har gruppen set Alpha Solutions html og css igennem til formålet af at få systemets look tilpasset deres. Den er ikke fuldkommen magen til, men er i de store træk deres header og footer (<https://www.alpha-solutions.com/da> 1/12-21). Herudover blev systemet udviklet med udgangspunkt i Alpha Solutions egen farvepalette på deres hjemmeside.

Prototype

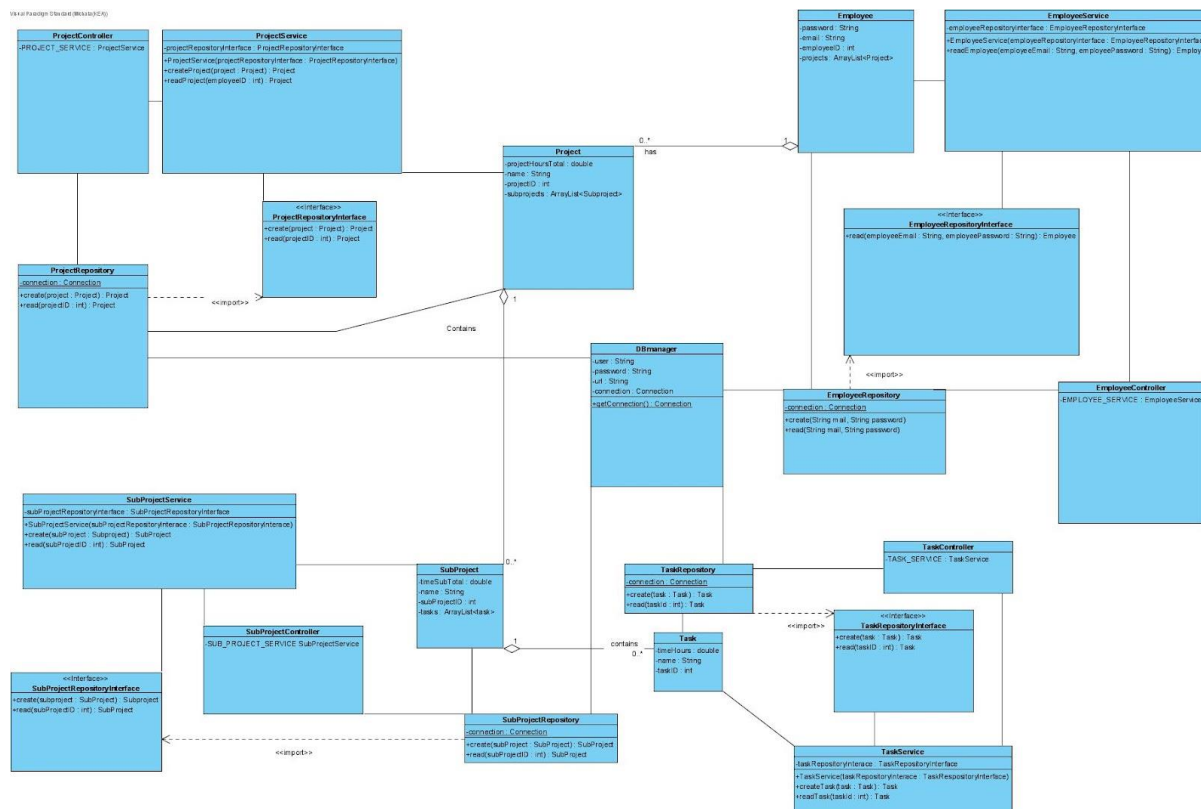
Gruppen undladte ubevidst i starten af processen at gøre brug af prototyper. Dette gav komplikationer. Uden fastlagt vision for det visuelle udtryk for brugerfladen, skabte det problemer i form af lange diskussioner, misforståelse og rettelser af flere omgange. Gruppen kunne derfor konstatere, at det havde været meget praktisk med en prototype fra start, så alle gruppemedlemmer kunne blive nogenlunde enige om det visuelle udtryk på forhånd og føle deres holdning til udseendet blev medtaget.

Diagrammer

De lavede diagrammer, er dem som gruppen mener der er behov for, for at en ny person kunne sætte sig ind i programmet blot ved at studere dem nærmere.

Start klassediagram

Det første udkast af systemets klassediagram vises til oversigt over systemets udvikling. Der er blevet forsøgt at lave et system med en opdelt og modul orienteret arkitektur ud fra Martin fowler(<https://martinfowler.com/bliki/PresentationDomainDataLayering.html> 15-12 -2021). Lagdelingen af systemet ses ved at skille præsenteringslaget, domænelaget og datalaget fra hinanden. Dette opdeles endnu engang på lodrette niveau, således hvert domæne har tilhørende klasser og bliver full stack(<https://martinfowler.com/bliki/PresentationDomainDataLayering.html> 15-12 -2021).



I ovenstående ses udarbejdelsen af første udgave af projektets klassediagram. Denne viser den forventede opdeling af programmet i fire domæner, hvorfor hvert domæne havde controller, service, repository interface og en repository klasse. Dette blev udført med henblik på overblik over projektets design forinden kodning. Hermed blev det nemmere at arbejde parallelt som gruppe, idet der allerede var en overensstemmelse imellem de enkelte gruppemedlemmer omkring systemets struktur.

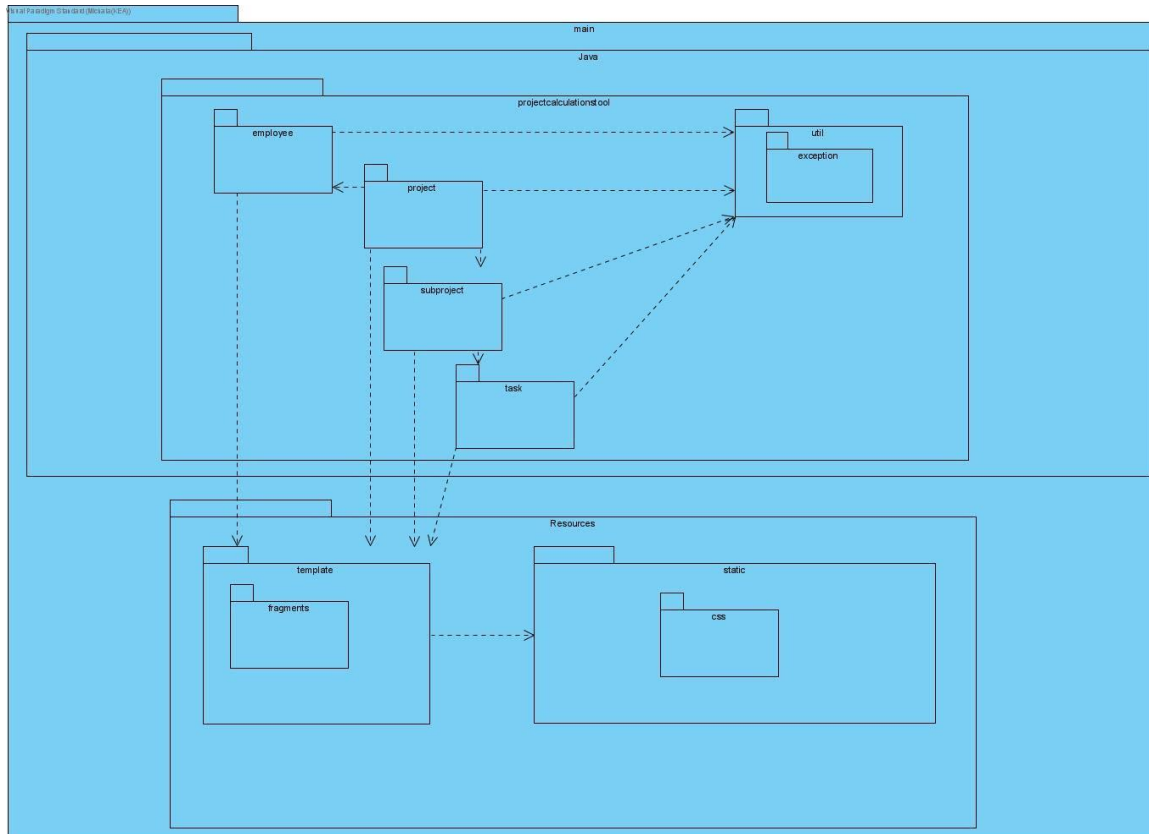
System oversigt

Her vises systemet udviklet, dette forsøges at blive gjort ved en indføring i det bredere perspektiv af systemet. Til at vise strukturen bag vises først et pakkediagram, hvor meningen er at møde opdelingen i programmet. Herefter ses et state machine diagram til fremvisning af brugerfladens funktionalitet med andre ord brugeroplevelsen.

Herefter fremvises de diagrammer lavet over systemets mere detaljerede funktioner og struktur. Det gøres ved fremvisning af et klassediagram, der afbilder slutprogrammet, efterfulgt af sekvensdiagram over at læse et projekt. Til slut vises entity relationship diagram af databasen.

Pakkediagram

Nedenstående diagram giver et overblik over pakkerne i programmet, samt relationen mellem. Dette er illustreret med pile der pejer i retningen af den kendte relation.



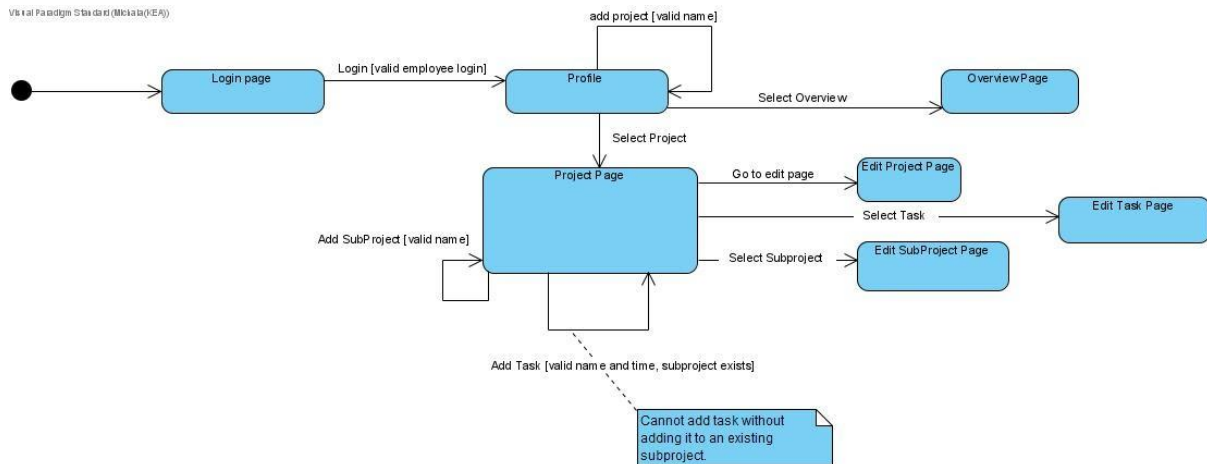
Pakkediagram modellen skal fremvise systemet, hvor opdelingen i domæner kan ses. Som billedet viser er forsøgt at lave et domain driven design arkitektur således de forskellige områder i systemet afgrænses fra hinanden og holdes sammen i full stack

pakker(<https://martinfowler.com/bliki/PresentationDomainDataLayering.html> 15/12-21). Domæne pakkerne kan så interagere med hinanden. Det ses heraf at 'employee' mappen interagerer med projektmappen, som interagerer med delprojektmappen, der interagerer med opgavemappen. Alle domænerne interagerer desuden med henholdsvis util mappen og template mappen. Template mappen interagerer med static mappen, hvori css mappen befinder sig.

Det er ikke muligt fra modellen at se, men i util mappen befinder DBManager klassen sig.

State Machine Diagram

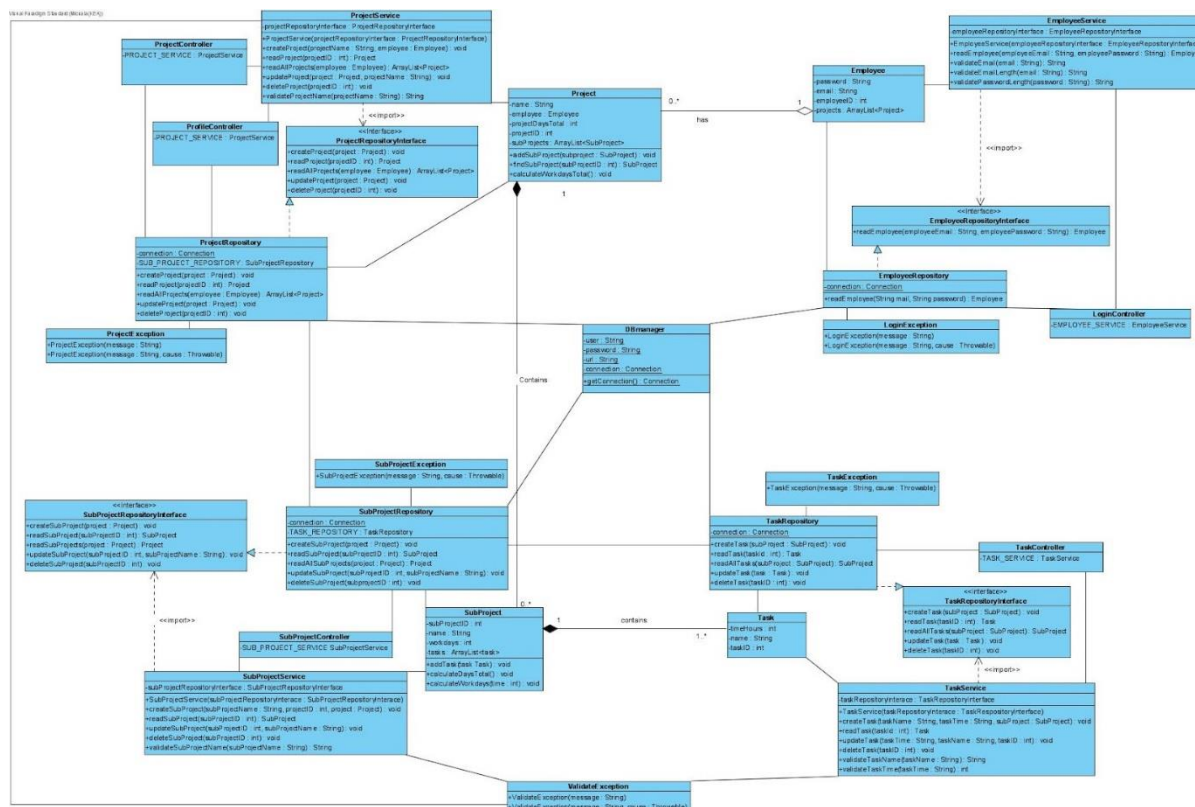
Gruppen valgte at gøre brug af et state machine diagram til overblik over, hvordan man navigerer mellem de forskellige websider, samt hvad der er af funktioner på siderne.



Ovenfor ses State Machine diagrammet, som viser, hvordan brugeren kan benytte sig af programmet. Projektsiden bliver samlingspunkt for vedligeholdelse af projektet, mens den ansattes profilside viser et overblik over de projekter som er i gang. Hertil skal knyttes at oprettelse af en opgave er afhængigt af allerede oprettet delprojekt, således tilføjer den ansatte delprojektet til projektet og opgaver til delprojekterne.

Slut klassediagram

Det færdige resultat af klassediagrammet, giver et overblik over relationen mellem de forskellige klasser, samt indholdet af relevante attributer og metoder. Her ses hvordan systemets forskellige klasser arbejder, både internt, men også samarbejdet klasserne imellem.



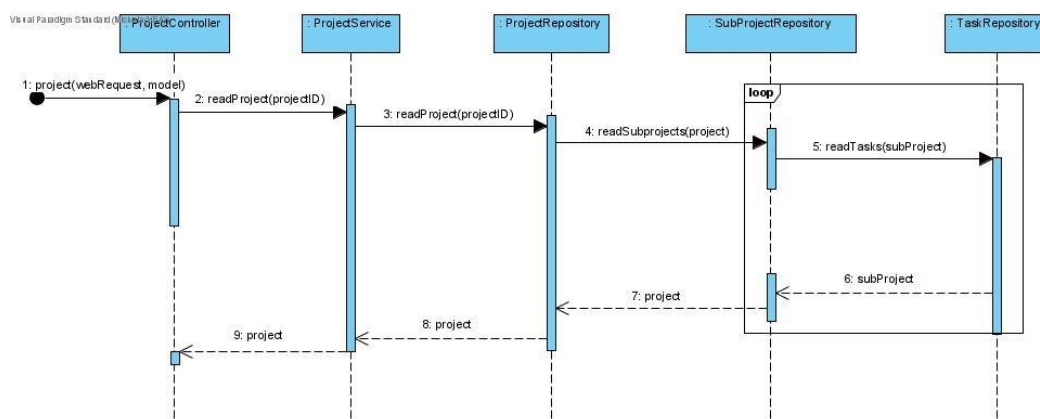
Diagrammet viser, hvordan programmet er lavet således der overholdes en lagdelt arkitektur, men denne er desuden opdelt således at hvert domæne bliver full stack (<https://martinfowler.com/bliki/PresentationDomainDataLayering.html> 15-12 -2021). Herved indebærer alle domænerne præsentationslag, domæne lag og datalag. Det er derfor i den givne domæne at alt, som har med den at gøre foregår, hvorfor interagering uddelegeres til de klasser som er ansvarlige for et givent domæne.

Controllerne tager imod data fra brugerfladen og fører det videre til servicelaget som validere inputtet. Servicelaget håndtere så enten data'en ved at sende en fejlbesked tilbage til controller eller videregiver dette til data laget, repository, som herfra gennem en "connection" laver kald til databasen og håndtere dataen.

Det er i domæne klasserne at beregning af, hvor lang tid de forskellige dele af projektet vil tage, sker idet de er informations eksperterne om dem selv. Til videre uddybelse af hvordan de forskellige lag i hvert domæne spiller sammen vises et sekvensdiagram over kaldet til at vise et projekt.

Sekvensdiagram

Gruppen valgte at gøre brug af et sekvensdiagram i forbindelse med handlingen der sker, når et projekt skal læses. Dette blev gjort for at illustrerer sekvensen af handlinger der sker i processen, under læsningen fra controlleren til repository laget, og tilbage igen.

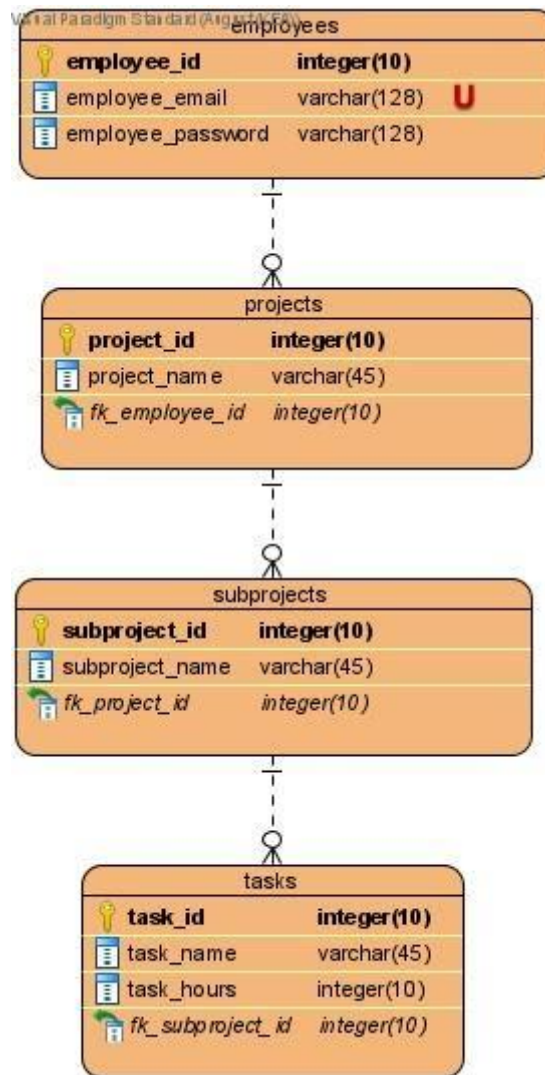


Som diagrammet illustrerer kaldes metoden “project” i controller klassen for projekt. Herfra laver controlleren et kald til projekt serviceklassen, der videre kalder projekt repository klassen. Det er så projekt repository klassen som kalder delprojekt repository klassen og det er delprojekt repository klassen der gentagne gange kalder til opgave repository klassen for hvert delprojekte det indeholder. Opgave repository returnere så det delprojekt, den har modtaget og overfører et helt delprojekt med opgaver i til delprojekt repository. Delprojekt repository returnerer ligeledes et projekt til projekt repository, der så sendes gennem alle lagene tilbage og viser brugeren et givent projekt.

Sekvensdiagrammet viser med ovenstående, hvordan domænerne er full stacked fra controlleren, som er første lag efter user interface helt ned til datalaget som indhenter data fra databasen(<https://martinfowler.com/bliki/PresentationDomainDataLayering.html> 15-12 -2021).

Entity Relationship Diagram

Til at danne et overblik over databasen valgte gruppen at gøre brug af et entity relationship diagram, som nedenstående. Dette illustrerer, hvordan de forskellige tabeller arbejder sammen, samt hvad de indeholder.



I databasen er fire tabeller, som er tilsvarende domænerne fremsat i de øvrige diagrammer. Alle domænerne har et id, så de er til at få fat på nede i databasen. Den ansatte har herudover to varchars; en email og et password, med en længde på 128 karaktere. Den ansattes email skal være unik, således det ikke senere er muligt at oprette brugere på samme email.

De tre øvrige domæner ligner hinanden meget, da de består af et navn og en foreign key. Den foreign key gør at en ansat kan have projekter, et projekt kan have delprojekter og at delprojekter kan have opgaver, ved at knytte keyen til et id. Hermed knyttes tabellerne nedefra og op, altså kan man fx ikke have et projekt uden at den er knyttet til en ansat.

Herudover har opgaver en int som er tid angivet i timer, denne og alle de andre columns kan ikke være tomme.

Patterns

Herunder fremsættes gruppens brug af patterns. Definitionen af et mønster(pattern) er principper og idiomer der kodificeres i struktureret format, som beskriver et problem og dets løsning, og som er navngivet (Larman, 2015, s 278). Der er her blevet gjort brug af forskellige mønstre til at skabe et solidt og lev vedligeholdende system. Basalt set forsøges at holde god kodelstil gennem at følge strukturanbefalinger i mønstrene.

MVC

MVC er et designmønster som står for 'Model-View-Controller', der består af brugerfladen(view), domæne objekter(Model) og applikations logikken(Controller) (Larman, 2015, 209). Det sørger for domæne objekter ikke har direkte adgang til brugerfladen. Dette overholdes i systemet ved at have controller klasser mellem domænelag og brugerfladen så de aldrig har direkte kontakt.

GRASP

GRASP består af 9 principper, der fungerer som et læringsredskab til at designe programmer i overensstemmelse med objekt orienteret design, med henblik på ansvar (Larman, 2015, s. 277). De 9 principper i GRASP er; creator, information expert, low coupling, controller, high cohesion, polymorphism, pure fabrication, indirection og protected variations.

Creator

Det er i systemet serviceklassen og repository klassens job at kreere modellerne. service tager sig af at kreere, når brugerinput modtages, mens repository gør det, når det modtages fra databasen. Desuden indeholder projekt klassen en arrayliste af delprojekter og delprojekt modellen indeholder ligeledes en arrayliste af opgaver.

Oprettelse af service klasserne foregår i spring controller klasserne. Service klassernes konstruktør forventer at modtage et objekt af repository interfasen. I spring controlleren sendes derfor et repository med, hvor repository klassen implementere repository interfasen. Hermed foregår en dependency injection.

Service klasserne bruger validerings exceptions, hvor både service- og repository klasserne bruger login, projekt, delprojekt og opgave exceptions. Disse exceptions uddybes senere i rapporten.

Information expert

Springcontrollerens ansvar er at skabe forbindelse mellem brugerfladen og programmet. Den tager fat i ting fra url og http og sender det videre i programmet, ved brug af webrequest- og model klassen. Model klasserne i domæne laget står for viden om dem selv, herunder kalkulerung af arbejdsdage. Service klasserne har ansvar for validering af brugerinput. Repository klasserne har til ansvar at klarlægge sql queries til databasen. Hertil er DBManagers job at skabe en connection via en jdbc driver til databasen.

Low coupling

Gennem programmet er forsøgt at minimere klassernes afhængighed af hinanden i, således et solidt system kan udvikles minimerende af problematikker ved ændringer af program. Det er blandt andet blevet gjort ved at opdele programmet i domæner, hvor hvert domæne er informations ekspert for dette. Det ses, når projektrepository lader delprojekt repository stå for CRUD(create, read, update og delete) af delprojekter. Et eksempel på repository kommunikation kan ses i sekvensdiagrammet.

Controller

I modsætning til i MVC er controller her service klasserne, hvilket er det første efter præsentationslaget (Larman, 2015, 308). Hermed er controller bindeled mellem præsentationslaget og logikken i domæne laget.

High cohesion

Systemet er opbygget således klasser ikke bør have for meget ansvar. Hvilket er med til at holde klasserne fokuseret og hermed lettere at vedligeholde også over tid. High cohesion opnås meget i forlængelse af det angivne i afsnittet om low coupling.

Indirection

Gennem systemet er flere klasser imellem fx brugerfladen og databasen, således sikres ingen direkte kommunikation mellem bruger og logik. Særligt er direkte tilknytning mellem repository og service fjernet ved i stedet at give service klasserne adgang til repository interfacen.

Protected variants

Ved at håndtere fejl, der måtte opstå i brugen af programmet i henholdsvis service og repository klasserne beskyttes systemet og gøres stabil.

Singleton

Ved at gøre brug af Singleton pattern sikrer man, der kun bliver oprettet et enkelt instans af en klasse, og at der kun er ét adgangspunkt. Dette er med til at sikre high cohesion og low coupling, ved at definere attributten i klassen og gøre metoden 'getConnection' statisk. Dette gør det muligt i andre klasser at kalde metoden uden at skulle have en instans af klassen og reducere hermed klassernes kendskab til hinanden (Larman, 2015, s. 442-446).

Implementering

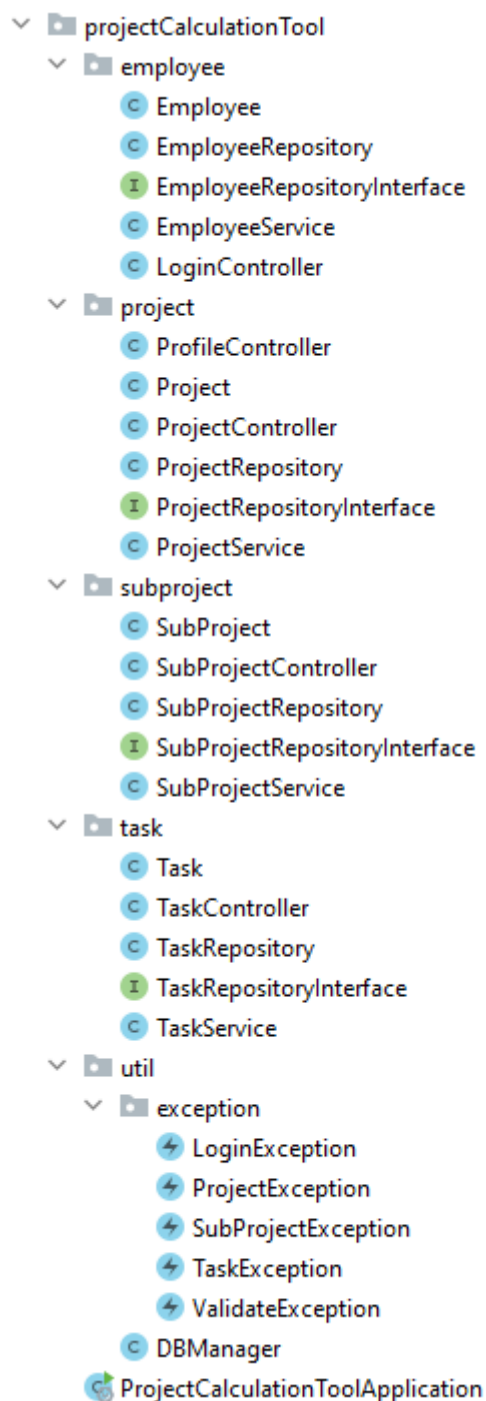
Til brug af applikationen skal der benyttes udvikler software, som kan compile Java og som understøtter; Spring Boot, Thymeleaf, JDBC, MySQL og Maven. Desuden skal man have en database manager, som understøtter MySQL. Gruppen har gjort brug af IntelliJ IDEA og MySQL workbench.

I følgende vil fremvisning af projektets kode blive fremført. Dette opdeles i programstruktur, særlige forhold, web og database.

Programstruktur

Strukturen i systemet er konstrueret ud fra Martin Fowler linket (<https://martinfowler.com/bliki/PresentationDomainDataLayering.html>, 15-12 -21) om lagdelt arkitektur. Her opdeles de forskellige lag i systemet, i forskellige klasser. Således bliver præsentationslaget udover html og css til spring controller klasser. Domænelaget bliver service og model klasser og er hvor alt forretningslogikken ligger. Datalaget bliver repository klasser, der via en specifik klasse, DBManager, er forbundet til databasen. Desuden er strukturen opbrudt vertikalt, så hvert domæne orienteret modul bliver full stack. Altså indeholder hvert modul alle lagene i sig.

Til supplement af tidligere fremviste pakke diagram, er nedenfor, et udsnit af systemets mappestruktur.



Overholdelse af domæne struktur

Til overholdelse af opdelingen med henblik på Martin Fowler

(<https://martinfowler.com/bliki/PresentationDomainDataLayering.html>, 15-12 -21), foregik læsning af projekt ved kommunikation mellem de tre repositories for projektdele. Dette kan ses visuelt som fremvist i sekvensdiagram under analyse og design afsnittet.

Kommunikationen mellem de 3 repositories starter i projekt repository, der henter grundværdierne, til et projekt objekt fra databasen, som er navn og id. Inden projekt objekt bliver videregivet op gennem systemet, bliver der tilknyttet et eller flere delprojekter til projekt objektets arrayliste. Dette gøres ved at have en forbindelse til delprojekt repository klassen, som vises nedenfor.

```
public Project readProject(int projectID) throws ProjectException {
    try {
        PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM projects WHERE project_id = ?");
        preparedStatement.setInt(1, projectID);

        ResultSet resultSet = preparedStatement.executeQuery();

        Project project = new Project();

        if (resultSet.next()) {
            int id = resultSet.getInt(1);
            String name = resultSet.getString(2);

            project.setName(name);
            project.setProjectID(id);

            project = SUBPROJECT_REPOSITORY.readAllSubProjects(project);
            project.calculateWorkdaysDaysTotal();
            return project;
        } else {
            throw new ProjectException("Invalid Project");
        }
    } catch (SQLException err) {
        throw new ProjectException("Failed read project", err);
    } catch (SubProjectException err) {
        throw new ProjectException("Failed read subproject", err);
    }
}
```

Projekt repository kalder metoden 'readAllSubProjects' på delprojekt repository. I delprojektets repository hentes et delprojekt's grundværdier fra databasen. Disse værdier bliver herefter vedlagt i et nyt delprojekt objekt, som har tilknytning til det pågældende projekt objekt. Dog inden at projektet sendes retur til projekt repositoryet, bliver der tilknyttet fra 0 og op til flere opgaver objekter til et delprojekt objektet. Dette gøres ved at have en forbindelse til opgave repository klassen, der bruger metoden 'readAllTasks'.

```
@Override
public Project readAllSubProjects(Project project) throws SubProjectException {
    try {
        PreparedStatement preparedStatement = connection.prepareStatement( sql: "SELECT * FROM subprojects WHERE fk_project_id = ?");

        preparedStatement.setInt( parameterIndex: 1, project.getProjectID());

        ResultSet resultSet = preparedStatement.executeQuery();

        while (resultSet.next()) {
            SubProject subProject = new SubProject(resultSet.getString( columnLabel: "subproject_name"));

            subProject.setSubProjectID(resultSet.getInt( columnLabel: "subproject_id"));

            subProject = TASK_REPOSITORY.readAllTasks(subProject);

            project.addSubproject(subProject);
        }

        return project;
    } catch (SQLException err) {
        throw new SubProjectException("Read subproject failed", err);
    } catch (TaskException err) {
        throw new SubProjectException("Read tasks failed", err);
    }
}
```

I opgave repository hentes der grundværdier, som i de andre to repositories. Dette ses nedenfor, hvor opgave repository henter tid, navn og id. Disse værdier sættes til et opgave objekt, der tilknyttes til et delprojekt. Dette returneres derefter til delprojekt repository, som derfra returnere et projekt til projekt repository. Herfra sendes projektet igennem systemet ud til brugerfladen.

```
@Override
public SubProject readAllTasks(SubProject subProject) throws TaskException {
    try {
        PreparedStatement preparedStatement = connection.prepareStatement( sql: "SELECT * FROM tasks WHERE fk_subproject_id = ?");
        preparedStatement.setInt( parameterIndex: 1, subProject.getSubProjectID());

        ResultSet resultSet = preparedStatement.executeQuery();

        while (resultSet.next()) {
            Task task = new Task(resultSet.getInt( columnLabel: "task_hours"), resultSet.getString( columnLabel: "task_name"));
            task.setTaskID(resultSet.getInt( columnLabel: "task_id"));
            subProject.addTask(task);
        }

        return subProject;
    } catch (SQLException err) {
        throw new TaskException("Failed reading Tasks", err);
    }
}
```

Forbindelsen til databasen

I projektet bliver JDBC brugt til at forbinde til en online MYSQL database. Forbindelsen bliver håndteret i en singleton klasse “DBManager”, som bruger javas “System.getenv” fra java.lang.System, til at få forbindelses informationer fra webhosten Heroku.

```
public class DBManager {  
    private static String user;  
    private static String password;  
    private static String url;  
    private static Connection connection = null;  
  
    public static Connection getConnection() {  
        if (connection != null) return connection;  
  
        user = System.getenv( name: "user");  
        password = System.getenv( name: "password");  
        url = System.getenv( name: "url");  
  
        try {  
            connection = DriverManager.getConnection(url, user, password);  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
  
        return connection;  
    }  
}
```

Når informationerne, som vist ovenfor, er angivet i 3 string variabler user, password og url, bruges variablerne til at lave en forbindelse ved hjælp af java.sql.DriverManager, som ligger forbindelsen ind i en JDBC driver. Herfra lægges forbindelsen i en attribut af klassen Connection, som er en del af java.sql.

Internt i systemet er det repository klasserne, der kalder DBManager. Idet DBManager klassen følger singleton mønsteret, kaldes direkte på den statiske metode og repository klasserne behøves derfor ikke at have en instans af klassen til benyttelse af metoden.

Nedenfor ses webhosten, herokus config vars, hvor informationer om adgang til databasen kan indføres. Det er denne, gruppen har udfyldt, og herfra klassen DBManager henter indholdet til de tre strenge.

Config Vars



password	[Redacted]
url	jdbc:mysql://hostname:3306/projectcalculationtool?reconnect=true&autoReconnect=true
user	[Redacted]

Her fremsættes et eksempel på, hvad som skulle indtastes, ved url;
jdbc:mysql://**hostname**:3306/projectcalculationtool?reconnect=true&autoReconnect=true, hvor
hostname skal være eget hostname.

Særlige forhold

Validering

Ved udarbejdelse af systemet var der fokus på robusthed, hvorfor validering af brugerinput virkede eminent. Vigtigheden af validering af brugerinput skyldes, at systemet kan bryde ned, hvis disse fejl ikke håndteres i programmet.

```
public String validateTaskName(String taskName) throws ValidateException {  
    if (taskName != null && !taskName.isEmpty() && taskName.length() <= 45) {  
        return taskName;  
    } else {  
        throw new ValidateException("Task name cannot be null or longer than 45 characters.");  
    }  
}  
  
public int validateTaskTime(String taskTime) throws ValidateException {  
    int time;  
  
    if (taskTime != null && !taskTime.isEmpty()) {  
        try {  
            time = Integer.parseInt(taskTime);  
            return time;  
        } catch (NumberFormatException err) {  
            throw new ValidateException("Task time has to be a number in hours.");  
        }  
    } else {  
        throw new ValidateException("Task time cannot be empty and has to be a number in hours.");  
    }  
}
```

Ovenstående billede er et udklip fra gruppens system, under TaskService. I udklipet ses det at både TaskName og TaskTime valideres. Således sikres at det er gyldigt input forinden at opgaven

videresendes til repository laget. Desuden er det ved validering af input, at systemet sender exceptions, ved forkert brugerinput. Herved sikres at systemet ikke kan bryde sammen.

```
public String validateEmail(String email) throws ValidateException {  
    email = validateEmailLength(email);  
  
    // regexApproved string Fra https://reqexr.com/3e48o  
    String regexApproved = "^([\\w-\\.]+@([\\w-\\.]+)[\\w-]{2,4})$";  
    if (email.matches(regexApproved)) {  
        return email;  
    } else {  
        throw new ValidateException("Please enter a valid email");  
    }  
}
```

Regex, som gøres brug af på billedet, validere om en string indeholder en gyldig e-mail syntax, taget fra følgende link (<https://regexr.com/3e48o>, 16-12-2021). Regex er en form for regulært udtryk, der består af karaktere som former sig til et søgemønster, hvorfor man kan bruge søgemønsteret til at søge efter bestemte ting i en streng (https://www.w3schools.com/java/java_regex.asp, 16-12-2021).

Exceptions

Java har indbyggede exceptions, som er smarte i forhold til at vide hvad der er galt. Dog til præcisering af fejlbeskeder oprettes egne exceptions. Disse skal bedre kommunikere, hvor i programmet, der opstår fejl.

```
} catch (SQLException err) {  
    throw new TaskException("Creating Task failed", err);  
}
```

Ovenstående udklip fra gruppens system viser, hvordan Javas indbyggede SQLException bliver håndteret, og kastet videre som en af gruppens egne exceptions. Dette gør fejlbeskederne letlæselige og gør det nemt at navigere hen til fejllens oprindelse.

Systemets egne exceptions består af; LoginException, der håndterer brugerens login oplysninger, i tilfælde af forkert email eller password. Navne Exceptions, som håndterer

oprettelsen af henholdsvis projekt, delprojekt og opgaver og `ValidateException`, der håndterer fejl, i forhold til brugerinput. Eksempel på Exception klasse struktur ses nedenfor.

```
public class ValidateException extends Exception{

    public ValidateException(String message) { super(message); }

    public ValidateException(String message, Throwable cause) { super(message, cause); }
}
```

Brugen af klassen `ValidateException` og hvordan denne kastes ved valideringsfejl, ses nedenfor. Fejlen som nedenfor kastes, er i metoden validering af delprojekt navn. Her sikres at brugerinput af navn, ikke er mere end 45 tegn langt, null eller tomt. Hermed sikrer validerings exceptionen at brugeren ikke crasher programmet.

```
public String validateSubProjectName(String subprojectName) throws ValidateException {
    if (subprojectName != null && !subprojectName.isEmpty() && subprojectName.length() <= 45) {
        return subprojectName;
    } else {
        throw new ValidateException("Subproject name cannot be null or longer than 45 characters.");
    }
}
```

Foruden det interne brug af exception fejlbeskederne, sender systemet desuden denne fejlbesked ud til brugeren.

Session og sikkerhed

Webapplikationen gør brug af session, af sikkerhedsmæssige grunde, men også for at videregive et projekt objekt. Objektet bliver tilknyttet sessionen, når brugeren klikker ind på et projekt. Måden Objektet bliver tilknyttet på, er ved at gøre brug af spring klassen `webrequest` hvori metoden `setAttribute` ligger i. Denne skal bruge et name, value og scope ([spring/webrequest/setAttribute](#), 16-12-2021). Et eksempel heraf kan ses herunder.

```
webRequest.setAttribute( name: "project", project, WebRequest.SCOPE_SESSION);
project.setEmployee(employee);
```

Sikkerhedsmæssigt gøres der brug af session, når en ansat angiver et korrekt login. Ved korrekt login sættes ansat i en attribut i sessionen. Herefter tjekkes det, om denne sessions attribute findes på sider, den ansatte ikke skal kunne tilgå foruden at være logget ind. Hvis ikke den ansatte er i sessionen omdirigeres den ansatte til forsiden.

```
Employee employee = (Employee) webRequest.getAttribute( name: "employee", WebRequest.SCOPE_SESSION);
if (employee == null) {
    return "redirect:/";
}
```

Web

Så Heroku kan forbinde til den eksterne database, benyttes variabler der indeholder forbindelsesinformationer til databasen. Disse variabler bliver omtalt som config vars, på Heroku. Yderligere tilføjes en fil til programmet kaldet system.properties, hvori java sættes til 11. udgave, idet Heroku ikke samarbejder med nyere versioner af java.

De otte gyldne regler

Her fremskrives de otte gyldne regler, og hvordan de bliver benyttet igennem webapplikationen (<https://capien.co/shneiderman-eight-golden-rules-interface-design>, 16/12-21).

Ved brug af webapplikationen forsøges der at holde konsistens gennem det. Dette gøres ved farvevalg, ikoner der går igen og ved at siderne, som har samme funktionalitet holdes ens. Desuden er der forsøgt at gøre brugerfladen nem at navigere gennem genveje, ved tilbage knapper og ved at altid kunne komme tilbage igennem header menuen, til profilsiden.

Project name cannot be null or longer than 45 characters.

Bruger modtager informativ feedback gennem fejlbeskeder, som vist ovenfor, således de kan se, hvad de gjorde galt. Dette er med til at sikre et stabilt program, hvilket går mere i dybden med, under exceptions afsnittet.

Subproject Name

Press save when finished

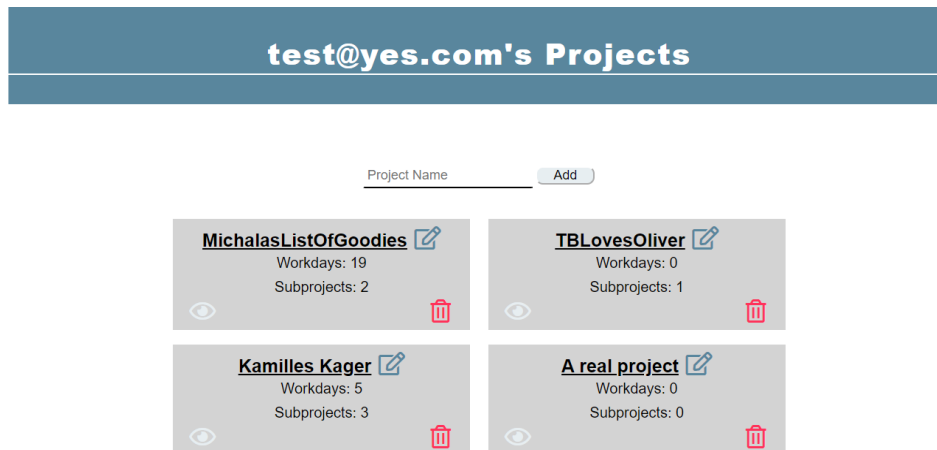
Det ønskes at være nemt at overskue handling gennem cancel knapper, på de forskellige redigeringssider, som vist foroven. For at reducere korttidshukommelsen er der overskrifter på de forskellige sider, som fortæller, hvor den ansatte befinder sig, som vist nedenfor.



Systemet gør ikke brug af dialog ved afslutnings reglen, eller af kontrol og frihed, hvor bruger kan personificere.

Gestaltlovene







Gennem designet af webapplikationen har gruppen haft gestaltlovene i tankerne, for at skabe et solidt udtryk. Det er forsøgt i håb om at holde et mere overskueligt og brugervenligt design(<https://www.nielsgamborg.dk/?p=gestaltlovene> 16/12-21).





Herover ses webapplikationens profilside, hvor gestaltlovene er forsøgt implimenteret (<https://digitypes.dk/hvordan-bruger-man-gestaltlovene-design/>, 16/12-21). Ovenfor ses alle seks gestaltlove. Reglen om lighed gør sig gældende ved at hjernen automatisk fortæller en at de grå kasser er det samme. Reglen om forbundethed gør sig ikke så gældende idet de fire kasser er opdelte, men i forlængelse af reglen om lukkethed, hvorfor de fire kasser ses som indgående i en større ramme. Dette suppleres yderligere at reglen om figur og baggrund, hvorfor hjernen endnu mere ønsker at sætte de fire kasser sammen som en figur. Foruden disse fire regler, gør reglen om nærhed og reglen om symmetri sig gældende. Nærhed ses ved at de fire kasser forbindes, men også i, hvordan de røde skraldespande sammenkobles og samme for de øvrige ikoner. Symmetri reglen kommer mest til udtryk i bunden af hvert projekt kasse. Det er behageligt for øjnene at de to ikoner er overfor hinanden, i modsætning til ikonet, der øverst i kassen står alene.

MichalasListOfGoodies

Workdays: 19

	Katte	Workdays: 19	
	Athena	Hours: 45	
	Nyxos	Hours: 101	

	Design	Workdays: 0	
---	--------	-------------	---

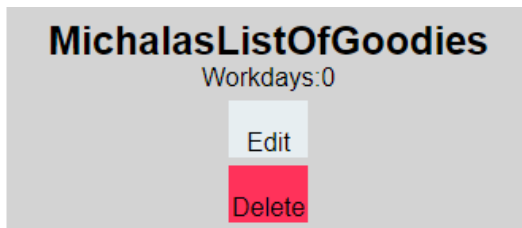
Omkring ovenstående billede kommenteres kun på de regler, som særligt adskiller sig fra profilsiden. Alle seks gestaltlove gør sig gældende for denne side, men her skrives om forbundethed og symmetri.

Forbundethed er tydeligt brugt idet delprojekter og dets opgaver vertikalt hænger sammen. Dette brud mellem et delprojekt og et andet delprojekt hjælper med at signalere at de ikke hører sammen.

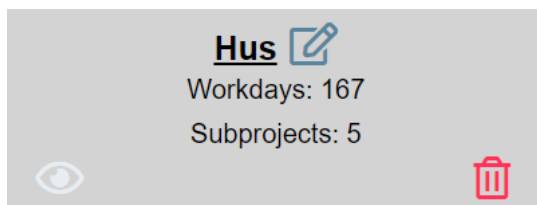
Symmetri bliver brudt af henholdsvis opgaver, som er indrykket og ved at tilføjelse af delprojekt er ovenover delprojekter, mens tilføjelse af opgaver er under. I første tilfælde er det meningsgivende idet hjernen ubevidst herigennem ved at opgaver er forskellig fra delprojekter. Det er dog uhensigtsmæssigt, at der er inkonsistens mellem tilføjelse af delprojekter og opgaver.

Fontawesome

Font awesome er en hjemmeside med open source ikon bibliotek(<https://fontawesome.com/v5.15/icons?d=gallery&p=2> 16-12-2021). Tilvalget om brug af ikoner fra font awesome er blevet gjort på baggrund af den brugeroplevelses test udført (Bilag 3). Nedenfor ses et udklip af, hvordan brugerfladen så ud forinden test.



Inden videre styling var gestalt principperne om lukkethed, nærhed og forbundethed allerede overholdt. Dog resulterede brugertesten i kritik af udseendet og lethed af navigation, hvorfor indførelse af ikoner samt ændring af funktion ved rediger knap blev foretaget. Nedenfor ses hvordan et projekt på profilsiden ser ud efter ændringer udført på baggrund af testerens indput. Ikon brug er blevet anvendt i systemet til at give brugeren et hurtigt og nemt overblik over funktionaliteten på siden.



Som afbilledet ovenfor, gør systemet brug af tre ikoner på profilsiden. Ikonerne er valgt så brugeren intuitivt skal kunne genkende dem og med det samme vide, hvad funktionen af at klikke på ikonet er. Pennen er at redigere, øjet er at se overblik over projekt og skraldespanden er slettefunktion. Rediger og slet går igen hele vejen gennem systemet for konsistensens skyld, hvilket skal optimere brugeroplevelsen og reducere brugerens hukommelse belastning. Som tidligere nævnt forsøges farvevalgene at være i overensstemmelse med Alpha Solutions hjemmeside og er for de tre ikoner hentet derfra. Den røde farve skal signalere noget seriøst og associeres med at slette, hermed yderligere optimere brugeroplevelsen.

Foruden de tre ikoner allerede nævnt, blev yderligere to ikoner benyttet; en nøgle og et profil ikon, på login siden. Dette var som ved de tre andre til optimering af navigering på siden og reducere behov for forforståelse af brugen af applikationen.

```
<script src="https://kit.fontawesome.com/1a9ff76d50.js" crossorigin="anonymous"></script>
```

Brugen af fontawesome foregår ved at linke i html head tag til et script med en reference til et fontawesome kit, som er baseret på Javascript, se ovenstående udklip.

```
<span class="far fa-trash-alt fa-2x"></span>
```

Ovenfor ses hvordan ikonet indsættes i html filen gennem et span tag med class navn tilsvarende dets class på fontawesomes side.

HTML/CSS

HTML står for 'Hyper Text Markup Language' og er således et opmærkningssprog brugt til webudvikling. Hvor det er muligt at lave en webside kun med HTML vil det ikke i længden være hensigtsmæssigt, da styling af websider fremmer brugeroplevelsen. Derfor har man ved hjælp af Cascading Style Sheets(CSS) formateret den pågældende HTML.

Når der arbejdes med både html og css er det vigtigt at den pågældende html side refererer til det korrekte stylesheet. Dette gøres ved at bruge link tagget i html sidens head som set nedenfor.

```
<link rel="stylesheet" href="/CSS/project.css">
```

Required fields

HTML har indbygget en required functions, som kan bruges i input fields. Disse sikrer at bruger skal udfylde felterne med type data ellers vises en fejlbesked. Da dette kan blive slået fra gennem en almen browser såsom google chrome, er det grunden til validering af brugerinput.

ID og Class

Under udvikling af brugerfladens html og css, er der taget brug af class og id attribut typerne, som hjælper til styling af webapplikationer.

ID er en global attribut, der hjælper med at definere og identificere et element, som ikke forekommer flere gange i den pågældende HTML fil(https://developer.mozilla.org/en-US/docs/web/html/global_attributes/id 16-12-2021). ID bruges altså kun på unikke elementer.

```
<form id="addproject" action="/addproject" method="POST">
  <input type="text" placeholder="Project Name" name="projectname" class="textfield" required>
  <input type="submit" value="Add" class="submit">
</form>
```

Ovenstående udklip forekommer i html siden 'profile', hvor der findes en form med ID 'addproject'. Denne form tilføjer et projekt til den pågældende brugers profil og findes kun dette ene sted i programmet.

Class bruges til de elementer der forekommer, mere end en gang, og er lavet til at håndtere flere elementer bundet af samme class i HTML (https://developer.mozilla.org/en-US/docs/web/html/global_attributes/class 16-12-2021).

Brugen af class i denne webapplikation er herunder på de elementer der går igen eller der er behov af flere af de samme elementer. Dette dækker over de steder, der er taget brug af thymeleaf

for each loop.

```
<!-- project box-->
<div class="projectbox" th:each="project: ${projects}">
  <ul>
    <li>
      <a th:href="/project?id=' + ${project.projectID}" th:text="${project.name}"
        class="projectName" th:name="${project.name}"></a>
      <a class="editbtn" th:href="/project?id=' + ${project.projectID}">
        <span class="far fa-edit fa-2x"></span>
      </a>
    </li>
  </ul>
</div>
```

Herover ses hvor der bliver itereret henover brugerens projekter med den class der indeholder den pågældende information. Da der kan være flere projekter er det vigtigt at det er en class og ikke et id.

Wrapper og content divs

Wrapper og content er navne på div ids, som der kan hjælpe personen, der styler indholdet i div'erne. Wrapperen bruges som den yderste div, for at indeholde header og footer. Wrapperen kan derfor bruges, til at skabe luft i yderkanten på websiden og bruges på alle html sider. Det samme princip gør sig gældende for content div'en, men hvorimod wrapperen, er den yderste div, så er content div'en, den som skifter indhold fra side til side.

Det forsikrer om, at når styling gives til disse div'er, så formindskes redundant kode. Nedenfor ses eksempel på de to div'er.

```
<div id="wrapper">
  <th:block th:insert="fragments/general :: header"></th:block>

  <div id="content">
    <th:block th:insert="fragments/general :: footer"></th:block>
  </div>
</div>
```

Class clearfix

HTML filerne gør brug af en attribut klasse, kaldet clearfix. Clearfix er en klasse i CSS, der har nogle CSS egenskaber, som designer det element, som den bliver givet til.

Clearfix bruges når der i stylesheetet bliver brugt float, på eventuelt en eller flere elementer. Float rykker rundt på elementer og efter elementer er flyttet, kan der ske designmæssige problemer, som kan løses ved at give elementets beholder/div en klasse af clearfix, for at stoppe floaten.

Edit.css

Edit.css er en CSS fil, der bliver givet ud til 3 forskellige edit HTML sider til reducere af redundant kode. Dette er muligt, idet de tre html sider, er af samme overordnede design. Den ene af siderne afviger en smule fra de to andre og kan ved ønske om nærmere styling give problemer. På baggrund af deres nuværende udtryk, er én css fil for alle tre html sider dog rigeligt.

Spring Boot

Spring Boot er et java framework, der benyttes til udvikling af webapplikations delen, af systemet. Spring Boot håndterer HTTP (Hyper Text Transfer Protocol) forespørgsler, såsom “get” og “post”, ved brug af annotations. Spring Boot bruger @GetMapping og @PostMapping, som svarer til get og post. @PostMapping og @GetMapping skal bruge en parameter, som angiver, hvornår metoden skal aktiveres. @PostMapping benyttes til at sende data ved create og update, mens alt andet håndteres i @GetMapping. Eksempler på disse to mappings ses nedenfor.

```
@PostMapping("/login")
public String login(WebRequest webRequest) throws LoginException, ValidateException {...}

@GetMapping("/logout")
public String logout(HttpSession session) {...}
```

@GetMapping gør brug af url attributter, hvor @PostMapping ikke gør.

Klassen, der skal gøre brug af spring annotations, skal være angivet som en spring controller, det gøres ved at angive annotationen @Controller over klasse navnet, som vist nedenfor.

```
@Controller
public class LoginController {
```

Metoderne i spring controller klasserne, gør brug af spring klasser, som WebRequest og Model.

```
String projectName = webRequest.getParameter( paramName: "projectname");

Employee employee = (Employee) webRequest.getAttribute( name: "employee", WebRequest.SCOPE_SESSION);
```

WebRequest er en springklasse, der giver adgang til generelle HTTP metadata (docs.spring.io, 15-12-2021). Ovenfor ses et udklip af koden, hvor webrequest benyttes til at hente et projektnavn, en hel ansat attribut, som derefter angiver sessionens omfang. Webrequest benyttes overordnet til at modtage parametre sendt fra HTML eller sætte et objekt i sessionslaget, sidstnævnte ses eksempel på nedenfor.

```
webRequest.setAttribute( name: "project", project, WebRequest.SCOPE_SESSION);
```

I systemet bliver webrequest klassen gjort brug af, dels til at modtage brugerinput og dels til at sende objekter rundt, mellem de forskellige websider.

Springcontrolleren kan ved brug af Model klassen, sende parametre videre til HTML siden. Dette gøres ved at benytte model klassens metode ‘addAttribute’, som ses nedenfor, hvor en værdi tilføjes, som HTML siden modtager ved hjælp af Thymeleaf.

```
model.addAttribute( attributeName: "project", project);  
model.addAttribute( attributeName: "message", webRequest.getParameter( paramName: "message"));
```

I spring controller klasserne benyttes desuden @ExceptionHandler til at håndtere fejlbeskeder. Nedenfor vises et udsnit fra LoginController klassens @ExceptionHandler metode.

```
//Reference https://stackoverflow.com/questions/804581/spring-mvc-controller-redirect-to-previous-page  
@ExceptionHandler({LoginException.class, ValidateException.class})  
public String handleLoginException(Model model, Exception exception, WebRequest webRequest) {  
    String referer = webRequest.getHeader( headerName: "Referer");  
    model.addAttribute( attributeName: "message", exception.getMessage());  
    return "redirect:" + referer;  
}
```

@ExceptionHandler metoden håndtere her exceptions, af typen login og validering, hvor den ved brug af webrequest klassen, kan kalde metoden get header og heri søge efter “Referer” og redirecte til denne. Referer er en del af http’en og udhentes herfra, det er en reference til den tidligere url side, hvor en request blev sendt fra (<https://web.dev/referrer-best-practices/>, 16/12-21).


@ExceptionHandleren skal bruge parameter i form af exceptions. Hertil vedlægges exceptionens fejlbesked i en model, der videresendes til thymeleaf under navnet “message”.




Project Calculation Tool

Invalid email or password

To view projects - log in

 hejsa@h.dk

 ...

Sign In

Et eksempel på en sådan fejlbesked sendt til brugerfladen kunne se sådan ud.

Thymeleaf

Der er gennem html'en i webapplikationen brugt Thymeleaf, som er en moderne java skabelonsmiljø for web og standalone miljøer(<https://www.thymeleaf.org/> 16-12-2021). Brug af thymeleaf gør systemet mere dynamisk, idet data kan sendes fra brugerfladen, til systemet og ligeledes at data kan sendes fra systemet og ud til brugeren.

En af Thymeleafs funktioner er at kunne iterere over lister ved brug af for each løkker, hvor data sendes fra det øvrige system. På denne måde kan der i webapplikationen blive vist et helt projekt og dets delprojekter, samt opgaver, gennem ét projekt objekt, hvilket vises nedenfor.

```
<div id="overview" th:each="sp: ${project.getSubProjects()}">
  <h3 th:text="${sp.name}"></h3> <!--+ ' ' + ${sp.time}-->
  <div id="tasks" th:each="task: ${sp.getTasks}">
    <h4 th:text="${task.name} + ' ' + ${task.getTimeHours()}"></h4>
  </div>
</div>
```

En yderligere thymeleaf funktion, som er essentiel for systemet, er brug af fragments. Dette har til funktion at genbruge html kode der går igen på flere sider, herunder headeren og footeren.

Måden dette implementeres på, er ved at lave en html fil, 'general', til at have de elementer, der gøres brug af gennem programmet. Herefter defineres elementet med th:fragments tag som vist nedenfor til senere reference i andre html filer.

```
<header th:fragment="header">
```

Når et fragment skal benyttes i anden html fil, refereres som vist nedenfor. Således bedes thymeleaf om at indsætte fra general html filen, i fragments mappen det specifikke fragmentet navngivet 'header'.

```
<th:block th:insert="fragments/general :: header"></th:block>
```

Endnu en funktion som thymeleaf tilbyder, er at kunne videregive data med hjælp fra springs @PostMapping annotation. Dette gøres ved hjælp af forms, som har en submit felt, der sender data ned til spring controller laget i systemet.

```
<form action="addTask" method="POST">
  <input type="hidden" name="subprojectId" th:value="${sp.getSubProjectID()}">
  <input type="text" name="taskName" placeholder="Task Name" class="textfield" required>
  <input type="text" name="taskTime" placeholder="Hours" class="textfield" required>
  <input type="submit" value="Add" class="submit">
</form>
```

Ovenstående viser, at der bliver lavet et type hidden felt hvor thymeleaf gemmer værdien fra et delprojektID, som derefter bliver submitted og sendt ned i controller laget til yderligere data manipulation. Dette er en normal måde at sende data fra side til side gennem et system.

Da Thymeleaf netop er god til at overføre data fra backend til frontend betyder det også, der kan gives fejlbeskeder til brugeren. Derfor er der blevet arbejdet med en error klasse i html, som i main.css filen er stilet, dette sikre konsistens af alle fejlbeskeder som brugeren ser.

```
<p th:text="${message}" class="error"></p>
```

Ovenstående ses det p tag som fejlbeskeden lægges i.

Database

Gruppen har igennem projektet gjort brug af en MySQL database, til at opbevare informationer om data. Til undgåelse af redundant data og gøre det lettere at vedligeholde, blev der desuden gjort brug af normalization, hvor informationen i databasen blev delt ud i flere tabeller. Dette blev gjort i stedet for opbevaring af alt data, i en enkelt tabel (Murach's, 2019, s. 319).

Gruppen fravalgte brugen af stored procedures, for at skille funktionalitet og dataopbevaring ad, samt for adgang til brugen af generated keys. Hvorfor gruppen gjorde brug af sql queries i

intelliJ IDEA i stedet. Brugen af queries, sikrer databasens integritet, da det sikrer, at den data der bliver sendt ned, er nøjagtig og konsistent. Det er vigtigt, da den mindste stavfejl vil kunne medføre problemer i databasen (<https://www.netinbag.com/da/internet/what-is-database-integrity.html>, 16/12-21).

SQL

```
( sql: "INSERT INTO projects(project_name, fk_employee_ID) VALUE (?, ?);", Statement.RETURN_GENERATED_KEYS);
```

Ovenstående er et eksempel, fra gruppens system, på en insert statement i forbindelse med oprettelsen af et projekt. I statementen indsættes to spørgsmålstegn, som values, med henblik på henholdsvis at gøre det umuligt at foretage sql injections (https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html, 16/12-21), samt at gøre oprettelse af projekter dynamisk. Dette gøres ved at metoden modtager to parametre, som indsættes gennem prepared statements og altså ikke direkte i querien. Således er det dataen modtaget gennem brugerens input, som definere den værdi projektet oprettes med "project_name".

```
( sql: "SELECT * FROM projects WHERE project_id = ?;");
```

Til læsning af projekt er det nødvendigt at udføre en select statement, som det ses i ovenstående eksempel. Systemet medsender her et projekt id til at stå i stedet for spørgsmålstegnet, gennem prepared statements.

```
( sql: "UPDATE projects SET project_name = ? WHERE project_id = ?");
```

Der er gennem en update query, sikret mulig manipulation af projekterne. Dette ses i ovenstående udklip, fra gruppens system, hvor et projekt navn opdateres ud fra specifikt projekt id. Sidst vises nedenfor, et udklip af hvordan, det også er gjort muligt at slette et projekt. Dette gøres ved at vælge et projekt, på dets id og derefter sende en delete query til databasen.

```
( sql: "DELETE FROM projects WHERE project_id = ?;");
```

Udover valget om at benytte sig af sql queries, benyttes joins ikke. Dette skyldes omstrukturering af systemet, på baggrund af problematikker, ved overholdelse af GRASP principperne, ved brug af joins. Til overholdelse af domæne struktur, hvor hvert domæne skal stå for oprettelse af indehavende model, blev resultset sendt med ned i de forskellige repositories. Dette

gav komplikationer med iterator funktionaliteten af resultset, selv efter vejledning, hvor vejleders eksempel stod i samme problematik. Derfor blev valget om restrukturering taget.

Cascade

Cascade bruges på MYSQL databasen, til automatisk at slette børnerækker, som er forbundet via foreign keys, til forældre rækker. Når databasen modtager en delete statement fra webapplikationen, bliver cascaden 'on delete' eksekveret ned gennem de forskellige tabeller.

Cascade bliver sat på databasen under foreign key options under 'On Delete' og bliver sat på alle tabeller, der skal have en cascade effekt. (<https://www.javatpoint.com/mysql-on-delete-cascade>, 15-12-2021)

Foreign Key Options

On Update: NO ACTION

On Delete: CASCADE

```
CONSTRAINT `fk_project_id` FOREIGN KEY (`fk_project_id`) REFERENCES `projects` (`project_id`) ON DELETE CASCADE ON UPDATE NO ACTION
```

Test

Der er gennem programmet blevet testet på diverse klasser, i de forskellige domæner og den totale dækningsgrad ses i udklipet, nedenunder for henholdsvis klasser, metoder og linjer. I util mappen er det kun DBManager klassen som testes, da det ikke virkede meningsfuldt at teste exceptions klasserne, idet de kun består af konstruktører.

Element ^	Class, %	Method, %	Line, %
projectCalculationTool.employee	75% (3/4)	47% (9/19)	56% (30/53)
projectCalculationTool.project	40% (2/5)	50% (17/34)	43% (67/154)
projectCalculationTool.subproject	75% (3/4)	46% (13/28)	28% (33/114)
projectCalculationTool.task	75% (3/4)	41% (10/24)	26% (29/110)
projectCalculationTool.util	16% (1/6)	20% (2/10)	30% (8/26)

JUnit test

JUnit er et framework til unit tests af programmeringsproget java (<https://junit.org/junit4/> 16-12-2021), og en JUnit @Test er en metode i en klasse specifik til testning af programmer (https://www.vogella.com/tutorials/JUnit/article.html#unittesting_junit_test 16/12-21). Ved testning af gruppens program skal environment variables sættes forinden start af test, da testene ellers fejler.

Gruppen har gjort brug af JUnit testing, til sikring af at programmet opfører sig, som forventet og metoderne gør det de er lavet til. Til visning af at en metode, er en testmetode, bruges annotationen '@Test' i linjen over den. Nedenfor ses en test af, hvorvidt metoden createproject kaster en exception, ved forsøgt oprettelse af projekt, med for langt et navn.

[illegible]

Der er testet om de rigtige exceptions, kastes på de rigtige tidspunkter, i nogle af klasserne. Men idet systemets forskellige lag har mange klasser, med overvejende ens metoder, er der valgt ikke at teste alle metoder, i alle klasser. At teste hele programmet ville være noget man kunne gøre i konstruktionsfasen. Udnyttelse af denne måde at teste på skal sikre et solidt program, ved at teste diverse variationer af metoder tidligt, og sikre at fejl bliver fundet hurtigt, samt at skelettet af testning er klargjort til næste fase.

Alle variationer af valideringsmetoder er testet, da dette anses som noget af det vigtigste for programmet. Idet kalkulering af tidsforbrug fra hvert delprojektselement var en del af kerneopgaven, blev test af delprojekts udregning af tid til dage udført. På baggrund af dette blev udregning af tid på projekter, testet til 100% dækningsgrad.

Under JUnit tests er der taget brug af test execution order(<https://www.vogella.com/tutorials/JUnit/article.html#test-execution-order> 16-12-2021).

Denne specifikke test type, sørger for at den rækkefølge der bliver sat, er den rækkefølge testene vil blive eksekveret i. Dette har været nødvendigt i selenium test, af oprettelse af projekt, delprojekt og opgaver. Nedenfor ses udklip af brugen af rækkefølge på test; det ses at `testNavigationValidProject` skal eksekveres først og derefter `testNavigationInvalidProject`. Inden `@order` annotationen kan blive brugt skal testklassen have `@TestMethodOrder` annotationen.

```

@Test
@Order(1)
public void testNavigationValidtProject() {
    WebElement projectName = selenium.findElement(By.name("projectname"));
    projectName.sendKeys( ...charSequences: "SeleniumProjectTest");

    projectName.submit();

    assertEquals( expected: "SeleniumProjectTest", selenium.findElement(By.name("SeleniumProjectTest")).getText());
}

@Test
@Order(2)
public void testNavigationInvalidtProject() {
    WebElement projectName = selenium.findElement(By.name("projectname"));
    projectName.sendKeys( ...charSequences: "Project name cannot be longer then 45 characters this is the test");

    projectName.submit();

    assertEquals( expected: "Profile", selenium.getTitle());
}

```

Integration test

Integrations tests omhandler at teste de metoder der har med repository at gøre. Dette vedrører createTask metoden, i opgave repository, som har til opgave at lave opgaver og tilføje dem til delprojekts arrayliste af opgaver.

```

@Test
public void testCreateTask() throws TaskException {
    // Arrange
    SubProject subProject = new SubProject( subProjectName: "Fødselsdagssnacks");
    subProject.setSubProjectID(1);

    ArrayList<Task> tasks = new ArrayList<>();
    Task bakeTask = new Task( time: 4, name: "Bake cake");

    tasks.add(bakeTask);

    subProject.setTasks(tasks);

    // Act
    taskRepository.createTask(subProject);

    // Assert
    assertTrue( condition: bakeTask.getTaskID() != 0);
}

```

Herover bliver testet om opgave kommer ud med et ID på 0, da dette ikke burde være muligt, hvorfor det bør være en gennemført test.

Ui test - Selenium

Ved brug af Selenium test i programmet, er der behov for en passende driver, som skal passe til det pågældende operativsystem, henholdsvis Windows og macOS. Denne driver er vedlagt i programmet under webdriver mappen, nede ved tests. Til brug af samme, skal det opsættes om man er på mac eller windows. Disse er i setup, hvor man udkommentere den, som ikke skal anvendes og sørger ligeledes for den anden ikke er det. En vigtig note til brug af Selenium tests, er at programmet skal være kørt inden kørsel af selenium testen.

```
@BeforeAll
| public static void setUp() {
|     // Load selenium driver
|     // download chromeDriver.exe fra http://chromedriver.storage.googleapis.com/index.html?path=96.0.4664.45/
|
|     //Windows chromedriver
|     System.setProperty("webdriver.chrome.driver", "src/test/java/webDriver/chromedriverWindows.exe");
|
|     //Mac chromedriver
|     //System.setProperty("webdriver.chrome.driver", "src/test/java/webDriver/chromedriverMac");
|
|     selenium = new ChromeDriver();
| }
```

Ved testning af programmet benyttede gruppen sig af en Selenium test, af login. Hermed kunne test af interaktion med brugerfladen imiteres og automatiseres. Dette blev gjort med udgangspunkt i eksempel på github(<https://github.com/Tine-m/2.semUITestAutomation/blob/main/SystemTestAutomation/src/test/java/GoogleDemoTest.java> 16-12-2021), hvor testen blev tilpasset login testning af systemet. Grundet kun en mulig bruger, foretages ét validt login, mens flere ikke mulige login, forsøges testet. Da denne del af programmet var testbar i de tidligere iterationer der foregik før deployment, blev den oprettet ved at skulle navigere til <http://localhost:8080>, systemets login side, og senere ændret til <https://projectcalculationtool.herokuapp.com/> efter deployment til heroku.

Konklusion

Projektet er udarbejdet ud fra kerneopgaven, fremsat af Alpha Solutions. Opgaven er opdelt i fire use cases, hvoraf de første to iterationer i etableringsfasen, udarbejder registrer projekt og vedligeholdelse af projekt. De øvrige to use cases login og projektoversigt er tydeligt mindre prioriteret, men er sideløbende udviklet på, således færre større ændringer er at håndtere senere.

Det konkluderes at slutresultatet er et robust system oplagt til videre udvikling. Det forventes at imødegå disse i konstruktionsfasen. Procesmæssigt vurderes der ikke akkurat brug af

Unified Process, idet etableringsfase iteration tre, i højere grad er ufokuseret, både ved delt opmærksomhed på rapport, men også ved “udfyldelse” af produkt, hvilket kunne argumenteres at høre sig til i konstruktionsfasen.

Mangler/refleksion - Hvor langt nåede vi?

En kendt fejl er at programmet crasher, hvis forbindelsen til databasen fejler. Dette kunne imødegås ved en håndtering af den exception, som smides når dette sker.

Herudover er der en vigtig mangel, med henblik på udarbejdede brugeroplevelses testen, at der ingen sikkerhed er ved sletning elementer. Det virker som en vigtig forbedring ,særligt med henblik på at kunne slette et helt projekt. Det kunne sikres ved en dialogboks, som kunne poppe op og spørge den ansatte, om de var sikre på sletningen inden den réelle sletning.

Endnu en mangel er, ifølge use casen, at kunne se en oversigt over projekter, med tids kalkulering. Dette opfyldes ikke idet den hverken er stylet og ej heller viser opsummeringen, af hvor meget tid hvert element måtte tage, ligesom på projektsiden.

Herudover fandt gruppen frem til nogle mulige styling forbedringer; returknop fra overview side, der fører tilbage til projekt side og projektside styling således at diverse ikoners funktioner var mere intuitive.

En sidste mangel er, hvordan, ved brugen af ‘referer’ fra http header i `@ExceptionHandler`, skriver fejlbeskederne op i url’en, hvilket ved flere fejlforsøg, også af forskellige typer af fejlbeskeder, alle ender i url’en. Dette resulterer i endnu en fejl, idet den første fejl i url’en er den som sendes videre til thymeleaf, hvorfor fejlbeskeden for brugeren aldrig ændrer sig.

Videreudvikling

Da der i dette projekt blev fokuseret på mvp kravene, er det klart, der er manglende features. Derfor kunne en visualisering af projektet, ved et gantt diagram, introduktion af kompetencer/ressourcetyper dag for dag og et belastnings overblik, over hver arbejdsdag tilføjes.

Desuden er styling og flere brugertest, helst med reel kunde og ikke proxy, til sikring af et produkt, slutkunde vil være tilfreds med, at ønske.

Litteraturliste

Larman, Craig, Applying UML and Patterns | Pearson education, inc. 3. udgave, 2015

Murach, Joel, MySQL | Mike Murach & Associates, inc 3. udgave, 2019

- <https://www.alpha-solutions.com/da/om-os> Hentet den 03. december 2021
- <https://www.baeldung.com/thymeleaf-in-spring-mvc> Hentet den 07. december 2021
- <https://maven.apache.org/what-is-maven.html> Hentet den 01. december 2021
- <https://maven.apache.org/> Hentet den 01. december 2021
- (<https://www.infoworld.com/article/3388036/what-is-jdbc-introduction-to-java-database-connectivity.html>) Hentet den 11. december 2021
- <https://www.selenium.dev/> Hentet den 04. december 2021
- [Distributed Workflows](#) Hentet den 30. november 2021
- <https://www.projectmanager.com/software> Hentet den 12. december 2021
- <https://www.agilealliance.org/glossary/pairing/#q=~> Hentet den 13. december 2021
- <https://www.slideteam.net/time-quality-cost-cost-resources-time-schedule-quality-scope.html> Hentet den 14. december 2021
- <https://www.alpha-solutions.com/da> Hentet den 01. december 2021
- <https://martinfowler.com/bliki/PresentationDomainDataLayering.html> Hentet den 15. december 2021
- <https://regexr.com/3e48o> Hentet den 16. december 2021
- https://www.w3schools.com/java/java_regex.asp Hentet den 16. december 2021
- [spring/webrequest/setAttribute](#) Hentet den 16. december 2021
- <https://capien.co/shneiderman-eight-golden-rules-interface-design> Hentet den 16. december 2021
- <https://www.nielsgamborg.dk/?p=gestaltlovene> Hentet den 16. december 2021
- <https://digitypes.dk/hvordan-bruger-man-gestaltlovene-design/> Hentet den 16. december 2021
- <https://fontawesome.com/v5.15/icons?d=gallery&p=2> Hentet den 16. december 2021
- https://developer.mozilla.org/en-US/docs/web/html/global_attributes/id Hentet den 16. december 2021
- https://developer.mozilla.org/en-US/docs/web/html/global_attributes/class Hentet den 16. december 2021

- docs.spring.io, Hentet den 15. december 2021
- <https://web.dev/referrer-best-practices/> Hentet den 16. december 2021
- <https://www.thymeleaf.org/> Hentet den 16. december 2021
- <https://www.netinbag.com/da/internet/what-is-database-integrity.html> Hentet den 16. december 2021
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html Hentet den 16. december 2021
- <https://www.javatpoint.com/mysql-on-delete-cascade> Hentet den 15. december 2021
- <https://junit.org/junit4/> Hentet den 16. december 2021
- https://www.vogella.com/tutorials/JUnit/article.html#unittesting_junit_test Hentet den 16. december 2021
- <https://www.vogella.com/tutorials/JUnit/article.html#test-execution-order> Hentet den 16. december 2021
- <https://github.com/Tine-m/2.semUITestAutomation/blob/main/SystemTestAutomation/src/test/java/GoogleDemoTest.java> Hentet den 1. december 2021

Bilag

Bilag 1: Simply database

Database-server

- Server: mysql21.unoeuro.com via TCP/IP
- Servertype: Percona Server
- Serverforbindelse: **SSL benyttes uden certificeringsautoritet** ⓘ
- Serverversion: 5.7.35-38-log - Percona Server (GPL), Release 38, Revision 3692a61
- Protokolversion: 10
- Bruger: kibshede_dk@94.231.103.77
- Servers tegnsæt: cp1252 West European (latin1)

Bilag 2: Iterationsplan

Iterationsplan

I1 (START 22/11-21 SLUT 23/11-21)

Sikring af forståelse af problemstilling

Iterationsplan

Ordliste

Use case model start(brief text)

Liste over use cases

Stakeholder analyse tabel

Stakeholder axis

Udvidet Risikoanalyse

Slut forventning: At være færdig med inception fasen, hvorfor requirements og business modelling er fremskrevet. Desuden at projekt vurderes feasible.

E1 (START 23/11-21 SLUT 30/11-21)

Planlæg iteration E2

Oprette Git repository

Oprettelse af database

kigge på diagrammer - hvilke overvejer vi at medtage - hvilke er meningsgivende

- Domain model
 - Ordliste
 - State Machine Diagram
 - Pakke Diagram
 - ER diagram
 - Design
- Klassediagram

- Use case fully dressed

Start kode

- Lav header og footer i general fragments
- hardkodet login kun godkendt én bruger (ikke oprettelse af brugere)
 - select user
- oprette og læse projekt
 - create og read
 - html
- oprette og læse subproject
 - html
 - CR

Slut forventninger: At kunne login med hardkodet bruger, at kunne oprette ét projekt i dets fulde nedbrudte form. Det forventes desuden at relevante test færdiggjort og en begyndelse på web html med begrænset css.

E2 (START 1/12-21 SLUT 7/12-21)

Lave plan for næste iteration E3

Kig på relevante diagrammer for denne del SD

Genbesøg klassediagram og use cases

validering

restrukturere program

- exception handling - kun throw sql exception i repository
- lav specifikke exceptions

implementeringer

- kalkulering af timer
- Kalkulering i dage
- Gøre muligt at se overblik

html og controller funktionalitet til display sub project side og task side

validering af det lavet i e1

(CR)UD tilføjes til hele projekt

unit test

oprette SQL script

finpudsning - css

Deploy til Heroku

LOOK: Forventes MVP fuldt funktionelt. Der forventes også modtager feedback på user test.

E3 (START 8/12-SLUT 10/12)

UX test

Rapportskrivning

Sikring af krav forståelse

- Spørgsmål til undervisere

readme fil start

html og css af program

Kigge diagrammer igennem

Sikre testdata og test

LOOK: Forventes et færdigt produkt - self med mangler, men finpudsning stoppes. Rapport påbegyndes i større træk og spørgsmål stilles til undervisere. Opdatering af diagrammer udføres med henblik på feedback og rapportskrivning.

Slutiteration (START 13/12 SLUT 16/12)

Skrive rapport

LOOK: Aflevering af rapport og projekt. Projekt er funktionelt og deployet. Rapport er færdigskrevet, gennemlæst og afleveret.

Bilag 3: User test

UX TEST

Det ligner en skamside

Minimalistisk

Kunne ikke finde project knappen

profile side

“Ligner en side lavet af amatører”

lidt mærkeligt den ikke gemmer workdays

ville gerne kunne se projektindhold fra profile side

Edit knap ved project skal hedde edit project name

Det kunne være nice med et overview ude i profilesiden.

Delete knap skal have en are you sure prompt

Project side

knapperne er lidt mærkelige i forhold til hvor de sidder

task add knapperne er for små

Der står ikke hvad tidsintervallet er i i add task

Forsøgte at tilføje to tasks i to forskellige subprojects.

“irriterende den ikke kan gemme teksten imens man adder en anden task”

Det er en tidsberegner.

ingen tilbageknap er irriterende

tænker edit er edit project og ikke edit navn

en knap tilbage til profile side

Skulle måske have lavet det ud fra noget andet end Alpha Solutions hjemmeside