



Projektkalkulationsværktøj

2. Semester Eksamensprojekt

E24C - Gruppe 1 - 28/05/2025

Navn	Fødselsdato	Github navn
Simon Pedersen (SP)	18/01/1996	SimonP-Bot
Victor Krogh Jensen (VJ)	15/08/1996	Viictator
Hannibal Ussing-Widholm (HU)	05/06/2002	huw02
Zahaa Al-khakani (ZA)	04/05/1999	Zahaawii

Github repository:

[Github repository](#)

Scrum board:

[Scrum board](#)

Deployed version:

<https://g1ssolutions.azurewebsites.net/>

Indledning	5
Introduktion til projektet	5
Formål	5
Problemformulering	5
Underspørgsmål	5
Afgræsning	6
Teknologivalg	6
IDE og værktøjer	6
Backend	6
Database	6
Backend	7
Eksterne tjenester	7
Dependencies	7
Hovedafsnit	8
IT og forretningsforståelse	8
Feasibility study	8
Introduktion	8
Teknisk feasibility	8
Økonomisk feasibility	8
Udgifter	8
Indtægt	9
Operationel og organisatorisk feasibility	9
Dataetisk & juridisk feasibility	9
Go eller No go	10
Systemudvikling	11
Indledning	11
Kravspecifikation	11
Funktionelle krav	11
Ikke-funktionelle krav	12
User stories	12
Design	13
Domain model	13
ER-model	15
De 3 normalformer	17
SQL Scripts	18
User interface design	19
Designprincipper:	22
Implementering	24
Arkitektur/UML package diagram	24
Aktivitetsdiagram 1	26
Navigationsflow	26
Aktivitetsdiagram 2	30

Navigationsflow	31
Klassediagram	32
Designovervejelser og principper	33
MVC-arkitektur	33
Dependency Injection og Inversion og Control	34
GRASP	34
Low Coupling	34
High Cohesion	35
Kodeeksempler og beskrivelser	35
Login:	35
Sort by skills:	38
Sort my sub tasks:	46
Status på implementering	51
Nuværende implementeringer:	51
Fremitidige implementeringer:	51
Forbedringer i systemet	52
Test	53
Integrationstest	53
Controllertest	56
Maven yml test	58
Scrum processen	59
Sprint 1	59
Sprint planning	59
Product backlog	60
Sprint backlog	60
Daily stand up	60
Sprint review	61
Sprint retrospective	61
Sprint 2	61
Sprint planning	61
Product backlog	62
Sprint backlog	62
Sprint review	62
Sprint retrospective	62
Konklusion på scrum process	63
CI/CD (Continuous Integration / Continuous Deployment) og Deployment	64
Continuous Integration (CI)	65
Continuous Deployment (CD)	66
Gennemgang af workflow-konfiguration	67
Azure setup og Miljøer	67
Qodana	68
Brug af LLM (Large language models) som sparringspartner	70
Kørselsvejledning	71
Lokal opsætning	71

Konklusion	72
Litteraturliste	73
Indledning	73
IT forretningsforståelse	73
Systemudvikling/teknologi	73
Domain model	74
Kode	74
Database	75
User interface design	76
Scrum	76
CI/CD	76
Viden	77
Bilag	77
Daily scrum	87
05-05-2025	87
06-05-2025	88
07-05-2025	90
12-05-2025	91
13-05-2025	93

Indledning

Introduktion til projektet

Alpha Solutions er en IT virksomhed med over 80 medarbejdere fordelt på afdelinger i tre forskellige lande. Denne internationale struktur kan give udfordringer i forbindelse med koordinering og planlægning af projekter på tværs af teams og geografiske placeringer.

På trods af virksomhedens vækst og mange års erfaring inden for IT-udvikling har Alpha Solutions identificeret et behov for et værktøj, der kan skabe bedre overblik og struktur i deres projektarbejde. Derfor er der efterspurgt et system, der kan hjælpe med administration og nedbrydning af projekter – herunder håndtering af delprojekter, opgaver og tidsforbrug.

Det ønskede produkt skal understøtte både projektlederens behov for overblik og planlægning samt medarbejderens behov for indsigt i egne opgaver. Systemet skal dermed bidrage til en mere effektiv og gennemsigtig projektstyring i organisationen.

Formål

Formålet med dette projekt er at udvikle et værktøj, der skaber overblik og struktur i Alpha Solutions' projektstyring. Projektet giver os som studerende mulighed for at anvende de tekniske og metodiske færdigheder i en realistisk kontekst med en reel virksomhed som samarbejdspartner. Produktet skal være et relativt simpelt og brugervenligt program til projektledere og medarbejdere. Programmet skal være i stand til at oprette projekter, sub-projekter og tasks derunder, med tilknyttede deadlines og tidsestimater. Dette har til formål at give projektlederen det nødvendige overblik for at kunne få en virksomhed til at køre rundt økonomisk.

Problemformulering

Hvordan kan vi udvikle et digitalt værktøj, der giver Alpha Solution bedre overblik over deres projekter, ressourcer - herunder opgavestyring, tidsestimering og budget.

Underspørgsmål

- Hvordan kan projektlederen benytte systemet til at planlægge og følge projekternes fremskridt samt udfordringer?
- Hvordan kan vi sikre os, at arbejdsopgaver og ressourcer fordeles realistisk og effektivt?

- Hvordan kan projektet hjælpe medarbejderne med at få indsigt i deres egne opgaver og deres deadlines?
- Hvordan kan systemet understøtte projektlederens estimeringer vedrørende tid- og budgetopfølgning?

Afgrænsning

Projektet fokuserer grundlæggende på de kernebehov, som Alpha Solutions har givet udtryk for, at de har behov for. Projektet fokuserer på at udvikle et system der kan bruges til at organisere projekter, opgaver, tidsestimering og budgetopfølgning.

Teknologivalg

For at realisere et projektkalkulationsværktøj, der matcher Alpha Solutions behov og krav til funktionalitet, sikkerhed og brugervenlighed, har vi udvalgt en række teknologier, der understøtter både backend- og frontendudvikling, test, deployment og versionsstyring.

IDE og værktøjer

- IntelliJ IDEA 2024.2 (Ultimate Edition) Build #IU-242.20224.300, built on August 6, 2024
- Github til versionsstyring og samarbejde
- Github Actions til CI/CD
- Github Projects
- Maven til dependency management og build
- Figma til at lave design for hjemmesiden
- Visual paradigm til visualisering af modeller

Backend

- Java 21
- Spring Boot framework til webapplikationen og arkitektur
- Spring web til endpoints og controllers
- Spring dev tool for hurtigere udvikling og auto-reload
- JBCrypt til hashing af adgangskoder og sikre login
- JDBC template til database interaktion

Database

- MySQL primær database i produktion

- H2 Database in memory test database til test

Backend

- Thymleaf
- HTML
- CSS

Eksterne tjenester

- Sendgrid bruges til sende mail¹

Dependencies

Ser man på koden, så bygger den på en række vigtige dependencies, som skal være korrekt installeret for at systemet fungerer som forventet. Nedenfor er de vigtigste dependencies, som systemet benytter:

- Spring Boot (3.4.4) - Bruges til at bygge en webapplikation i Java.
- Spring Boot Starter Web - Håndterer HTTP-anmodninger.
- Spring Boot Starter JDBC - Bruges til at kommunikere med MySQL-databasen via JDBC.
- Spring Boot Starter Thymeleaf - Gør det muligt at integrere Java-variabler direkte i HTML-siderne for at systemet er mere dynamisk.
- MySQL Connector/J - Driveren som gør det muligt for applikationen at forbinde og arbejde med en MySQL-database.
- H2 Database - En in-memory database, som bruges til integrationstest.
- Spring Boot DevTools - Bruges til at understøtte hot reloading og forbedre udviklingsprocessen.
- Spring Boot Starter Test - Indeholder testværktøjer som JUnit til at skrive og køre test.
- jBCrypt (v0.4) - Bruges til at kryptere brugernes adgangskoder, som opfylder GDPR-krav.
- SendGrid Java - Gør det muligt at sende en email direkte fra applikationen.

Disse dependencies hentes og håndteres af Maven, som automatisk sørger for, at de nødvendige biblioteker er tilgængelige og at projektet kan bygges og køres korrekt.

Link til Github Readme: [README](#)

¹ <https://github.com/danvega/sendgrid>

Link til Github Contribute: [Contribute](#)

Hovedafsnit

IT og forretningsforståelse

Feasibility study

Introduktion

I denne feasibility study undersøges gennemførigheden af projektet Alpha Solutions. Formålet er at vurdere, om projektet er realistisk at gennemføre inden for de givne rammer. Som f.eks teknisk, økonomisk, tidsmæssigt og i forhold til data etik og sikkerhed. Analysen skal danne grundlag for beslutningen om, hvorvidt udviklingen af programmet bør påbegyndes.

Teknisk feasibility

Teamet har de nødvendige kompetencer og færdigheder der skal til for at udvikle en MVP, der tilfredsstiller opgavens krav. På baggrund af vores indledende user stories kan vi se en efterspørgsel om kryptering af passwords og visuelle charts på frontenden til at hjælpe brugere med hurtigt at få overblik. Disse tiltag vil kræve, at teamet tilegner sig ny viden og kan blive teknisk risikabelt.

Økonomisk feasibility

Siden vi er i gang med vores eksamsprojekt, vurderer vi at en økonomisk feasibility study med dette i tankerne ville blive tynd, fordi der ikke er nogen former for økonomiske transaktioner inkluderet, samt alle udviklingsværktøjer allerede er sat til rådighed af skolen. Derfor opdiger vi et scenerie hvor vi er et lille softwareløsningsfirma og opdiger nogle tal.

Udgifter

Udgifterne kan deles op i tre hovedkategorier. Software, hardware og mandetimer. Den maksimale udgift på software er estimeret til 10.000 kr. til de nødvendige software licenser. Det forventes at størstedelen af softwaren er gratis eller allerede tilgængelig via eksisterende licenser. Programmet sættes op direkte via kundernes cloud services på deres budget, derfor vil vores firma ikke have nogle udgifter på deployet databaser og webapplikationer. Investeringer i hardware forventes at koste omkring 40.000 kr. Det indebærer computere eller andre nødvendige enheder til udvikling eller test. Den største udgiftspost er udviklingstiden (mandetimer). Vi estimerer at udviklingen af programmet vil

tage omkring en måned, når teamet består af fire medarbejdere med en månedsløn på 50.000 kr.

Indtægt

Projektpris kalkuleret på baggrund af udgifter + en profit på 100.000 kr.

Projektets pris over for kunden fastsættes med udgangspunkt i de samlede udgifter samt et tillæg for profit. Med en ønsket profit på 100.000 kr. og en samlet udgift på 250.000 kr. kalkuleres projektprisen til at være 350.000 kr.

Udgifter	Beløb (kr.)	Indtægter	Beløb (kr.)
Software	10.000		
Hardware	40.000		
Mandetimer	200.000	Projektpris	350.000
I alt	250.000	I alt	350.000

Profit	100.000
--------	---------

Operationel og organisatorisk feasibility

Når vi overleverer projektet, indebærer det opsætning på kundens infrastruktur og en dybdegående gennemgang af hvordan programmet virker udadtil, men også bag systemet samt en skriftlig træningsmanual. Dermed antager vi at Alpha Solutions selv kan varetage al vedligeholdelse og efterfølgende udvikling, der må forekomme. Hvis Alpha Solutions i fremtiden ønsker vores hjælp til flere features eller videreudvikling, vil dette betegnes som et nyt projekt og behandles derefter.

Dataetisk & juridisk feasibility

Projektet vurderes at være både dataetisk og juridisk gennemførligt. Systemet implementeres på kundens egen cloud-infrastruktur, hvilket betyder, at ansvaret for drift, vedligeholdelse og datasikkerhed primært ligger hos kunden. Dermed reduceres vores firmas direkte ansvar for håndtering af persondata efter overlevering.

Systemet håndterer persondata i form af brugernavne og krypterede adgangskoder. Når en administrator opretter en ny bruger, genereres et brugernavn og et midlertidigt password, som overleveres til brugeren. Da denne overlevering håndteres af kunden selv, anbefales det, at der etableres interne retningslinjer for at minimere risikoen ved denne proces, f.eks. for at undgå usikker overlevering via e-mail eller fysiske noter.

Ved oprettelse kan brugeren tilføje selvvalgt kode, hvilket gør informationen endnu mere personfølsom, brugeren bliver først aktiveret efter admin godkendelse. For at beskytte disse oplysninger gemmes alle passwords krypteret ved brug af Bcrypt, hvilket betyder, at i tilfælde af et datalæk vil adgangskoderne ikke kunne aflæses direkte. Anvendelsen af kryptering lever op til kravene i GDPR artikel 32 (Datatilsynet, 2018) om passende tekniske og organisatoriske sikkerhedsforanstaltninger.

Systemet sikres yderligere gennem opsætning af firewall-regler, således at applikationen kun kan tilgås internt på kundens netværk. Risikoen for eksternt databrud minimeres dermed betydeligt og kan kun opstå i tilfælde af et sikkerhedsbrud på kundens cloud service.

Projektet vurderes til at være dataetisk og juridisk feasible, forudsat at kunden implementerer passende interne procedurer for håndtering af overrækkelse af login oplysninger og vedligeholder et højt sikkerhedsniveau på deres cloud-platform.

Go eller No go

På baggrund af vores vurderinger anser vi projektet som teknisk, økonomisk, operationelt, dataetisk og juridisk gennemførligt.

På den tekniske del er projektet gennemførligt med stor sikkerhed, minimum på MVP niveau. Udfordringer kan forekomme når teamet skal tilegne sig ny viden omkring kryptering af passwords og overførsel af data til diagrammer eller andet visuelt på frontenden som kan give et klart overblik for brugeren.

Økonomisk set er projektet realistisk for et lille softwarefirma, baseret på opfundne tal, hvor udgifterne til software, hardware og mandetimer samlet estimeres til 250.000 kr.

Projektpriisen fastsættes til 350.000 kr., hvilket sikrer en ønsket profit på 100.000 kr.

Operationelt antager vi, at opsætning og overlevering sker direkte på kundens cloud-infrastruktur, understøttet af en skriftlig træningsmanual. Alpha Solutions forventes herefter selv at kunne stå for den fremtidige drift og vedligeholdelse. Ønsker kunden yderligere udvikling, vil dette blive behandlet som nye projekter.

Dataetisk og juridisk er projektet gennemførligt, da persondata håndteres forsvarligt med kryptering af adgangskoder via Bcrypt og begrænsning af adgang gennem firewall-regler. Ansvaret for drift og datasikkerhed overgår til kunden ved overlevering, hvilket minimerer vores videre forpligtelser. Der anbefales interne retningslinjer for sikker overlevering af loginoplysninger, for at reducere mulige risici, når en ny bruger oprettes og overrækkes.

Samlet set vurderer vi, at projektet kan gennemføres uden væsentlige tekniske, økonomiske, operationelle, dataetiske eller juridiske udfordringer.

Systemudvikling

Indledning

Ud fra følgende informationer påbegynder vi arbejdet med systemudvikling, hvor formålet er at omsætte forretningsideen til systemudviklingsmodeller der understøtter kundens krav om produktet.

For at forstå og modellere kundens ideer anvender vi først en analytisk model i form af kravspecifikation, user stories og domain model. Herefter benytter vi designmodeller til at visualisere systemets opbygning og sikre en struktureret samt kvalitetsbevidst tilgang til udviklingen af det ønskede produkt.

Kravspecifikation

Alpha Solutions er en mellemstor international virksomhed med over 80 medarbejdere fordelt på kontorer i flere lande. Denne geografiske spredning skaber udfordringer i forbindelse med planlægning og koordinering af projekter. For at imødekomme disse udfordringer har Alpha Solutions efterspurgt et digitalt værktøj, der kan hjælpe med at nedbryde, planlægge og følge op på projekter.

Funktionelle krav

Produktets hovedformål er at nedbryde komplekse projekter i overskuelige enheder, hvilket gøre via det følgende hierarki ønsket af Alpha Solutions:

Projekter > Sub-projekter > Task > Sub-task

Ved at realisere et produkt, der skal understøtte denne struktur, skal følgende funktioner etableres:

- Oprettelse, visning, redigering og sletning (CRUD) af projekter, sub-projekter, task og sub-task
- Akkumulere tidsestimering på baggrund af sub-task, task og sub-projekts.
- Mulighed for at tilknytte medarbejder med specifikke kompetencer til enkelte projekter eller opgaver.
- Sortering og filtrering af opgaver efter fx. prioritet, status eller kompetence
- Login-funktion med rollebaseret adgang (admin, projektleder, medarbejder)

Ikke-funktionelle krav

- Anvendelse af MySQL-database som primær database
- Applikationen udvikles som en webapplikation i Spring Boot
- Frontend skal være baseret på Thymeleaf + HTML/CSS og tage hensyn til brugervenlighed ved at følge designprincipper (Golden rules)
- Der skal være sikkerhed omkring brugerlogin med kryptering af passwords
- Applikationen skal kunne deployes til Azure
- Projektet skal understøtte CI/CD gennem Github Actions med automatiseret test.
- Dokumentationen af produktet skal være nemt tilgående og forståelig for udviklere i form af README og CONTRIBUTOR guidelines.

User stories

For at imødekomme kundens krav til funktionaliteten af produktet, har vi udarbejdet en række user stories baseret på de informationer, vi har modtaget fra kunden. User stories hjælper med at konkretisere de processer og funktioner, som produktet skal understøtte.

Ved at udvikle og implementere de beskrevne scenarier sikrer vi, at produktet lever op til kundens ønsker og forventninger.

Her er tre eksempler på de udarbejdede user stories. Alle user stories kan ses i Bilag 4 eller på vores [Scrum board](#):

User story	Acceptkriterier
Som bruger vil jeg kunne logge ind, så jeg kan tilgå mine projekter	Når jeg logger ind, skal jeg have en startside hvor jeg får bekræftet at jeg er logget ind
Som projektleder vil jeg kunne se, oprette, opdatere og slette medarbejder i systemet, så de kan tilgå systemet eller fjernes fra systemet	Jeg tilgå en side hvor jeg kan oprette en ny, opdatere eller slette en bruger. Jeg skal derefter have en bekræftelse på at brugeren er oprettet.
Som projektleder vil jeg kunne se, oprette, redigere og slette medarbejders skills, så jeg kan se hvilke kompetencer medarbejderne har	Ved oprettelse af bruger skal jeg have muligheden for at tilføje medarbejderens skills ved at krydse specifikke skills af på medarbejderen. Jeg skal have en "fjern" knap, som jeg kan trykke på ved hver medarbejders skills, så de kan fjernes. Jeg skal have en knap der kan tilføje og opdatere skills på en medarbejder, så de kan få nye eller opdaterede skills.

Gennem arbejdet med kravspecifikationer og user stories har vi opnået et klart helhedsbillede af, hvordan kunden ser og ønsker systemet udarbejdet. Med dette som udgangspunkt er vi nu i stand til at påbegynde designfasen og udforme en løsning, der matcher både forretningsbehov og tekniske rammer.

Design

Domain model

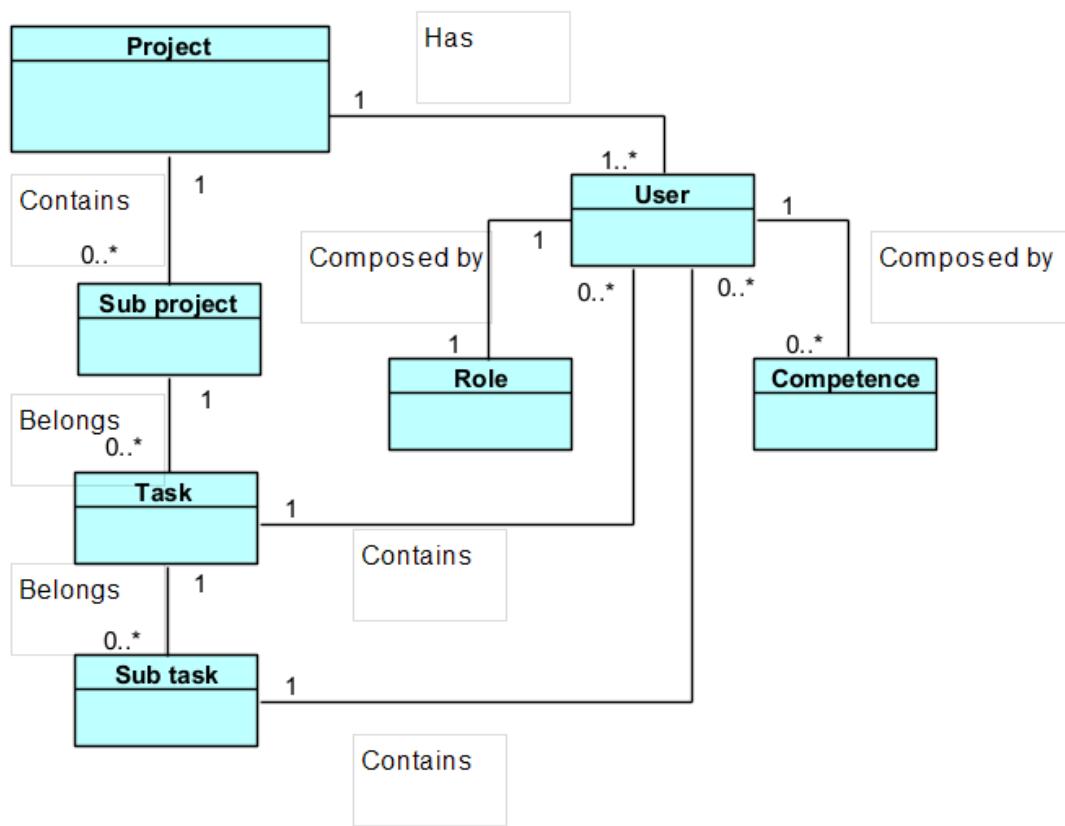
For at opnå en bedre forståelse af Alpha Solution behov og forretningslogik, har vi udarbejdet en domain model. Domain modellen bruges til at identificere og præsentere koncepter, aktører og relationer i det specifikke domæne².

² [Domain Modeling: What you need to know before coding | Thoughtworks](#)

Ved at benytte domain modellen oversætter vi virkelighedsforståelse og nedbryder det til konceptuelle klasser for at skabe et grundlag som reflekterer med strukturen for det givne område³.

Ud fra ovenstående beskrivelse af vores model, har vi opbygget den på følgende måde:

Oplægget fra Alpha Solutions' CTO, Klaus, samt vores møder med PO (Product owner) har vi identificeret følgende konceptuelle klasser:



Figur 1) Domain model

Opstillingen af vores model, er gjort af følgende årsag for at skabe en struktureret og dynamisk gennemgang af hvordan vi har analyseret modellen.

Et projekt kan bestå af et til flere sub project, men der kan kun eksisterer 1 projekt, Hvert sub project kan indeholde nul til mange task, og hver task kan igen have nul til mange sub task.

³ [Domain Modeling in Object-Oriented Analysis and Design\(OOAD\) | GeeksforGeeks](#)

En task kan dog ikke eksistere uden et tilknyttet sub project og en sub task kan ikke eksisterer uden en task

User har en central klasse, som har nul til mange til både task og sub task, hvilket betyder at ingen brugere eller flere brugere kan være tilknyttet til samme task og at en bruger kan være tilknyttet til flere opgaver.

Derudover er Employee sammensat af en role og nul til mange skills. En bruger kan kun have en rolle som fx projektleder eller en udvikler, men kan have flere kompetencer som frontend, backend og scrum master.

Denne opbygning som konstrueret giver os et dynamisk overblik for projektets hierarki og brugerens tilknytning.

Ved at udarbejde en domain model har vi opnået et overblik over systemets centrale entiteter, relationer og funktionalitet. Dette giver et solidt grundlag for at designe den underliggende database og dermed konstruerer en ER-model, der afspejler systemets datastruktur.

ER-model

For at kunne strukturere og implementere de relationer vi identificerede i vores domæne-model, har vi udarbejdet en ER-model, der afspejler databasens opbygning. I opbygningen af ER-modellen har vi haft fokus på at sikre entydige relationer mellem projekter, brugere, opgaver og roller. Dette er gjort med henblik på at opnå god performance og skalerbarhed i systemet. Vi lavede en ER-model tidligt i designprocessen, som kan ses i bilag 1. Under Sprint 1 opdaterede vi ER-modellen, hvor vi tilføjede flere kolonner til tabellerne og Assignees- og Relationstabeller, som var nødvendige for at udvikle det bedst mulige system. Denne ER-model kan ses i bilag 2.

Tabellerne i databasen dækker over følgende centrale begreber:

- **Employee:** Indholder informationer om brugere i systemet, som navn, e-mail, brugernavn, adgangskode samt en reference til deres rolle.
- **Roles:** Definerer hvilke roller der findes i systemet og bruges som reference i employee-tabellen.

- **Project, SubProject, Task og SubTask:** Et projekt kan have en eller flere subprojekter, som hver kan have en eller flere tasks, og hver task kan have flere subtasks. Disse tabeller har relationer til hinanden gennem fremmednøgler og anvender “ON DELETE CASCADE” for at sikre, at relaterede data slettes automatisk.
- **ProjectAssignees, TaskAssignees og SubTaskAssignees:** Anvendes til at koble medarbejdere til specifikke projekter, task eller subtasks med mange til mange relationer.
- **Skill og SkillRelation:** Bruges til at registrere medarbejdernes kompetencer og muliggør filtrering og sortering.
- **AwaitingEmployee:** En særskilt tabel til brugere, som endnu ikke er bekræftet i systemet. Dette gør det muligt at lave et godkendelsessystem, når man som bruger vil oprette sig i systemet.

I designet har vi været opmærksomme på at navngive tabellerne og kolonnerne konsekvent. Kolonnenavn som project_name, subproject_name og employee_name øger læsbarheden, mindske tvivl og misforståelser i udviklingsprocessen. Vi har gjort brug af “NOT NULL” constraints ved kolonner som employee_username og employee_password for at sikre at brugere ikke kan oprettes med manglende data.

Ved at anvende “AUTO_INCREMENT” på alle entiteters primærnøgler undgår vi risikoen for at primærnøglerne duplikeres ved at overlade generering af unikke ID'er til databasen.

Under sprint 2 valgte vi at fjerne tabellen subprojectAssignees, da vi vurderede at det var mest effektivt, at tilføje medarbejdere under oprettelsen eller redigeringen af en task eller en subtask. Listen af disse medarbejdere stammer fra tabellen projectAssignees. Derfor ville det være dobbeltarbejdet og overflødigt at tilføje medarbejdere til et subprojekt. Den endelige ER-model under Sprint 2 kan ses i bilag 3.

Samlet set giver ER-modellen et solidt datagrundlag, som understøtter systemets funktionalitet og kundens behov. Vores ER-model sikrer en logisk og overskuelig databasestruktur, som er nem at videreudvikle på.

De 3 normalformer

Ved design af databasen har vi bestræbt os på at følge de tre normalformer for at sikre database design, der reducerer redundans og bevarer dataintegritet. Herunder beskrives, hvordan vi har arbejdet med hver normalform:

Første normalform (1NF):

Vi har sikret, at alle tabeller overholder 1NF ved at sørge for, at:

- Alle attributværdier er atomiske (Ingen lister eller sammensatte værdier i et felt)
- Hver kolonne indeholder værdier af samme datatype
- Hver række er entydig identificeret via en primærnøgle

Et eksempel på dette kan ses på vores Employee tabel. Tabellen indeholder en række pr. medarbejder, og kompetencer opbevares i en separat tabel for at undgå flere værdier i samme felt.

Anden normalform (2NF):

Alle tabeller, der har sammensatte primærnøgler, er designet sådan, at alle ikke-nøgle attributter afhænger af hele primærnøglen og ikke kun en del af den. I tilfælde hvor dette ikke kunne opfyldes har vi valgt følgende:

- Opdele tabellen i to separate tabeller med relation via en fremmednøgle
- Fx i forbindelse med mange-til mange relationer mellem "employee" og "skill".

Tredje normalform (3NF):

For at opnå 3NF har vi sikret, at:

- Der er ingen nøgle attribut der afhænger af en anden nøgle-attribut
- Eventuelle afhængige oplysninger (som fx rollenavn) er placeret i separate tabeller (role), der refereres via en foreign key.

Denne struktur minimerer data dublering og gør databasen skalerbar samt lettere at vedligeholde.

Vi har bestræbt os på at følge de tre normalformer i hele databasen, men der findes en enkelt undtagelse. Tabellen "awaitingEmployee" overholder ikke fuldt ud 3. NF. Tabellen er designet til midlertidigt at opbevare brugere, der er oprettet, men som endnu ikke er aktiveret i systemet. Strukturen indeholder enkelte felter, der ikke direkte afhænger af primærnøglen, men snarere af hinanden hvilket bryder principperne for normalformerne.

Vi har dog valgt at beholde tabellen i denne form af følgende årsager:

- Tabellen er midlertidig og isoleret fra øvrige datamodeller og bruges kun i forbindelse med førstegangs oprettelse og aktivering.
- Den fungerer som en buffer og slettes, når brugeren er aktiveret i systemet.
- Ændringer eller redundans i tabellen har ingen indflydelse på kernefunktionen for det øvrige system.

SQL Scripts

I opbygningen af vores database har vi haft stor fokus på dataintegritet og performance. Dette har vi sikret gennem et velstruktureret SQL-script, som opretter tabeller, relationer og constraints på tværs af databasen.

Vi har valgt at benytte int som primærnøgle i alle vores tabeller, og alle disse primærnøgler er autogenereret med "AUTO_INCREMENT". Dette sikrer at hver række i en tabel får en unik id og samtidigt frifanger brugeren for manuelt at skulle holde styr på id'er samt gør systemet skalerbart.

```
CREATE TABLE employee (
    employeeID INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT UNIQUE,
    employee_Name VARCHAR(100) NOT NULL,
    employee_email VARCHAR(100) NOT NULL UNIQUE,
    employee_username VARCHAR(100) NOT NULL UNIQUE,
    employee_password VARCHAR(256) NOT NULL,
    roleID INTEGER,
    FOREIGN KEY (roleID) REFERENCES roles(roleID) ON DELETE SET NULL
);
```

Figur 2) Udsnit af employee tabel fra databasen

Som det ses figur 2 af SQL-scriptet ovenfor, har vi i employee-tabellen sat "NOT NULL" constraints på alle kolonner som email, username og password. Det skyldes, at disse oplysninger er vigtige i forhold til at en bruger kan login korrekt. Samtidigt har vi gjort at employee_email og employee_username skal være unikke, så vi undgår dobbeltopprettelser og brugernavnekonflikter i systemet.

```
CREATE TABLE projectAssignees
(
    projectID INTEGER,
    employeeID INTEGER,
    PRIMARY KEY (projectID, employeeID),
    FOREIGN KEY (projectID) REFERENCES project (projectID) ON DELETE CASCADE,
    FOREIGN KEY (employeeID) REFERENCES employee (employeeID) ON DELETE CASCADE
);
```

Figur 3) Udsnit af projectAssignees tabel fra databasen

Flere af relationerne i systemet er mange til mange relationer. Eksempelvis mellem employee og project, task og subtask. For at håndtere dette har vi oprettet tabeller som projectAssignees (figur 3), taskAssignees og subtaskAssignees, hvor en kombination af to fremmednøgler udgør en sammensat primærnøgle. Dette betyder at hver række er unik og det sikrer, at en medarbejder ikke kan tildeles samme opgave flere gange.

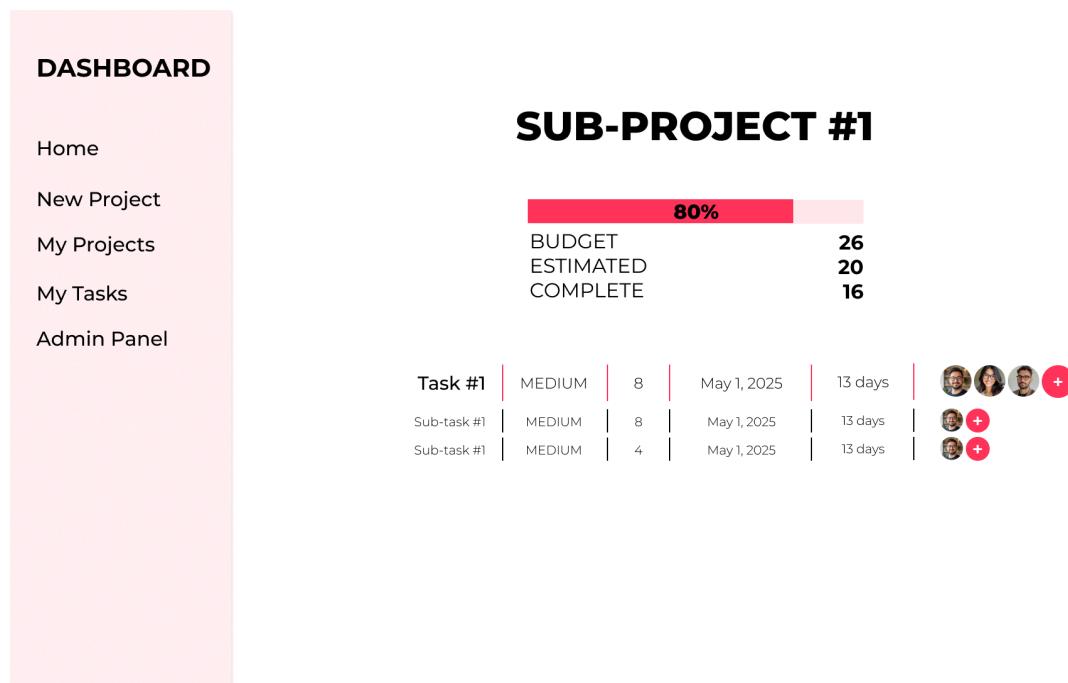
Vi har også lavet tabellen skillRelation, som håndterer koblingen mellem employee og skill, da en medarbejder kan have nul til mange kompetencer. AwaitingEmployee er en særskilt tabel til brugere, som endnu ikke er oprettet i systemet.

Alle relationer er lavet med “FOREIGN KEY” constraints og mange steder med “ON DELETE CASCADE”, hvilket sikrer at relaterede data slettes automatisk, hvis f.eks. et projekt slettes fra databasen.

Samlet er vores SQL-script lavet til at være robust, skalerbart og let forståeligt.

User interface design

Siden projektopgaven er stillet af Alpha Solutions, valgte vi at droppe en del af vores kreative frihed for at tage udgangspunkt i Alpha Solutions’ design. Vi startede med at besøge deres hjemmeside og tog elementer som farver, skriftyper, og andre noter af deres stilart som f.eks. at de bruger skarpe kanter, ingen skygger og stilken generelt er flad. Med disse observationer turde vi godt springe direkte ud i high-fidelity wireframing, hvilket gjorde at vi kunne kombinere user flow med et så tæt på som muligt, gennemført og færdigt designprodukt allerede fra start vha. designværktøjet Figma.



Figur 4) Skærmbillede over sub project

På figur 4 kan man se to af vores indledende og mest kreative UI idéer i Figma, som vi gerne ville vække til livs i systemet. Her er vi inde på et subproject, og dette subproject har en “progress bar” som man nemt kan se hvor langt man er med subprojectet, på en hurtig men også lidt tilfredsstillende måde.

Designet gør systemet lidt videospil agtigt og gør det lækkert at fuldføre tasks og se den her progress bar nå tættere på 100%. Udover dette havde vi dertilhørende “stats” til subprojectet. “Budget” som skulle være den indledende mængde udviklingstimer som er frigivet til projektet på et overfladisk niveau, inden tasks/subtasks osv. er blevet oprettet og estimeret. “Estimated” den samlede estimering for alle tasks/subtasks i projektet. “Complete”, den samlede mængde af disse estimeringer der har en status markeret som fuldført.

Den anden essentielle idé til vores UI, var vores task/subtask struktur. Altså, at en task kan have de her subtask “children” og dermed får vi et infinite scroll view, hvor man nemt kan se hver enkelt task, men også dens dertilhørende subtasks i samme view. På denne måde “sletter” vi et lag i vores 4-lags-system (Project -> subproject -> task -> subtask). Vi beholder den fulde kompleksitet og funktionalitet, men vi sparer brugerne for at skulle tilgå endnu et irriterende lag.

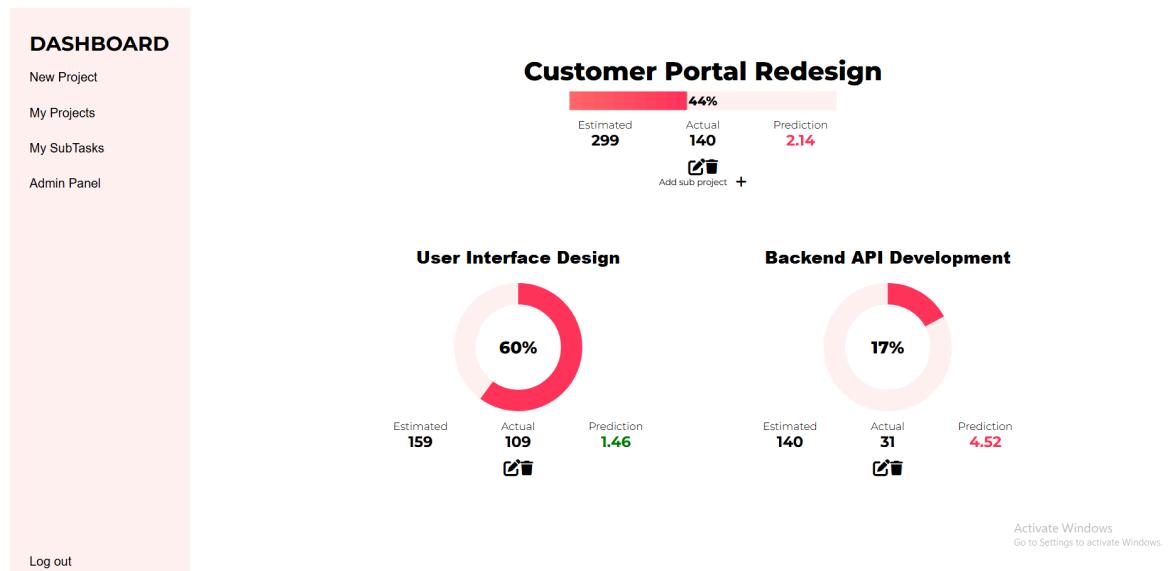
Vores første modvind:

Til et PO-møde fremviste vi vores produkt og gennemgik hele user flowet. PO, blev forvirret over de før tiltalte “stats” og hvad de stod for. Efter en længere snak, kom vi frem til at det her med et indledende og overfladisk “Budget” inden identificering af tasks faktisk ikke gav mening, men at man i praksis ville undersøge noget nærmere, inden man bare lægger et overfladisk budget. Derefter kom vi frem til nye key values: “Estimated”, “Actual” og derefter “Prediction”, som er forholdet mellem de to første værdier, altså et billede af hvor præcis man er i sine estimeringer kontra de egentlige ressourcer brugt. Dette betød også, at vi nu tiltænkte hele systemet anderledes. Førhen havde hvert et lag i databasen en “estimate” og “status” kolonne. Men med den nye “bottom-up” tankegang ville vi kun behøve “status” og “estimate” i vores subtask table, og blot kalkulere og vise en tasks “status” og “estimate” baseret på dets subtasks inde i vores kode. Denne logik følger vi hele vejen op i lagene.

```
public List<SubTask> getAllSubtasksByProjectId(int projectid) { 6 usages  ↗ viictv
    String sql = """
        SELECT subtask.* FROM subtask
        JOIN task ON subtask.taskID = task.taskID
        JOIN subproject ON task.subProjectId = subproject.subprojectId
        JOIN project ON subproject.projectID = project.projectID
        WHERE project.projectID = ?;
    """;
    return jdbcTemplate.query(sql, new SubTaskRowMapper(), projectid);
}
```

Figur 5) Skærmbillede af metode i repository

Figur 5 viser hvordan man med en nested join sql query kan få alle subtasks hørende til et specifikt projekt. Denne logik bruger vi til at udregne et projekts stats og progress bar på frontenden.



Figur 6) Endpoint af et specifikt projekt, (/project/{id})

Figur 6 er et skærbillede af det endelige produkt, hvor vi kan se de nye designprincipper taget i brug. Figuren viser endpointet “/project/{id}” og her kan man se det specifikke projekt i toppen og dets subprojects nedenunder med cirkulære progress bars i stedet, for at vise at man altså kigger på subprojects og ikke alm. projects.

Designprincipper:

Under designudviklingen, har vi nok kørt mest på intuition, men med Nielsen og Molichs 10 UI Guidelines (Interaction Design Foundation, n.d) og Gestalt principperne (Levende Streg, n.d) i baghovedet. F.eks. “Error prevention” princippet som har til formål at undgå at slutbruger ender ud i en opgave hvor de selv skal problemløse for at finde en fejl i programmet som de har forårsaget. Dette håndterer vi nogenlunde vha. bl.a. “required” html-attributter i vores create endpoints, som set i figur 7 nedenunder.

```
<textarea placeholder="Project Name" th:field="*{ projectName }" class="inputName" rows="2" maxlength="100" required></textarea>
```

Figur 7) Skærbillede af HTML-kode som eksisterer i et <form> tag

Sådan opnår vi en nem måde at fortælle brugeren, at de mangler at udfylde et felt hvis de f.eks. prøver at lave et projekt uden et navn. Browseren håndterer alt designet af denne lille advarsel og er derfor et nemt og effektivt valg. Udseende kan ses på figur 8.

Project Name

Please fill out this field.

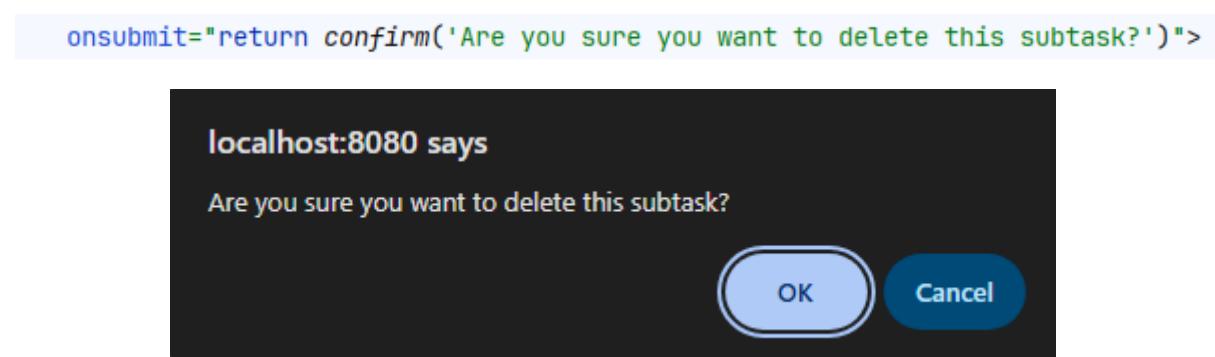
Project Description

Start Date End Date

Create

Figur 8) Skærmbillede af "/projects/new" endpoint, som viser en advarselsbesked.

Siden at brugeren kan komme til at klikke forkert diverse steder og at slet-knappen derfor er kritisk udsat, dobbeltjekker vi med brugeren når vedkommende prøver at slette en entitet. Dette er også med til at øge error prevention for brugeren. Især er det vigtigt fordi vi ikke har implementeret nogen form for recovery funktion når entiteter slettes, som f.eks. en papirkurv- eller arkivering feature hvor man kan få lov at gendanne sine entiteter. På figur 9 kan man se at vi har brugt en "confirm"-type efterfulgt af en selvbestemt advarselsbesked i vores onsubmit html-attribut inde i vores <form> tag. Igen en nem måde at få browseren til at arbejde for os.



Figur 9) Bekræftelsesboks når man trykker på sletknappen.

Udover dette har vi prøvet at holde et konsistent og minimalistisk design hele vejen igennem, bl.a. via brug af få farver og figurer, tal og ikoner frem for tekst hvor muligt.

På baggrund af vores analyse og designovervejelser bevæger vi os nu videre til implementeringsfasen, hvor vi omsætter vores modeller og arkitektur til konkret funktionalitet i systemet.

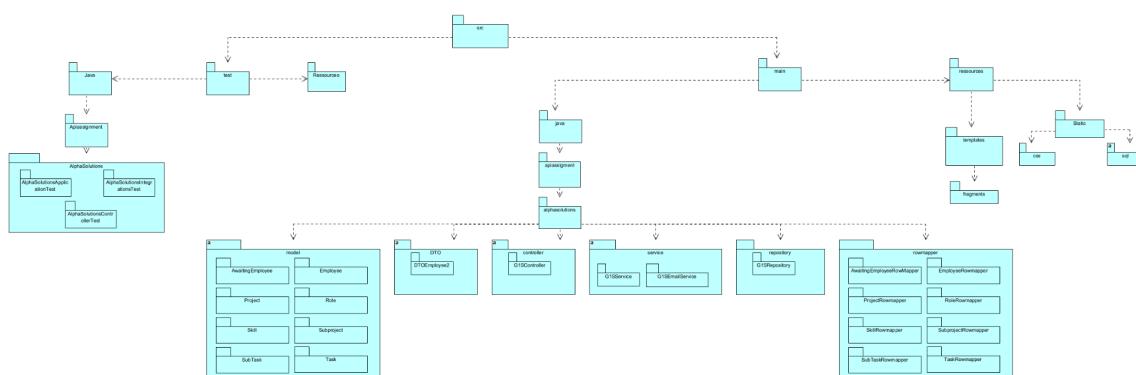
Implementering

I dette afsnit beskrives implementeringen af det udviklede system, herunder hvordan vi har omsat krav og design til konkret kode. Implementeringen er baseret på Spring og struktureret efter MVC-arkitekturen.

Vi har arbejdet ud fra en agil udviklingsproces med løbende integration og afprøvning, hvor vi har prioriteret funktionalitet ud fra vores MVP og backlog. Under implementeringen har vi anvendt en række designprincipper og mønstre for at sikre høj kvalitet, genanvendelighed og vedligeholdelse.

Afsnittet indeholder blandt andet vores arkitektur design, aktivitetsdiagram, klassediagram samt udvalgte kodestumper, der viser løsninger og overvejelser i systemets opbygning

Arkitektur/UML package diagram



Figur 10) UML package diagram over systemets arkitektur

Figur 10 viser den overordnede arkitektur for vores program. Pakkeopdelingen følger en klassisk Spring Boot struktur baseret på MVC arkitekturen. Vi har inddelt systemet i logiske lag: controller, service, repository og model, hvilket sikrer en tydelig adskillelse af ansvar. Denne struktur gør systemet nemmere at teste, vedligeholde og videreudvikle.

Desuden har vi samlet alle Rowmapper-klasserne i en separat pakke for at fremme genbrug og overblik ved brug af JDBC Template til manuel database-mapping.

Vi har struktureret systemet i tre overordnede lag:

- Præsentationslaget (View + Controller)

Dette lag udgør grænsefladen mod brugerne og omfatter HTML-sider med Thymeleaf samt controller med endpoints. Her håndteres bl.a. navigation og rettighedsstyring, så brugerne kun kan tilgå sider og funktioner afhængigt af deres rolle.

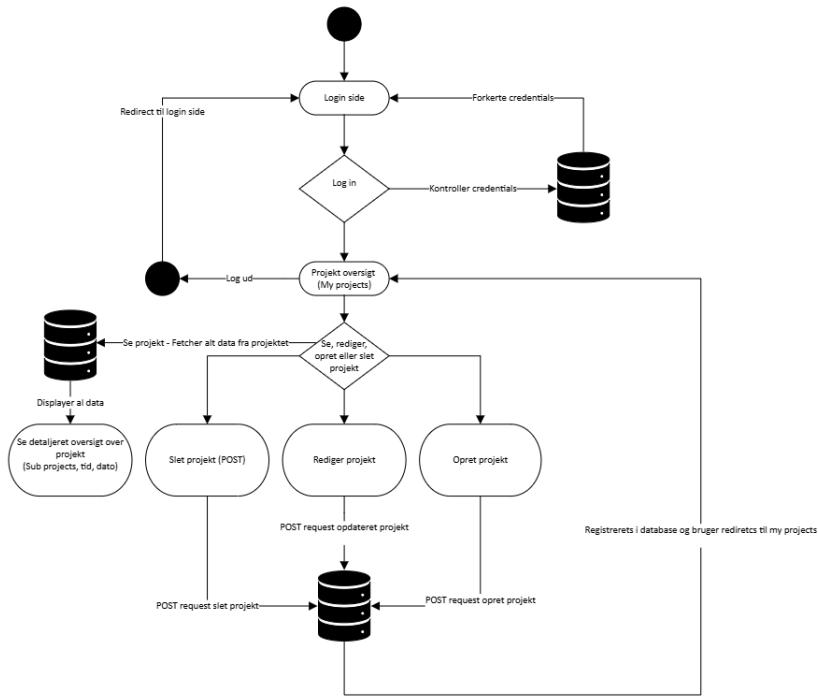
- Domænelag (Service + model)

Domænelaget indeholder forretningsmæssig logik og domæne struktur. Her ligger service klasserne som håndterer behandling og validering af data, samt modelklasser, som repræsenterer systemets entiteter.

- Data access lag (Repository)

Dette lag står for interaktionen med databasen og indeholder repository-klasser, som bruger SQL og JDBC template til at hente, indsætte, opdatere og slette data. Resultaterne sendes videre til service-laget, som bearbejder og sender dem videre til frontend.

Aktivitetsdiagram 1



Figur 11) Aktivitetsdiagram funktionalitet over CRUD funktion

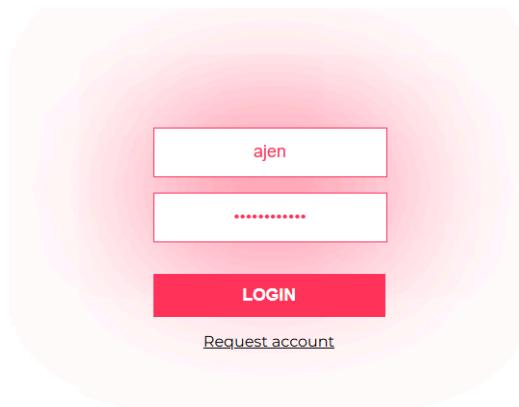
Figur 11 viser navigationflowet og de processer der foregår, når en bruger logger ind og tilgår sin projektoversigt. Når brugeren logger ind med korrekte loginoplysninger kontrolleres disse med databasen. Herefter bliver brugeren omdirigeret til ”My Projects”, hvor listen af projekter, som brugeren er tilknyttet, bliver hentet og vist.

Brugeren kan herefter se detaljer for et specifikt projekt og tilgå de underliggende subprojekter, tasks og subtasks. Hvis brugeren har rollen som projektleder eller admin, så har brugeren mulighed for at oprette, slette og redigere et projekt. Dette ses på aktivitetsdiagrammet, hvor der ved POST-requests til databasen, enten oprettes, slettes eller opdateres et projekt.

Efterfølgende bliver brugeren automatisk sendt tilbage til produktoversigten med den opdaterede liste. En mere visualiseret vejledning kan findes i næste afsnit.

Navigationsflow

Den første side man ser i systemet er vores loginside. Vi går ud fra at man har en konto man kan logge ind med:



Figur 12) Login side

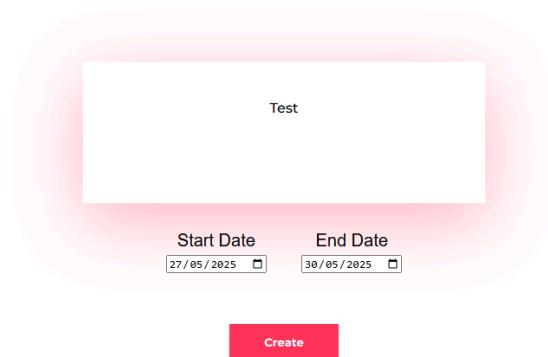
A screenshot of a dashboard landing page. On the left, there is a sidebar with a pink header containing the text "DASHBOARD". Below the header are four menu items: "New Project", "My Projects", "My SubTasks", and "Admin Panel". The main content area has a title "Customer Portal Redesign" and a progress bar showing "44%". Below the progress bar, there are three numerical values: "Estimated 299", "Actual 140", and "Prediction 2.14". There is also a small icon of a person with a checkmark and the word "assigned". At the bottom of this section, there are five colored circles with abbreviations: "A", "MN", "LP", "MA", and "+4", followed by a link "add more...".

Figur 13) Vores landing page efter login er vores "My projects" side. Her kan vi se vores nuværende projekter

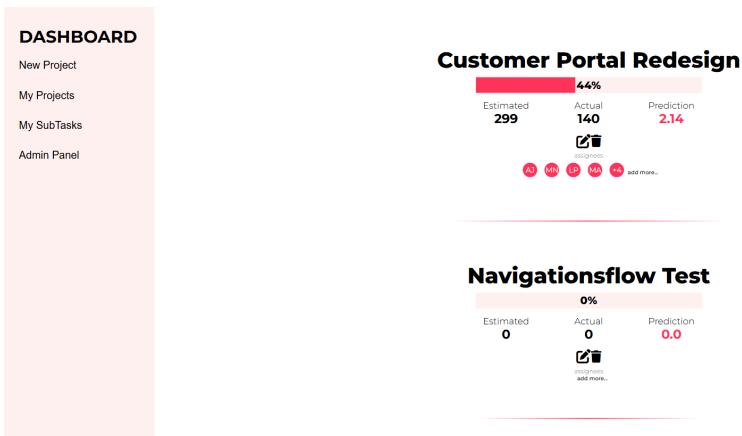
A screenshot of a "New Project" creation form. The title "Project Name" is at the top in a large, bold, dark font. Below it is a large input field with the placeholder "Project Description". At the bottom of the form are two date input fields: "Start Date" and "End Date", each with a date picker icon. A red "Create" button is located at the very bottom center of the form.

Figur 14) Vi trykker "New Project" i Dashboardet og kommer til denne side.

Navigationsflow Test



Figur 15) Felterne udfyldes og vi trykker "Create":



Figur 16) Vi bliver nu sendt tilbage til "My Projects" og kan se vores nyoprettede projekt på listen.

Herfra kan vi tilgå vores UPDATE og DELETE funktionaliteter. Trykker vi på det lille blyant ikon på vores nyoprettede projekt, sendes vi ind i vores UPDATE view. Her kan vi opdatere alle værdierne, men også tilføje assignees til projektet:

Navigationsflow Test



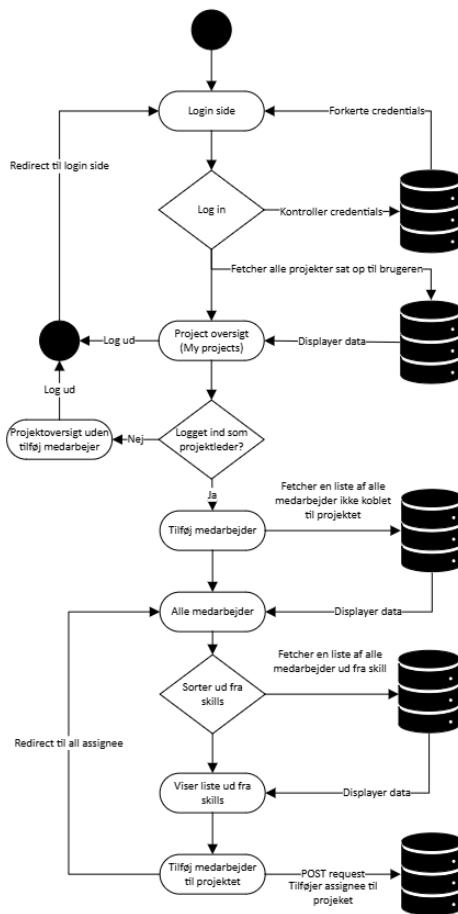
Figur 17) /projects/edit/{id} endpoint

Navigationsflow Test

The screenshot shows a navigation flow test interface for editing a project. It features a large central text area labeled "Test". Below this are two date input fields: "Start Date" (27/05/2025) and "End Date" (30/05/2025). Underneath the dates is a "Assignees" section with a plus icon and the text "add more...". At the bottom right is a red "Save" button.

Figur 18) /projects/edit/{id} endpoint

Aktivitetsdiagram 2



Figur 19) Aktivitetsdiagram for at tilføje medarbejder til projekt

Figur 19 viser navigationflowet og de processer der foregår, når en bruger logger ind og ønsker at tilføje en medarbejder til et projekt. Brugeren starter med at logge ind, hvorefter loginoplysningerne valideres. Hvis disse er korrekte, så bliver brugeren logget ind og en liste af projekter, som brugeren er tilknyttet, bliver hentet fra databasen. Disse projekter bliver vist på siden "My Projects".

Systemet kigger nu på brugerens rolle. Hvis rollen er projektleder eller admin, så har brugeren nu mulighed for at tilføje medarbejdere til et projekt. Hvis brugeren ønsker at tilføje medarbejdere til et projekt, så vises en liste af medarbejdere, som ikke allerede er tilføjet til projektet. Listen kan filtreres ud fra ønskede kompetencer, hvor systemet henter en ny liste af medarbejdere ud fra den valgte kompetence.

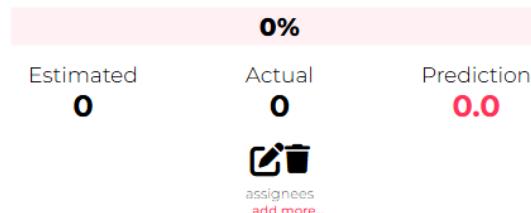
Brugeren kan tilføje den eller de ønskede medarbejdere til projektet via en POST-request.

Systemet opdaterer nu databasen og medarbejder/ene er nu tilføjet til projektet. En mere visualiseret vejledning kan findes i næste afsnit.

Navigationsflow

Flowet starter ud på samme måde som det forrige flow. Denne gang vil vi dog gerne tilgå medarbejdere og deres skills. På næste skærbillede kan vi se hvordan vi tilføjer medarbejdere til vores projekt ved at klikke "add more" highlighted med rød:

Navigationsflow Test



Figur 20) Navigationsflow test

Herefter kommer vi til en ny side, som viser alle medarbejdere som ikke på nuværende tidspunkt er tilføjet til projektet. Herinde kan vi også vælge at sortere i listen af medarbejdere ud fra deres kompetencer. Hvis man f.eks. skal bruge en backend developer, kan man nemt filtrere efter det:

Select a Skill						
Employee ID	Name	Email	Skills	Edit Panel		
2	Marie Nielsen	marie.nielsen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	mnie	2	<ul style="list-style-type: none"> Frontend Java Project Management Add
3	Peter Hansen	peter.hansen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	phan	1	<ul style="list-style-type: none"> Frontend Backend Java Add
4	Louise Pedersen	louise.pedersen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	lped	1	<ul style="list-style-type: none"> Python SQL JavaScript React Add
5	Mikkel Andersen	mikkel.andersen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	mand	1	<ul style="list-style-type: none"> Angular Add
6	Sofie Christensen	sofie.christensen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	schr	1	<ul style="list-style-type: none"> Project Management Database Design Add
7	Jacob Larsen	jacob.larsen@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	jlar	2	<ul style="list-style-type: none"> Cloud Architecture Add
8	Emma Schmidt	emma.schmidt@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	esch	2	<ul style="list-style-type: none"> Python SQL JavaScript Add
9	Frederik Møller	frederik.moller@alphasolutions.dk	Frontend Backend Java Python SQL JavaScript React Angular UI/UX Design Testing DevOps Cloud Architecture Project Management Agile Methodologies	fmol	2	<ul style="list-style-type: none"> UI/UX Design Testing Agile Methodologies Add

Figur 21) Sortering af medarbejder

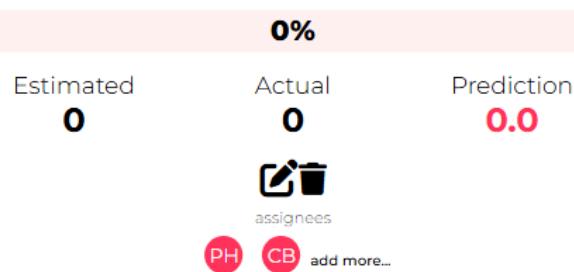
Hermed får vi backend medarbejderne "Peter Hansen" og "Christian Berg". Vi tilføjer begge til projektet:

Employee ID	Name	Email	Username	Role ID	Skills	Edit Panel
3	Peter Hansen	peter.hansen@alphasolutions.dk	phan	1	<ul style="list-style-type: none"> Frontend Backend Java 	<button>Add</button>
11	Christian Berg	christian.berg@alphasolutions.dk	cber	1	<ul style="list-style-type: none"> Frontend Backend 	<button>Add</button>

Figur 22) Tilføjelse af medarbejder

Nu kan vi se på de to små avatars ude i vores "My Projects" view, at begge personer er blevet tilføjet til projektet:

Navigationsflow Test



Figur 23) Tilføjet medarbejdere til projektet

Klassediagram

Efter arbejdet med domænemodellen, ER-modellen og aktivitetsdiagram har vi gennem scrumprocessen udarbejdet et klassediagram, som visualiserer og illustrerer relationer mellem klasser, deres attributter og metoder.

Vores klassediagram er opbygget ud fra MVC-arkitekturen, hvor controller-, service-, repository- og modelklasser er adskilt for at opnå en høj grad af struktur og genanvendelighed. Dette understøtter vores ide om at programmet skal være vedligeholdelsesvenligt og nemt at videreudvikle.

Klassediagrammet viser tydeligt, hvordan entiteter som "project", "subproject", "task", "subtask" og "employee" hænger sammen. Eksempelvis har "subtask" et taskId fra "task" klassen, og "task" har en subprojectId fra "subproject" klassen, og til sidst har "subproject" en

projectId fra "project" klassen. Hver af disse klasser kan have en eller flere medarbejdere tilknyttet, hvilket er repræsenteret med en liste af "employee" som en attribut. Udover har klasserne som task, subproject og project en liste af "subtask", som anvendes til at beregne det samlede timeforbrug.

Klasserne "role" og "skill" er koblet til klassen "employee", hvilket gør at brugere kan defineres med en specifik rolle og en række af kompetencer. Vi har valgt at lade rowmapperklasser håndtere mappet mellem SQL og Java-objekter, hvilket holder vores repositories rene og overskuelige. Til sidst har vi de centrale klasser som "G1SController", "G1SService" og "G1SRepository", som repræsenterer flowet i programmet og adskiller datatilgang fra forretningslogik. Dette er gjort for at opnå en tydelig ansvarsfordeling og gøre systemet testbar og skalerbar.

Samlet giver vores klassediagram et klart overblik over systemets arkitektur. Det har været et vigtigt og godt redskab til at omdanne vores databasedesign og domænemodel til en funktionel Java-kode. Hvis vi skulle lave det om, så ville vi dog overveje at opdele vores controller-, service- og repositoryklassen i flere domæneområder. På den måde ville klassediagrammet også blive opdelt i flere klassediagrammer. Det ville gøre klassediagrammet mere overskueligt og afspejle en tydeligere struktur i applikationen.

Designovervejelser og principper

I udviklingen af AlphaSolutions kalkulationsværktøj har vi haft fokus på at skabe en struktureret, skalerbar og vedligeholdesesvenlig kode. For at opnå dette har vi gjort brug af forskellige principper inden for programmering som MVC, GRASP, Dependency Injection og Single Responsibility Principle.

MVC-arkitektur

Projektet er bygget op med en MVC-struktur (Model-View-Controller), som er afspejlet i vores pakkestruktur. Pakkestrukturen består blandt andet af en controller, service, repository og model. Denne opdeling sikrer ansvar og muliggør at frontend og backend kan udvikles uafhængigt af hinanden. Vores G1SController håndterer HTTP-requests, og kommunikere videre med servicelaget, som kalder den eller de relevante metoder fra vores repository. Herefter er det repositoriets ansvar at hente data fra databasen. Data som repositoriet

henter fra databasen bliver koblet på modelklasser som Employee, Project, Task og SubTask.

Dependency Injection og Inversion og Control

Da vi anvender Spring Boot til at bygge vores applikation, så har vi mulighed for at gøre brug af den understøttede Dependency Injection som den tilbyder. Det kan ses i vores G1SController, hvor G1SService bliver dependency injected direkte i konstruktøren. På den måde er controlleren ikke direkte afhængig af servicelaget og kan testes med mockobjekter. Ved hjælp af annoteringer som @Service, @Repository og @Controller fortæller vi Spring Boot, at disse klasser er vigtige, hvor den automatisk laver en instans af objektet og gemmer det i sin IoC-container, så vi senere kan bruge den instans andre steder i programmet. Disse instanser kaldes i Spring for Beans. Ved at uddeletere instanshåndtering til Spring, så følges principippet om Inversion of Control.

GRASP

Vi har designet systemet med GRASP-principperne i tankerne (General Responsibility Assignment Software Patterns). Særligt har vi haft fokus på Information Expert og Controller. Eksempelvis håndterer G1SRepository alt vores datalogik, hvor G1SService har ansvaret for den forretningsmæssige logik. Dette sikrer at ansvaret er fordelt til de klasser, som har mest viden om den respektive operation.

I vores projekt har vi forsøgt at efterleve principperne om low coupling og high cohesion. Disse principper har til formål at sikre, at applikationens klasser er flexible og nemme at vedligeholde, hvilket er vigtigt i større systemer med flere funktionaliteter.

Low Coupling

Low coupling refererer til graden af afhængighed mellem klasser. Hvis der er en lav coupling, så har ændringer på en klasse mindre indflydelse på andre klasser. Dette har vi forsøgt at opnå i vores system ved at adskille controlleren og repository via et servicelag. På den måde er controlleren ikke direkte afhængig af datalaget, hvilket gør det muligt at lave ændringer i repository-implementeringen uden at det påvirker controlleren og den logik som findes deri.

En mulig forbedring kunne være at opdele serviceklassen i mindre domæneopdelte services. Det ville mindske afhængighederne endnu mere og styrke princippet om low coupling.

High Cohesion

High cohesion refererer til graden af samhørighed mellem de elementer, der findes inden for samme klasse. Klasser har kun metoder i sig, som relaterer til klassen eget ansvarsområde, hvilket gør koden mere overskuelig, lettere at forstå og nemmere at vedligeholde. I vores system har vi forsøgt at opnå high cohesion ved at adskille ansvar i forskellige lag og klasser. Controlleren håndterer udelukkende HTTP-requests, hvor databehandling og forretningslogik er samlet i Serviceklassen, og direkte databaseoperationer foregår i repository.

Dog kan man argumentere for, at vores serviceklasse (G1SService) indeholder for mange forskellige ansvarsområder, f.eks. project-, employee-, task- og subtaskhåndtering. En mulig forbedring kunne derfor være at opdele serviceklassen i flere services, f.eks. ProjectService, EmployeeService, TaskService, SubtaskService osv.. Dette ville øge cohesion yderligere ved at gøre hver klasse endnu mere fokuseret på et bestemt domæneområde.

Kodeeksempler og beskrivelser

Følgende afsnit vil tage udgangspunkt i de givne metoder, hvor vi har brugt ekstra meget tid og energi på. Afsnittet vil inddrage relevante kodeeksempler, og der vil blive forklaret, hvordan vi som gruppe har reflekteret over, hvordan man laver effektiv kode. Dette har ført til at vi enkelte gange er nødsaget til at gå tilbage og refaktorere vores kode, så den var mere skalerbar, læsbar og bedre performances mæssigt.

Login:

En essentiel funktion i vores program var, at alle medarbejdere, uanset rolle, skulle kunne logge ind i systemet og have en profil. Vi var i starten i tvivl, om alle medarbejdere skulle kunne oprette en profil i systemet, men kom herefter frem til, at det kun skulle være projektledere og admins, som kunne oprette nye brugere, da det systemet kun skulle bruges af Alpha Solutions, og de ønskede ikke en masse ukendte brugere i deres system.

Derfor lavede vi et simpelt login system, hvor vi havde et GET endpoint der viste html siden "login", som var en login formular, og så havde vi et POST endpoint, som tjekkede den data

brugeren prøvede at logge ind med, altså brugernavn og kode. Følgende GET og POST endpoints kan ses på nedenstående figur.

```

211
212     @GetMapping(@RequestMapping("/login"))
213     public String login(){
214         return "login";
215     }
216
217
218     @PostMapping(@RequestMapping("/login"))
219     public String checkLogin(@RequestParam("checkUsername") String username, @RequestParam("checkUserpassword") String password,
220                             HttpSession session, Model model){
221         Employee employee = gISService.findByUsername(username);
222
223         if(employee == null || !gISService.verifyPassword(password, employee.getEmployeePassword()))
224             model.addAttribute(attributeName: "wrongLogin", attributeValue: true);
225         return "login";
226     }
227     session.setAttribute(s: "employee", employee);
228     session.setMaxInactiveInterval(3600);
229     return "redirect:/projects";
230 }
```

Figur 24) "login" GET og POST endpoint i controlleren

Som det ses på skærmbilledet fra vores program, så tager vi to RequestParam med, når brugeren prøver at logge ind, det er brugernavn og kode. Vi bruger først brugernavn til at kalde metode "findByUsername", dette er en metode i vores serviceklasse, som kalder en metode i vores repository med samme navn. Metoden ser ud som følgende:

```

586
587     public Employee findByUsername(String username) {
588         try {
589             String sql = "SELECT * FROM employee WHERE employee_username = ?";
590             Employee emp = jdbcTemplate.queryForObject(sql, new EmployeeRowMapper(), username);
591             return emp;
592         } catch (EmptyResultDataAccessException e) {
593             return null;
594         }
595     }
```

Figur 25) Metoden "findByUsername" i vores repository, har username som parameter og finder så en bruger ved hjælp af dette username.

Ovenstående metode gør brug af en simpel DQL(Data Query Language) statement, altså en "SELECT" statement, hvor den henter den ønskede data fra databasen. Vi ønsker i dette tilfælde at finde en bruger med et givent brugernavn. Vi har defineret brugernavne (Usernames), som unikke i vores database, og vi kan derfor være sikker på, at der enten kom én eller ingen brugere tilbage fra databasen. I tilfælde af at vi ikke kan finde nogle brugere, så vil vores metode returnere null.

Som vi så kan se på figur 24, linje 226, så har vi en "IF-statement" der tjekker, hvorledes "employee", altså vores bruger, er null, hvis den er null, så har den ikke kunnet finde en

bruger med det givne brugernavn. Dette vil resultere i at personen bliver redirected tilbage til login, og der vil komme en besked op, som fortæller brugeren “wrong username or password”.

I samme “IF-statement” på linje 226, der tjekker vi også, om brugerens password stemmer overens med det brugernavn, som blev tjekket for. Her bruges metoden “verifyPassword”, som er en metode der ligger i vores serviceklasse.

```
432
433
434     public boolean verifyPassword(String password, String encrypted) { 3 usages  ± Zahaawii *
435         return BCrypt.checkpw(password, encrypted);
436     }
437
```

Figur 26) Metoden “verifyPassword” er en metode i serviceklassen, som benytter sig af en hashing algoritme kaldet for BCrypt

Som det kan ses på metoden, så er det en boolean, så den vil returnere true eller false, hvorledes brugerens indtastede kode stemmer overens med den hashede kode, som befinner sig i databasen. Vi benytter os af JBCrypts dependency, som har forskellige metoder heriblandt “checkpw”.

Disse metoder kommer fra biblioteket “jbcrypt”, som indeholder “BCrypt” algoritmen og som har alle de metoder vi gør brug af.

Den indeholder også metoden “hashpw”, som er en metode, der hasher den kode brugeren indtaster, når der bliver oprettet en ny profil eller opdateret en profil. Vores database er dermed mere sikker, da alle adgangskoder er hashede med BCrypt.

Når brugeren prøver at logge ind, og vores “verifyPassword” metode bliver kaldt, så indsætter den to parametre i metoden “checkpw”. Disse to er den kode, som brugeren har indtastet på login siden, og det er den hashede kode, som ligger i databasen og som hører til det brugernavn, brugeren lige har indtastet. Metoden “checkpw” hasher så den indtastede kode, og hvis de to hashede koder nu stemmer overens, så returnerer den true, hvilket betyder at brugeren har indtastet den rigtige kode.

Systemet har nu tjekket personens log-in og gemmer vedkommendes profil i sessionen. Der bliver dermed gemt et “employee” objekt i session, dette objekt indeholder “employeeld”, “employeeName”, “employeeEmail”, “employeeUsername”, “employeePassword” og “roleld”.

De sidste funktioner der sker i vores endpoint er, at vi sætter sessionen til at have en "max Inactive Interval" på 3600 sekunder, dette betyder at, hvis brugeren er inaktiv i over 1 time, så vil "employee" bliver fjernet fra brugerens session, og de skal derfor logge ind igen.

Hherefter redireceter vi personen til "/projects", som er overblik over vedkommendes projekter.

Sort by skills:

I dialog med vores PO, så blev der lagt stor vægt på, at systemet skulle være overskueligt og skalerbart. Med dette i mente, gik vi i tænkeboksen, og fandt frem til, at når man skulle tilføje en medarbejder til et projekt, så ville det være en klar fordel, hvis man kunne se en overskuelig liste over alle de medarbejdere, som ikke allerede var tilføjet til projektet.

Derudover ville det være en fordel, hvis man kunne sortere i sine medarbejdere ud fra, hvilke skills/kompetencer den givne medarbejder havde. Vi skitserede den givne idé i form af en user story med tilhørende tasks og fremlagde det for PO, som var overbevist om, at det ville være en god ide. Da vi så gik i gang med at programmere vores GET endpoint, hvor vi ville have en liste over dem, som ikke var i projektet, men havde en given skill, så stødte vi på et problem, som vi ikke var forberedt på.

Idet vi hentede alle vores mange skills fra databasen, så kunne den ikke finde vores endpoint, når vi filtrerede medarbejderne ud fra skill'en kaldet "UI/UX design". Dette skyldes at der var en skråstreg i den givne skill, og browseren så derfor URL'en som et nyt endpoint. Vi fik fikset problemet ved at lave et GET endpoint, som kunne URL-kode den givne skill, som man valgte. Følgende endpoint kan ses på nedenstående billede, hvor vi benytte "UriUtils.encode(skill, StandardCharsets.UTF_8);" til at kode URL'en, så den ikke tror vores RequestParam er en del af selve URL'en.

```

20   public class G1SController {
548     //nedenstående metode, modtager en pathvariable(projectId) og en requestParam(skill), den redirectes så en til
549     //project/projectId/assignees?skill=xxx" her bliver den valgte skill så URL-encoded, så den ikke forvirrer selve url'en
550     //dette er gjort, fordi den tidligere crashede, da man prøvede at tilgå "UI/UX", fordi den opfattede det som et nyt endpoint pga "/"
551     @GetMapping(@Path "/project/{projectId}/assignees/select-skill") huw02
552     public String redirectToSkill(
553       @PathVariable Long projectId,
554       @RequestParam String skill) {
555       return "redirect:/project/" + projectId + "/assignees?skill=" + UriUtils.encode(skill, StandardCharsets.UTF_8);
556     }
557
558     @GetMapping(@Path "/project/{projectId}/assignees")
559     public String getEmployees(
560       @PathVariable int projectId,
561       @RequestParam(required = false) String skill,
562       Model model, HttpSession session) {
563       if(!g1SService.isLoggedIn(session)){
564         return "redirect:/login";
565       }
566       List<Skill> skillList = g1SService.getAllSkills();
567       model.addAttribute("skills", skillList);
568       model.addAttribute("projectId", projectId);
569       List<Employee> employees; // laver en tom liste af employee
570       if (skill != null && !skill.isEmpty()) {
571         employees = g1SService.getAllEmployeesWithSkillNotPartOfProject(skill, projectId);
572         if (employees == null || employees.isEmpty()) {
573
574           model.addAttribute("noEmployee", true);
575           model.addAttribute("foundEmployee", false);
576           return "employeesWithSkill";
577         }
578       } else {
579         employees = g1SService.getAllEmployeesWithSkillNotPartOfProject(skill, projectId);
580         if (employees == null || employees.isEmpty()) {
581           model.addAttribute("notPartOfProject", true);
582           model.addAttribute("foundEmployee", false);
583           return "employeesWithSkill";
584         }
585       }
586       model.addAttribute("foundEmployee", true);
587       model.addAttribute("listOfEmployee", employees);
588       return "employeesWithSkill";
589     }
590   }

```

Figur 27) 2 GET Endpoints, generere en liste over dem der ikke er i et projekt, medarbejdere kan sorteres ud for en given skill, kommentarer er fjernet fra billedet grundet metodens størrelse.

Selve hoved GET endpointet, “project/{projectId}/assignees”, har en PathVariable(projectId), dette er id’et på det givne projekt, hvor man vil tilføje en medarbejder til. Derudover har det RequestParam(skill), som er den givne skill brugeren kan sortere medarbejderne ud fra. Som det kan ses står den også til at være “required = false”, dette er nødsaget, da det skal være muligt for brugeren at se en liste over alle dem der ikke er i projektet, hvor de så ikke er sorteret efter skills.

Det første IF-Statement i metoden, tjekker brugerens session, hvis brugeren ikke er logget ind, og dermed ikke er en bruger i den givne session, så bliver vedkommende redirected tilbage til “login” siden.

Efterfølgende får vi en liste af alle skills i database, dette gør vi ved at kalde metoden “getAllSkills” fra vores service klasse, som så kalder “getAllSkills”, fra vores repository klasse, den laver en query, der henter alle Skill objekter fra databasen. Dette gøres ved at lave en DQL (Data Query Language) statement, altså det vi kender som en “SELECT”

kommando fra databasen. Vi laver så en return statement, som kalder `jdbcTemplate`'s egen metode kaldet "query". "query" returnerer en liste af objekter som den henter fra databasen.

Vi definerer hvad vores query skal hente ved at lave vores egen "Rowmapper".

“Rowmapper” er en funktion som konverterer hver række i den givne tabel til objekter. Som det kan ses i nedenstående eksempel har vi defineret en “SkillRowMapper” som er en Java klasse, vi selv har defineret.

```
468     }
469     public List<Skill>getAllSkills(){ 1 usage  ↳ huw02
470         String sql = "SELECT * FROM skill";
471         return jdbcTemplate.query(sql, new SkillRowMapper());
472     }

```

Figur 28) "getAllSkills" er en metode i repository, som henter alle skills i databasen.

Vores "SkillRowMapper" ser sådan her ud:

```
1 package apiassignment.alphasolutions.rowmapper;
2
3 import apiassignment.alphasolutions.model.Skill;
4 import org.springframework.jdbc.core.RowMapper;
5
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8
9 public class SkillRowMapper implements RowMapper<Skill> { 5 usages ± huw02
10     @Override no usages ± huw02
11     public Skill mapRow(ResultSet rs, int rowNum) throws SQLException {
12         Skill skill = new Skill();
13
14         skill.setSkillId(rs.getInt( columnLabel: "skillID"));
15         skill.setSkillName(rs.getString( columnLabel: "skill_name"));
16
17         return skill;
18     }
19
20 }
```

Figur 29) "SkillsRowMapper" en rowmapper der hente en liste af "Skill" objekter"

Som det kan ses på ovenstående figur, så implementerer vi i vores klasse, det funktionelle interface kaldet "RowMapper<T>". Vi vælger så at override "RowMapper"s egen metode kaldet "mapRow", og definere hvilken slags data vi gerne vil hente fra vores rækker i databasen. Vi får her "skillID" og "skill_name", som er de to værdier vores "skill"-tabel har i databasen.

Vi kan nu vende tilbage til vores GET endpoint og se at der bliver brugt "model.addAttribute", samt kaldt et par if-statements. Som det ses på figur 27, så tilføjer vi vores liste af "skills", samt vores "projectId" til model. Model.addAttribute er en metode vi bruger til at sende data til vores frontend, altså de HTML sider, vi kan se på selve hjemmesiden.

Vi tilføjer skills, så vi kan filtrere mellem alle de skills, der findes i databasen, og de skills bliver så hentet ind i vores dropdown, hvor man vælger en skill at sortere efter. "projectId" bliver tilføjet til model, så vi kan bruge projektets id i den URL vi smider afsted, når man trykker på sorter ud for en skill. Vi har nu nogle if-statements der tjekker, om det skill input vi modtager fra brugeren, er null eller ej, hvis man ikke vælger at sortere ud fra skills, så vil det være null.

Uanset om det givne skill input fra brugeren er null eller ej, så vil den kalde den samme metode "getAllEmployeesWithSkillNotPartOfProject". Denne metode kalder service klassen, og service klasse kalder en metode i repository med samme navn. Da vi lavede metoden første gang, så lavede vi den på en måde, som vi senere fandt ud af, ikke var super optimal.

Metoden kan ses på nedenstående figur, og som det kan ses på metoden, så har vi to parametre til metoden, "String skill" og "int projectId". Metoden får en liste af alle dem der ikke er en del af projektet, og en liste af alle dem der har en given skills. Så iterere den igennem begge liste, sammenligner dem og laver til sidst en færdig liste, hvor brugerens skills bliver tildelt dem. Derudover tjekkede den også, at dem der blev tildelt listen ikke var ejeren af projektet, da vedkommende stod i tabellen "project", som ejeren af projektet.

```

270
271     public List<Employee> getAllEmployeesWithSkillNotPartOfProject(String skill, int projectId){ 2 usages ± huw02
272         return g1SRepository.getAllEmployeesWithSkillNotPartOfProject(skill, projectId);
273     }
274
275     public List<Employee> getEmployeeBySkillNotPartOfProject(String skill, int projectId) { no usages ± huw02
276     List<Employee> notPartOfProject = g1SRepository.getEmployeeNotPartOfProject(projectId);
277     Employee ownerOfProject = g1SRepository.getProjectOwner(projectId).getFirst();
278     if(notPartOfProject == null || notPartOfProject.isEmpty()){
279         return null;
280     }
281     List<Employee> bySkill = g1SRepository.getEmployeeBySkills(skill);
282     if(bySkill == null || bySkill.isEmpty()){
283         return null;
284     }
285     List<Employee> finishList = new ArrayList<>();
286     for (Employee i : bySkill) {
287         for (Employee b : notPartOfProject) {
288             if (i.getEmployeeId() == b.getEmployeeId()) {
289                 if (!finishList.contains(i)) {
290                     if(i.getEmployeeId() != ownerOfProject.getEmployeeId()) {
291                         //ovenstående if statement tjekker at ejeren af projektet ikke kommer op på listen
292                         List<Skills> skillList = g1SRepository.getSkillsForEmployee(i.getEmployeeId());
293                         i.setSkills(skillList); //finder alle de skills der hører til en person
294                         finishList.add(i);
295                     }
296                     } //finder alle dem med en given skill
297                     } //finder alle dem der ikke er en del af projektet
298                     } //sammenligner id på de to lister
299     } //hvis den tredje liste(finishList) ikke har objektet i sig,
300     //så finder den personens skills, sætter dem på og tilføjer dem til listen
301     //man får dermed en liste af folk, som har en given skill og som ikke allerede er tilføjet til projektet
302     if(finishList.isEmpty()){
303         return null;
304     }
305
306     return finishList;
307 }

```

Figur 30) Gammel metode i service, der fandt medarbejdere der ikke var en del af et projekt, men havde en skill. Denne metode blev udskiftet med en mere effektiv metode.

Metoderne for at hente alle der ikke var en del af projektet, alle med en given skill og ejeren af projektet, kan ses på nedenstående figur.

```

473
474     public List<Employee>getEmployeeBySkills(String skills){ 2 usages ± huw02
475         String sql = "SELECT employee.* from employee " +
476             "join skillrelation on employee.employeeID = skillrelation.employeeID " +
477             "join skill on skill.skillID = skillrelation.skillID " +
478             "where skill_name = ?";
479         List<Employee> listOfEmployees = jdbcTemplate.query(sql, new EmployeeRowMapper(), skills);
480         if(listOfEmployees.isEmpty()){
481             return null;
482         }
483         return listOfEmployees;
484     }
485
486     public List<Employee>getEmployeeNotPartOfProject(int projectId){ 2 usages ± huw02
487         String sql ="SELECT * FROM employee " +
488             "WHERE employeeID NOT IN (" +
489             " SELECT employeeID " +
490             " FROM projectassignees " +
491             " WHERE projectID = ? )";
492         return jdbcTemplate.query(sql, new EmployeeRowMapper(), projectId);
493     }
494     public List<Employee>getProjectOwner(int employeeId){ 2 usages ± huw02
495         String sql =
496             "SELECT employee.* from employee " +
497             "join project on project.employeeID = employee.employeeID " +
498             "where project.projectID = ?";
499         List<Employee>employeeList = jdbcTemplate.query(sql, new EmployeeRowMapper(), employeeId);
500         if(employeeList.isEmpty()){
501             return null;
502         }
503         return employeeList;
504     }
505

```

Figur 31) 3 metoder i repository, som blev brugt i den gamle metode, når man skulle have en liste over folk, som kunne sættes på projektet.

Som det kan ses på metoden, så var den ikke helt optimal, da det krævede 3 queries til databasen, og en masse iterationer og if-statements gennem listerne af medarbejdere. Derfor tænkte vi på, hvordan vi kunne optimere metoden, så den blev mere skalerbar og så den havde bedre performance. Vi lavede derfor en ny metode, som kunne håndtere alt logik i vores repository, og så skulle service bare kalde metoden, med dets parameter frem og tilbage mellem controller og repository-laget. Metoden kan ses på den nedenstående figur.

```

507     public List<Employee> getAllEmployeesWithSkillNotPartOfProject(String skill, int projectId) { 4 usages ± huw02
508         List<Employee> finalList = new ArrayList<>();
509         Employee projectOwner = getProjectOwner(projectId).getFirst(); //vi får ejeren af projektet fra tabelen project
510
511         String sqlStart = "SELECT e.* FROM employee e"; //vi vil gerne ende med en liste af employees
512
513         String sqlJoinSkill = "JOIN skillrelation sr ON e.employeeid = sr.employeeid" +
514             "JOIN skill s ON sr.skillid = s.skillid"; //vi laver joins mellem tabellerne employee, skillrelation og skill
515
516         String sqlNotInProject = "WHERE e.employeeid NOT IN (" +
517             " SELECT pa.employeeid FROM projectassignees pa WHERE pa.projectid = ?" + //her indsætter vi projectId
518             ")"; //Vi vil kun have de employees som ikke allerede er en del af projektet.
519         String sqlWithoutOwner = "AND e.employeeid != ?"; //her indsætter vi employeeid, så vi ikke får ejeren af projektet
520         String sqlFilterSkill = "AND s.skill_name = ?"; //her indsætter vi den givne skill, som vi sortere efter
521
522         String finalSql; //starter med at lave en tom string, som efterfølgende bliver udfyldt med vores sql query
523         Object[] insertIntoQuery; //lav et tom array af datatypen Object
524
525         if (skill == null || skill.isEmpty()) { //tjekker at skill er null eller bare en tom string
526             finalSql = sqlStart + sqlNotInProject + sqlWithoutOwner;
527             // sammensætter vores string queries, nu får vi en liste over alle dem der ikke er i projektet, vi har ikke sorteret efter skills
528             insertIntoQuery = new Object[] { projectId, projectOwner.getEmployeeId() };
529             //vi smider nu projectId og ejer af projektets id i en array
530         } else { //i nedenstående metode, har vi bare sorteret dem ud for skills, så vi fx får alle dem der kan have
531             finalSql = sqlStart + sqlJoinSkill + sqlNotInProject + sqlWithoutOwner + sqlFilterSkill;
532             insertIntoQuery = new Object[] { projectId, projectOwner.getEmployeeId(), skill };
533         } //nu kan vi så lave en liste af employees, enten kun dem med en given skill eller bare alle der ikke er en del af projektet
534         List<Employee> employeeList = jdbcTemplate.query(finalSql, new EmployeeRowMapper(), insertIntoQuery);
535         //til sidst iterere vi igennem vores liste og for hver person sætter vi deres skills på dem.
536         for (Employee e : employeeList) {
537             List<Skill> skills = getSkillsForEmployee(e.getEmployeeId());
538             e.setSkills(skills);
539             finalList.add(e);
540         }
541     return finalList;
542 }
543
544
545

```

Figur 32) Den optimerede metode, som henter en liste af medarbejdere, som ikke er en del af et projekt, og som kan filtreres ud fra en given skill.

Som figuren viser, så fik vi kortet metoden ned, så vi kunne få det hele i en metode, som lå i vores repository. Metoden gør brug af en del “String concatenation”, hvor vi sammensætter flere strings og laver det til en lang string. I vores tilfælde gør vi brug af det, da den query vi sender af sted afhænger af, hvorledes brugerinputet Skills er null eller ej. Hvis det er null, så laver vi en query, der laver en liste af medarbejdere, som ikke er en del af projektet, hvis det ikke er null, så bliver det kun dem, som har den givne skill.

Derudover har vi også et array af objekter, som er den data, vi skal indsætte i vores query. Hvis skill er null, så skal den ikke indsættes, så er det kun projektets id og ejeren af projektets id, der skal indsættes. Vi ender dermed med at få en liste af medarbejdere, enten sorteret med eller uden skill, og kan så iterere gennem listen af medarbejdere og hente alle deres skills fra databasen og sætte det på dem.

Hjemmesiden vil se sådan her ud, når der ikke er sorteret efter brugere:

The screenshot shows the 'Select a Skill' interface. On the left sidebar, there are links for Home, New Project, My Projects, and Admin Panel. The main area has a heading 'Select a Skill' with a dropdown menu set to 'All employees' and a 'Filter' button. Below this is a table with columns: Employee ID, Name, Email, Username, Role ID, Skills, and Edit Panel. The table contains 11 rows of employee data. The skills listed for each employee are as follows:

Employee ID	Name	Email	Username	Role ID	Skills	Edit Panel
2	Marie Nielsen	marie.nielsen@alphasolutions.dk	mnie	2	• Frontend • Java • Project Management	Add
3	Peter Hansen	peter.hansen@alphasolutions.dk	phan	1	• Frontend • Backend • Java	Add
6	Sofie Christensen	sofie.christensen@alphasolutions.dk	schr	1	• Project Management • Database Design	Add
7	Jacob Larsen	jacob.larsen@alphasolutions.dk	jlar	2	• Cloud Architecture	Add
9	Frederik Møller	frederik.moller@alphasolutions.dk	fmol	2	• UI/UX Design • Testing	Add
10	Ida Olsen	ida.olsen@alphasolutions.dk	iols	1	• Java	Add
11	Christian Berg	christian.berg@alphasolutions.dk	cber	1	• Frontend • Backend	Add

At the bottom left is a 'Log out' link.

Figur 33) Hjemmeside URL: “/project/projectId/assignees?Skill=”, ikke sorteret efter nogle skills

Når der er sorteret efter en skill, vil den se sådan her ud:

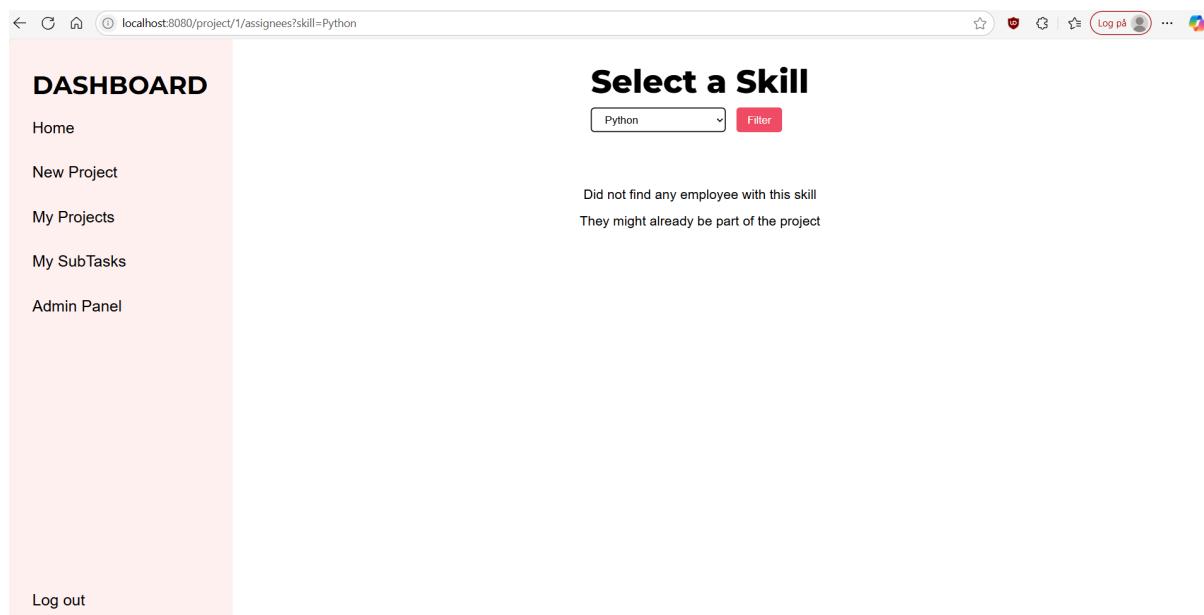
The screenshot shows the same 'Select a Skill' interface as Figure 33, but with a dropdown menu in the filter set to 'Frontend'. The table data is now filtered to show only employees with the 'Frontend' skill. There are 5 rows remaining:

Employee ID	Name	Email	Username	Role ID	Skills	Edit Panel
2	Marie Nielsen	marie.nielsen@alphasolutions.dk	mnie	2	• Frontend • Java • Project Management	Add
3	Peter Hansen	peter.hansen@alphasolutions.dk	phan	1	• Frontend • Backend • Java	Add
11	Christian Berg	christian.berg@alphasolutions.dk	cber	1	• Frontend • Backend	Add
13	Thomas Kjær	thomas.kjaer@alphasolutions.dk	tkja	1	• Frontend • Java	Add

At the bottom left is a 'Log out' link.

Figur 34) Hjemmeside URL: “/project/projectId/assignees?Skill=Frontend”, sorteret efter “Frontend”

Hvis der er sorteret ud for en skill, men den ikke har kunnet finde nogle brugere med den givne skill, måske fordi de allerede er tilføjet til projektet, så vil den se sådan her ud:



Figur 35) Hjemmeside URL: “/project/projectId/assignees?Skill=Python”, sorteret ud for en skill, fandt ingen medarbejdere

Sort my sub tasks:

Efter møde med vores PO, fandt vi ud af, at der var et behov for en medarbejder at kunne få et overblik over de subtask, som de havde og at kunne filtrere i dem efter givne parametre, såsom prioritet, slut tidspunkt og størrelse.

Vi udviklede dermed et GET endpoint, som kunne håndtere denne opgave, og figur 36 viser dette. Som det kan ses på endpointet, så bruger vi en RequestParam, som er en string kaldet “chosen”, dette er den faktor man filtrere subtask ud fra. Denne angav vi også med “required = false”, da det skulle være muligt at se en liste over subtask, som ikke var sorteret ud for en faktor.

```
@GetMapping("/mySubTasks/sortBy") huw02
public String sortMyTasks(@RequestParam(required = false) String chosen, HttpSession session, Model model){
    if(!g1SService.isLoggedIn(session)){
        return "redirect:/login";
    } //sorterer en employees subtask ud for en given faktorer, såsom prioritet
    Employee employee = (Employee)session.getAttribute("employee");
    List<SubTask>subTaskList = g1SService.getSortedSubtaskByEmployeeId(chosen, employee.getEmployeeId());
    if(subTaskList == null || subTaskList.isEmpty()){
        model.addAttribute("noSubTasks", true);
        model.addAttribute("found", false);
        return "mySubTasks";
    }
    model.addAttribute("found", true);
    model.addAttribute(subTaskList);
    return "mySubTasks";
}
```

Figur 36) Viser det GET endpoint der hører til my subtasks, her vises en liste over ens subtask, som så kan sorteres ud for forskellige faktorer.

Som det ses på endpointet, så er den første metode "isLoggedIn", denne metode er en metode, som de fleste af vores endpoints indeholder, da den tjekker, hvorledes personen er logget ind. Dette gøres ved at tjekke om der ligger et Employee objekt i brugerens session.

På linjen under, laver vi et employee objekt, ud fra det "employee" objekt der ligger i brugerens session. Efterfølgende laver vi en liste af subtask, hvor vi bruger vores RequestParam "chosen" og vi bruger medarbejdernes "employeed", som parametre i metoden "getSortedSubtaskByEmployeed". Metoden kalder service klassen, som så kalder metoden i repository. Vi startede med at lave nedenstående metode, som det kan ses på figur 37.

```

727     public List<SubTask> getSortedSubtaskByEmployeed(String sortColumn, int employeed) { no usages ± huw02
728         List<String> allowedSortColumns = List.of("subtask_estimate", "subtask_end_date", "subtask_end_date desc", "subtask_priority");
729         if(sortColumn == null || sortColumn.isBlank()){
730             String sqlNotSorted = "SELECT subtask.* FROM subtask " +
731                 "JOIN subtaskassignees ON subtask.subtaskId = subtaskassignees.subtaskId " +
732                 "WHERE subtaskassignees.employeed = ? ";
733             List<SubTask> subTaskListNotSorted = jdbcTemplate.query(sqlNotSorted, new SubTaskRowMapper(), employeed);
734             if(subTaskListNotSorted.isEmpty()){
735                 return null;
736             }//sætter subprojectId på subtask objektet
737             for(SubTask i: subTaskListNotSorted){
738                 i.setSubProjectId(getSubProjectIdWithSubTaskId(i.getSubtaskId()));
739             }
740             return subTaskListNotSorted;
741         }
742         if (!allowedSortColumns.contains(sortColumn)) {
743             return null;
744         }
745
746         String sql = "SELECT subtask.* FROM subtask " +
747             "JOIN subtaskassignees ON subtask.subtaskId = subtaskassignees.subtaskId " +
748             "WHERE subtaskassignees.employeed = ? " +
749             "ORDER BY " + sortColumn;
750
751         List<SubTask> subTaskList = jdbcTemplate.query(sql, new SubTaskRowMapper(), employeed);
752
753         if(subTaskList.isEmpty()){
754             return null;
755         }//sætter subprojectId på subtask objektet
756         for(SubTask b: subTaskList){
757             b.setSubProjectId(getSubProjectIdWithSubTaskId(b.getSubtaskId()));
758         }
759         return subTaskList;
760     }

```

Figur 37) Ovenstående metode er ikke længere i brug, da den blev effektiviseret.

Dog virkede det ulogisk, at man havde to queries, hvor den eneste forskel var, at query nr 2, havde "ORDER BY (efterfulgt af en filtrering, f.eks. prioritet)", når man i stedet kunne lave string concatenation logik, som kunne klare forskellen mellem de to queries. Derfor lavede vi metoden, som kan ses på figur 38.

```

762     public List<SubTask> getSortedSubtaskByEmployeeIdPerfected(String sortColumn, int employeeId) { 1 usage ± huw02
763         List<String> allowedSortColumns = List.of("subtask_estimate", "subtask_end_date", "subtask_end_date_desc", "subtask_priority");
764         String order = ""; //initialiser en tom string
765         if (sortColumn != null && !sortColumn.isBlank()) { // checker om vores sorterings input er en tom string
766             if (allowedSortColumns.contains(sortColumn)) { //checker om vores sorterings input er en af dem i vores DB
767                 order = "ORDER BY " + sortColumn; //starter vores string med mysql sorterings formen "order by"
768                 order += sortColumn; //definerer hvad vi vil sortere efter fx, slut dato
769             }
770         } //laver en query, hvor vi vil have alle subtask for en given employee og evt sortere dem ud for en parameter
771         String sql = "SELECT subtask.* FROM subtask " +
772             "JOIN subtaskassignees ON subtask.subtaskId = subtaskassignees.subtaskId " +
773             "WHERE subtaskassignees.employeeId = ? " + order;
774         List<SubTask> subTaskList = jdbcTemplate.query(sql, new SubTaskRowMapper(), employeeId);
775         //vi får forhåbenligt en liste med subtask tilbage, hvis den er tom, så returnere vi null
776         if(subTaskList.isEmpty()){
777             return null;
778         } //etter subprojectId på subtask objektet
779         for(SubTask b: subTaskList){
780             b.setSubProjectId(getSubProjectIdWithSubTaskId(b.getSubtaskId()));
781         } //nedenstående metode er nødvendig, fordi "order by" kommer til at sortere alfabetisk ved priority
782         //så sortere den efter high, low og så medium, fordi "l" kom før "m" i alfabetet, metoden sortere dem så efter den rigtige prioritet
783         if(sortColumn != null && sortColumn.equalsIgnoreCase("subtask_priority")){
784             return sortSubtasksByPriority(subTaskList);
785         }
786     }
787 }
```

Figur 38) Metode i repository, der returnerer en liste af subtask, kan sorteres ud for en faktor.

Metoden starter med at have en liste over strings, som var de faktorer, man kunne sortere sin subtask ud fra. Her havde vi estimat, slut dato, slut data faldende og prioritet.

Efterfølgende lavede vi så en tom string kaldet "order", denne ville blive udfyldt med "order by" og så hvad end man ville sortere med. Dette ville ske, hvis det input man fik fra brugeren, kaldet "sortColumn", ikke var null. Hvis det ikke var null, så lavede vi vores String "order" om til en blanding af "order by" og brugernes input, fx estimat.

Efterfølgende ville vi så lave vores query, hvor vi valgte alle data fra subtask. Her lavede vi en "join" på "subtaskAssignees", altså dem som var sat på en given subtask. Her koblede vi "subtaskAssignees" foreign key, kaldet for subtaskId sammen med "subtask" primary key kaldet subtaskId. Vi kunne til sidst lave en "where" statement, hvor vi kaldte "subtaskAssignees" employeeId = ?, dette gjorde, så vi kun fik alle de subtask der hørte til et specifikt employeeId.

Nu kunne vi ved hjælp af jdbcTemplate's metode "query" lave en liste af subtask for den givne bruger, der var logget ind i systemet. Det skyldtes, at vi som parametre i den kaldte "query" metode, havde angivet vores sql statement, altså de informationer vi ville have fra systemet. Vi havde defineret en "rowmapper", som vi kaldte for "SubtaskRowMapper", og den tredje parameter var den employeeId, som blev fanget fra session.

Da vores “SubtaskRowMapper” ikke var defineret så den hentede subprojectId’et til alle de givne subtask, da det kom fra en anden tabel, så lavede vi følgende metode:

```

788
789     public int getSubProjectIdWithSubTaskId(int subtaskId){ 3 usages ± huw02
790         String sql ="SELECT task.* from task " +
791             "join subtask on task.taskId = subtask.taskId " +
792             "where subtask.subtaskId = ?";
793         List<Task>subProjects = jdbcTemplate.query(sql, new TaskRowMapper(), subtaskId);
794         if(subProjects.isEmpty()){
795             return 0;
796         }
797         return subProjects.getFirst().getSubprojectId();
798     }

```

Figur 39) Man giver metoden et subtaskId, og får så det subprojectId tilbage, som hører til den givne subtask.

Denne metode tog som parameter subtaskId’et for en subtask, og dens “parent task”, som vores subtask hørte til og dens taskId lå inde i subtask tabellen. Vi kunne derfor lave en simpel query, hvor vi fik subprojectId ved hjælp af en join mellem task og subtask.

Task tabellen indeholdt nemlig den subprojectId som vi ledte efter. Som set på figur 38, linje 779 og 780, så kunne vi iterere gennem listen og sætte subprojectId’et på alle subtask objekterne, dette var nødvendigt, da vi skulle bruge subprojectId’et til at gøre, så man kunne trykke på alle ens subtask, og så ville man ryge ind på den givne subtask. SubprojectId’et var nemlig en del af url’et “/subproject/{ProjectId}/edit/subtask/{subtaskId}”, som var der, hvor man så subtask i større detaljer.

Den sidste del af figur 38, var linje 783-785, dette blev vi nødt til at implementere, da vi stødte på en fejl.

Fejlen var, at når Mysql skulle sortere en query ud fra “ORDER BY”, så sorterede den værdier som kun indeholdt bogstaver. Så ville den sortere det alfabetisk, hvilket var et problem, da vores prioriteter hed “high”, “medium” og “low”, men de blev så sorteret “high”, “low” og “medium”, da “L” kommer før “m” i alfabetet. Derfor implementerede vi følgende metode til at fikse problemet:

```

840     public List<SubTask>sortSubtasksByPriority(List<SubTask>subTasks){ 1 usage ± huw02
841         List<SubTask> subTasksList = new ArrayList<>();
842         if(subTasks == null || subTasks.isEmpty()){
843             return null;
844         }
845         for(SubTask i: subTasks){
846             if(i.getSubtaskPriority().equalsIgnoreCase( anotherString: "High")){
847                 subTasksList.add(i);
848             }
849         }
850         for(SubTask b: subTasks){
851             if(b.getSubtaskPriority().equalsIgnoreCase( anotherString: "Medium")){
852                 subTasksList.add(b);
853             }
854         }
855         for(SubTask c: subTasks){
856             if(c.getSubtaskPriority().equalsIgnoreCase( anotherString: "Low")){
857                 subTasksList.add(c);
858             }
859         }
860         return subTasksList;
861     }

```

Figur 40) Sorterer subtask efter prioritet.

Følgende metode bliver kun aktiveret, hvis input fra brugeren er "subtask priority" og er meget simpel, da den bare looper gennem listen af subtask, og sorterer efter high, medium og low, hvor den til sidst returnerer den sorterede liste. Den returnerede liste med subtasks vil så blive vist for brugeren under "My SubTasks", hvis der er en liste, som ikke er tom. I tilfælde af at listen er tom, så vil der komme en besked op, der fortæller brugeren, at der ikke kunne findes nogle subtasks.

Vi håber følgende afsnit har givet et indblik i vores kodningsproces, og hvordan vi igennem forløbet har udviklet os, så vi har kunnet lave mere effektiv kode. Nogle af de tiltag der er blevet taget har været med til at øge læsbarheden af vores kode, samt performances for selve programmet. Yderligere har den også gjort selve programmet mere klimavenlig, da de opdaterede metoder kræver færre kald frem og tilbage fra databasen, hvilket dermed kræver mindre energi.

Status på implementering

Følgende afsnit vil redegøre for hvilke implementeringer der blev inkluderet i systemet, samt hvilke implementeringer vi havde planlagt i et senere sprint. Vi vil også gennemgå, hvordan man kunne have lavet systemet anderledes og hvilke eventuelle ændringer der kunne have forbedret systemet.

Nuværende implementeringer:

Vores formål med første sprint var at bygge en succesfuld MVP, som havde de basisfunktioner et projekt kalkulerings værktøj havde brug for. Vi fik derfor lavet CRUD funktionalitet til projekt, sub projekt, task, sub tasks, medarbejdere og deres skills. Dette betød at vi havde et produkt, der kunne benyttes af Alpha Solutions, og vi fremlagde det for vores Product Owner til vores PO-møde. PO var imponeret og synes, der var god funktionalitet. Han ønskede dog, at der var et mere overskueligt overblik over ens projekter, og hvis man kunne få det grafisk, ville det være en fordel.

Til vores PO-møder var der også lagt vægt på, at Alpha Solutions var et stort firma med mange medarbejdere og projekter. Vi besluttede os derfor at lave filtrering og sorteringslogik, hvilket vil gøre systemet mere overskueligt. Første funktion var at man kunne tilføje medarbejdere til et projekt, men da der var mange medarbejdere i firmaet, så lavede vi en filtrering, hvor man kunne filtrere medarbejderne ud fra deres skills. Dette var relevant, da man så hurtigere kunne finde de medarbejdere, som havde de skills man skulle bruge i et givent projekt.

Vi besluttede os også for at lave en side, hvor man kunne se alle de subtask, som man havde sat sig selv på. Her fik man et hurtigt overblik over opgaverne og man kunne så trykke på hver enkelt opgave for at få mere af vide om selve opgaven. Vi implementerede her en sorteringslogik, hvor brugeren kunne sortere sine opgaver ud fra forskellige faktorer. Vi valgte at sortere ud fra 4 forskellige faktorer, som vi tænkte var de mest essentielle elementer, når man stod med en masse opgaver. De 4 elementer man kan sortere ud fra er estimat, slutdato stigende, slutdato faldende og prioritet.

Fremtidige implementeringer:

Som det også ses på vores scrumboard i "github projects", så havde vi funktionalitet i vores tredje sprint, som vi desværre aldrig fik startet på. Dette skyldtes, at vi valgte at lægge vægt på at skrive en god rapport. Vi fremlagde denne plan for PO, og han var tilfreds med den funktionalitet vi havde fået implementeret, og mente at tredje sprint funktionalitet ikke var en

nødvendighed for et godt system. Vi havde i tredje sprint tænkt os at lave følgende funktionaliteter:

- Gantt diagram der kunne illustrere et grafisk billede over, hvilket projekter der tog flest ressourcer, og der var tættest på deadline.
- Et overblik over hvor meget CO₂ et givent projekt ville tage. Her havde vi planlagt at beregne antallet af træer, der skulle plantes, for at et projekt var CO₂ neutralt. Dette kunne også illustreres grafisk.
- En homepage, hvor man kunne se et overblik over alle ens projekter. Her skulle det være muligt for en projektleder at sende besked til medarbejdere, og en medarbejder skulle så kunne se disse beskeder.
- En side hvor admin kunne se en liste over alle de projekter der lå i systemet, her skulle admin kunne slette hvert enkelt projekt.
- Logging system, så admins kunne se hvem der loggede ind, tilføjede, opdaterede eller slettede diverse ting.

Disse implementeringer kunne have været interessante at arbejde med. Men disse implementeringer krævede ændringer i vores database, og der skulle implementeres en masse logik, hvilket kunne påvirke allerede fungerende funktioner i vores projekt. Da vi havde en fastlåst deadline, ville vi ikke risikere at give os selv for mange opgaver i slutningen af projektet.

Forbedringer i systemet

Designet af vores system var lavet ud fra den viden, vi havde tilegnet os i vores undervisning. Vi havde dermed kun arbejdet med projekter, hvor vi havde én repository, service og controller klasse.

Ved projektets afslutning gik det op for os, at det kunne have været en fordel at opdele projektet i flere klasser, så man havde flere repositories, service og controller klasser til hver enkelt funktionalitet. Dette ville gøre systemet mere læsbart og nemmere at finde rundt i. Yderligere ville det også fremme single responsibility principippet, da hver serviceklasse kun ville have ansvaret for én given rolle. Dette ville også medføre, at vi havde mere High Cohesion i vores system, da der ville være en højere samhørighed mellem klasserne.

Test

For at sikre korrekt funktionalitet og stabilitet i vores kode, så har vi valgt at teste vores kode med en kombination af controllertest og integrationstest. Vi benytter `@SpringBootTest` til at teste metoderne i repositoryklassen og `@WebMvcTest` til at teste controllerlaget isoleret ved hjælp af `MockMvc`. Vi anvender også Maven til at automatisere kørslen af vores test, når vi opretter en pull-request. Dette hjælper os med at fange fejl og sikre at ny kode ikke ødelægger den funktionalitet, som i forvejen virker.

Integrationstest

Vi har valgt at køre vores integrationstest på en H2 in memory database, da det giver en række fordele i performance og testisolering. Fordi H2-databasen eksisterer i hukommelsen, så er den hurtig at oprette og vi undgår at teste direkte på produktionsdatabasen. Dermed forhindrer vi også mulige komplikationer og ændringer i produktionsdatabasen. Til opsætning af vores integrationstest har vi oprettet en separat `application-test-properties`, hvor vi aktiverer `testprofile` og får Spring til at pege på et SQL-script (`h2init.sql`), som vi også har lagt i `test/resources`.

```
spring.profiles.active=test

spring.h2.console.enabled=true
spring.datasource.generate-unique-name=false
spring.datasource.name=integrationtest
spring.datasource.username=test
spring.datasource.password=test
spring.sql.init.mode=never
```

Figur 41) Application-test-properties

```
@SpringBootTest
@Sql(executionPhase = Sql.ExecutionPhase.BEFORE_TEST_METHOD, scripts = {"classpath:h2init.sql"})
@Transactional
@Rollback(true)
public class alphaSolutionsIntegrationsTest {

    @Autowired
    private G1SRepository g1SRepository;

    @Autowired
    private G1SService g1SService;

    @BeforeEach
    void setup() {}

    @AfterEach
    void teardown() {}
```

Figur 42) Udsnit af opsætningen af integrationstestklassen

Som det ses på figur 42 ovenfor, så anvender vi nogle vigtige annoteringer, som @Sql, @Transaction og @Rollback. @Sql(... BEFORE_TEST_METHOD ...) betyder at h2init.sql skal køres før hver enkelt testmetode. Derudover har vi en kombination af @Transactional og @Rollback, som sørger for at eventuelle tilføjelser og ændringer lavet i en testmetode bliver rullet tilbage og nulstillet efter hver testmetode. Til sidst anvender vi @Autowired til at få vores G1SService og G1SRepository dependency injected i vores integrationstestklasse.

I vores integrationstest har vi primært haft fokus på CRUD-operationer. Udeover CRUD, har vi testet kryptering og validering af adgangskoder.

Der er skrevet tests for følgende entiteter og metoder:

Entitet	Testede metoder
Project	Opret, læs, opdater, slet
SubProject	Opret, læs, opdater, slet
Task	Opret, læs, slet
SubTask	Opret, læs
Assignees	Tilføj, hent, fjern fra task/subtask
Password	Kryptering og validering af adgangskode

Et eksempel på hvordan vi tester oprettelse af et nyt projekt:

```

@Test
void testCreateProject() {
    // Der oprettes et nyt projekt
    Project newProject = new Project();
    newProject.setProjectName("Test project");
    newProject.setProjectStartDate(Date.valueOf("2025-05-08"));
    newProject.setProjectEndDate(Date.valueOf("2025-05-09"));
    newProject.setEmployeeId(1);
    newProject.setProjectDescription("Test project description");

    g1SRepository.createProject(newProject);

    // Tester om projektet har fået tildelt et projectId
    assertTrue( condition: newProject.getProjectId() > 0);

    // Henter projektet fra databasen og tester om værdierne er de samme
    Project dbProject = g1SRepository.getProjectById(newProject.getProjectId());

    assertNotNull( actual: dbProject);
    assertEquals( expected: newProject.getProjectName(), actual: (dbProject.getProjectName()));
    assertEquals( expected: newProject.getProjectStartDate(), actual: dbProject.getProjectStartDate());
    assertEquals( expected: newProject.getProjectEndDate(), actual: dbProject.getProjectEndDate());
    assertEquals( expected: newProject.getEmployeeId(), actual: dbProject.getEmployeeId());
    assertEquals( expected: newProject.getProjectDescription(), actual: dbProject.getProjectDescription());
}

```

Figur 43) Integrationstest der tester metoden "createProject"

Vi starter med at instantiere et tomt projekt, hvorefter vi tildeler projektet nogle værdier, som navn, startdato, slutdato, medarbejder(projektleder/admin) og en projektbeskrivelse. Herefter kalder vi metoden `createProject(newProject)`, som hentes fra G1SRepository og opretter projektet i H2-databasen. Vi anvender en `assertTrue` for at teste om projektet har fået tildelt et korrekt id. Der oprettes nu et `dbProject`, hvor vi kalder `getProjectById(newProject.getProjectId())`. Denne metode henter et projekt fra H2-databasen, hvor id'et stemmer overens med id'et på det projekt vi oprettede i databasen. Vi bruger `assertNotNull` og `assertEquals` for at teste om projektet blev hentet korrekt fra H2-databasen og for at undersøge om projektets navn, startdato, slutdato, medarbejder og beskrivelser matcher med det projekt vi oprettede manuelt i starten.

```
@Test
void testEncryptPassword() {
    String testPassword = "hello";
    String encrypted = g1SService.encryptTest(testPassword);

    assertNotNull( actual: encrypted);
    assertTrue( condition: g1SService.decryptTest(testPassword, encrypted));
    assertFalse( condition: g1SService.decryptTest( password: "wrong", encrypted));
}
```

Figur 44) Integrationstest der tester metoden “EncryptPassword”

I figur 44 ovenfor undersøger vi, om vores krypteringsfunktionalitet virker korrekt. Vi har oprettet en test, som kontrollerer både krypteringen og kontrollerer om adgangskoden er korrekt. Testen hedder testEncryptPassword og anvender en String “hello” som et eksempel på brugerens adgangskode. Først krypteres adgangskoden med encrypt Test metoden, hvorefter vi tester at den krypterede kode ikke er null. Efterfølgende tester vi om den krypterede kode og den ikke krypterede kode stemmer overens ved hjælp af decryptTest metoden. Til sidst undersøges med en assert False, at en forkert adgangskode ikke ville blive valideret korrekt. På den måde sikrer vi at systemet beskytter brugerens adgangsdata og at systemet afviser ugyldige loginforsøg.

Controllertest

En controllertest, også kaldet en unit test af controllerlaget, bruges til at sikre, at vores endpoints fungerer korrekt. Vi tester om brugeren bliver sendt til den rigtige side, om redirects fungerer som de skal, og om data tilføres modellen korrekt. Disse test er vigtige for at undersøge om samspillet mellem systemet og brugeren virker hensigtsmæssigt.

```
@WebMvcTest(G1SController.class)
public class controllerTest {
```

Figur 45) Annotation der gør spring boot opmærksom på, at vi kun ønsker at teste vores controller

Vores testklasse er annoteret med @WebMvcTest(G1SController.class), som betyder at det kun er vores controller, der startes op under testene og ikke hele applikationen. Det gør testene hurtigere og mere fokuseret, fordi vi kun tester controllerens funktionalitet. Selve HTTP-requests simuleres ved brug af MockMvc, som gør det muligt at efterligne en GET og POST requests mod vores endpoints uden at der skal startes en server op.

I controllertestklassen har vi anvendt følgende annoteringer:

- `@WebMvcTest(G1SController.class)` fortæller Spring at det kun er controlleren, som skal testes.
- `@Autowired` bruges til at dependency inject en instans af `MockMvc`, som vi bruger til at simulere requests.
- `@MockitoBean` bruges til at mocke `G1SService`, så vi selv kan definere, hvad service metoderne skal returnere under testene.
- `@BeforeEach` og `@AfterEach` bruges til at oprette testdata inden en test og rydde op i data efter en test, så alle test starter med samme udgangspunkt.

```
@Test
void createSubproject() throws Exception {
    when(methodCall: g1SService.isLoggedIn(session)).thenReturn( t: true);
    mockMvc.perform( requestBuilder: get( uriTemplate: "/create/subproject/1").session(session))
        .andExpect( matcher: status().isOk())
        .andExpect( matcher: view().name( expectedViewName: "createSubproject"));

}
```

Figur 46) Unit test der tester metoden “CreateSubproject”

Et eksempel på en GET-request test er vores metode `createSubproject`. Denne test kontrollerer om brugeren har adgang til siden, hvor man kan oprette et nyt subprojekt. Vi bruger `when(g1SService.isLoggedIn(session)).thenReturn(true)` til at simulere at brugeren er logget ind, hvilket er nødvendigt for at få adgang til controllermetoden. Med `mockMvc.perform(get(...))` simulerer vi et kald til endpointet `/create/subproject/1`. Til sidst tjekker vi om siden bliver vist korrekt med `status().isOk()` og at den returnerer det rigtige view “`createSubproject`”. Denne test sikrer at URL'en returnerer den korrekte HTML-side.

```

    @Test
    void addSubproject() throws Exception {
        when( methodCall: g1SService.isLoggedIn(session)).thenReturn( t: true);
        mockMvc.perform( requestBuilder: post( uriTemplate: "/create/subproject").session(session)
            .param( name: "subprojectId", ...values: "1")
            .param( name: "sub projectName", ...values: "test")
            .param( name: "sub projectStartDate", ...values: "2025-05-10")
            .param( name: "sub projectEndDate", ...values: "2025-05-10")
            .param( name: "projectId", ...values: "2"))
            .andExpect( matcher: status().is3xxRedirection())
            .andExpect( matcher: redirectedUrl( expectedUrl: "/project/" + 2));
    }
}

```

Figur 47) Unit test der tester metoden “addSubProject”

I dette eksempel tester vi en POST-request, hvor vi simulerer at en bruger udfylder en formular til at oprette et nyt subprojekt, hvorefter formularen sendes videre til oprettelse. Vi bruger mockMvc.perform(post(...)) og tilføjer parametre som subprojectId, sub projectName, sub projectStartDate osv. med .param(). Herefter tjekker vi at POST-requesten resulterer i en redirect med status().is3xxRedirection(). Dette skyldes, at en redirect har statuskode 300 til 399. Til sidst testes om brugeren sendes til den korrekte endpoint med redirectedUrl("/project/" + 2). På den måde har vi testet om backenden modtager input korrekt og at applikationen reagerer som forventet med at brugeren redirectes til den rigtige side.

Maven yml test

```

22      | - name: Build with Maven
23      |   run: mvn -B test

```

Figur 48) Udsnit af maven.yml filen

Da vores projekt er bygget med Maven, så kan vi bruge kommandoen “mvn -B test”, som ses i figur 48, til at automatisere kørslen af vores testkode. Når denne kommando bliver kørt, så kompilerer Maven testkoden og kildekoden, hvorefter den tester alle de enhedstest, som er skrevet med JUnit-testframeworket. Dvs. at alle test i vores testklasser er annoteret med @Test.

“mvn -B test” kommandoen bliver anvendt i vores automatiserede Github Actions CI-pipeline process. Det betyder at kommandoen bliver kørt hver gang der laves en ændring i koden og der åbnes en pull-request. Hvis testen fejler, så er det ikke muligt at merge den nye kode ind

i main branchen, hvilket hjælper med at opdage fejl hurtigt og forhindrer at ny koden ødelægger funktioner i systemet.

I kommandoen ses “-B”, som står for “batch mode”. Det betyder at kommandoen kører uden der skal ventes på et brugerinput. Dette er nødvendigt, da vi gerne vil have at testene skal køres automatisk ved en push eller merge til main branch. Ud over at køre testene, så sørger Maven også for at bygge projektet som en del af processen.

Scrum processen

Da vi arbejder som et team, har vi valgt at anvende en agil udviklingsmetode for at understøtte arbejdsprocessen og effektivisere samarbejdet. Dette er årsagen til at vi har valgt at benytte scrum.

Scrum er et agilt framework, der hjælper os som et team med at strukturere vores arbejde, skabe en synlig process for alle udviklere, scrum master og product owner samt at sikre at vi opnår mest mulige ud af vores indsats.

Ved at have en Product backlog, Sprint planning, Sprint backlog, Daily stand up, Sprint review og Sprint retrospective - er der en glidende overgang over hvor vi individuelt er i processen, hvem der har udfordringer, løsninger og hvad der gik godt og mindre godt. Dette vil hjælpe med samarbejdet mellem udviklere.

Sprint 1

Sprint planning

Inden vores møde med vores product owner (PO) satte teamet sig sammen for at planlægge indholdet af sprint 1. Teamet blev enige om, at første sprint skulle fokusere på at levere et MVP. Derfor kiggede vi på vores User stories samt domain model med henblik på at identificere de potentielle user stories, der vil sikre os et funktionelt MVP efter første sprint. På baggrund af teamets effektive analyser fik vi udvalgt de relevante user stories til Sprint Backlog og forberedte vores forslag til PO.

Ved at have et møde med vores product owner (PO) har hele teamet bidraget til at opnå en fælles forståelse af projektets formål samt rammerne for det første sprint. Med enighed fra begge parter, udviklere og product owner, er vi kommet frem til, at vi skal lave et MVP produkt.

Formålet med at få etableret et MVP i første sprint er at sikre en tydelig rød tråd mellem kundens behov, forståelse og vores udvikling af projektet. På den måde kan vi tidligt i forløbet sikre os, at projektet/produktet bevæger sig i den rigtige retning og vi minimerer ressourcesspild ved misforståelser. Ved færdiggørelse af MVP kan vi få tilføjet yderligere funktionalitet, opgraderinger og forbedringer baseret på feedback fra kunden.

Product backlog

I starten af projektet udarbejdede vi en række user stories ud fra kravene til et MVP-projekt. Disse user stories dækkende blandt andet login, CRUD på projekter, visning af dashboard og oprettelse af subprojekter, tasks og subtask. Undervejs i sprint 1 opdagede vi dog, at nogle nødvendige funktioner manglede i vores backlog, før vi opfyldte kravene til MVP-projektet. Vi manglede en budget-attribut i både vores projekt og subprojekt, samt behovet for at kunne beregne og vise fremdrift i procent baseret på færdiggjorte tasks og subtasks. Disse blev efterfølgende tilføjet som user stories i sprint 2.

Sprint backlog

Sprint backloggen i Sprint 1 indeholdt user stories relateret til grundlæggende funktionalitet: login, CRUD på projekter, oprettelse og visning af subprojekter, tasks og subtasks, samt opbygning af dashboard via fragmenter. Mod slutningen af sprint 1 opstod der nye behov, såsom at kunne klikke ind på enkelte tasks og subtasks for at få mere detaljeret information. Vi fandt også nogle forbedringer for brugeroplevelsen, såsom et søgefelt når man tilføjer assignees til projekter. Disse funktioner er blevet omformuleret som user stories og planlagt til sprint 2.

Daily stand up

Vi har afholdt daglige stand-ups, hvor hvert gruppemedlem kort har gennemgået, hvad de har lavet, hvad de skal arbejde videre med, og hvilke forhindringer de eventuelt har mødt. Dette har givet et godt overblik og hjulpet os med at følge hinandens fremdrift. Generelt har der været få forhindringer. De fleste problemer er blevet løst hurtigt gennem debugging eller samarbejde.

Sprint review

Efter at have færdiggjort vores user stories og task havde vi et MVP, som vi kunne præsentere for vores PO. Under mødet blev vi opmærksomme på, at vi havde misforstået en del af hvad PO og AlphaSolutions ønskede. Det drejede sig specifikt om forskellen mellem estimeret tid og hvordan det blev fremvist. Dette var ikke tydeligt for os i første omgang og vi havde desværre gjort det på en anden måde.

Selv misforståelsen var ikke kritisk, men det var vigtigt at få afklaret, så vi ikke fik lavet et produkt som ikke afspejlede sig med kundens ønske. Gennem dialog med vores PO fik vi præciseret, hvad kunden ønskede. Vores product backlog er nu blevet tilrettet og opdateret, så den afspejler de krav og behov, som kunden ønsker for projektet.

Sprint retrospective

Under sprint retrospective blev det tydeligt, at det havde været en fordel at bruge mere tid på analyser og user stories i starten. Flere væsentlige funktioner blev først identificeret undervejs i sprintet, hvilket skabte lidt ekstra arbejde. Vi vurderede dog også, at teamet har været gode til at reagere på ændringer og justere planen løbende. Til næste sprint vil vi forsøge at være endnu mere præcise i vores backlog og user stories fra start, og måske planlægge mere tid til at designe frontend før udviklingen går i gang.

Sprint 2

Sprint planning

Teamet havde gennemført sprint 1 med succes, vi havde lavet en MVP der tilfredsstillede PO med vores funktionalitet, design og fremtidige planer for, hvad systemet skulle kunne. Vi fik ved sidste PO møde også afklaret nogle ting, som vi tidligere havde forstået på en anden måde. Dette inde blandt, blandt andet måden hvorpå systemets estimerede timer skulle foregå.

Da vi gik i dialog med Alpha Solutions, så havde vi forståelsen af, at dette skulle være et værktøj, som de brugte efter de havde forhandlet en pris med deres kunde, og det skulle dermed være et værktøj, som bekræftede at den forhandlede mængde timer, som de blev betalt for, også hænger sammen med de reelle antal timer brugt på et givent projekt. Dog fandt vi ud af, i samarbejde med PO, at systemet skulle bruges før man havde aftalt en pris/antal timer med kunden, derfor var der enkelte funktioner som vi skulle tænke på en anden måde, dog har det ikke påvirket projektet negativt på nogle måder.

Product backlog

Vi fik i Sprint 1 lavet alle vores nødvendige CRUD metoder, og vores product backlog var derfor fyldt op med User stories, som ville tilføje ekstra funktionalitet til programmet, som ville gøre det mere overskueligt at håndtere store og flere projekter af gangen. Vi havde blandt andet funktioner, som ville gøre det muligt at sortere i projekterne, ændre på medarbejdernes skills, danne grafiske modeller over projekterne med dets opgaver og kontrollere projektets estimerede og aktuelle brugte timer.

Sprint backlog

Vi valgte at tage de otte mest essentielle user stories med i Sprint 2, og gemme et par funktioner til Sprint 3. De mest essentielle user stories vurderede vi, ville være at man kunne registrere og opdatere en medarbejdernes skills. Man ville kunne se modeller over projektets timer heriblandt, om det estimerede og aktuelle antal timer stemte overens. Samt at systemet skulle gøres mere overskueligt for brugeren ved blandt andet at implementere sortering af projekter og et bedre UI.

Sprint review

Vores 2 sprint har været en succes, hvor der er blevet udviklet en masse gode features, og der er lavet et UI design, som ser godt ud, samt er nemt at bruge. Vi har været i dialog med PO til vores sidste møde, her blev der vist produktet, hvor han var mere end tilfreds. Han syntes systemet havde fået en masse gode features, og at vi havde forstået opgaven på en god måde.

Han var imponeret over, hvordan systemets design var nemt at interagere med, og at det var blevet overskueligt at kunne følge sine projekter, da man nu også havde grafiske model over sine projekter. Vi snakkede om vores nye features og de features, vi havde tænkt på at arbejde på i sprint 3.

Han synes, at det var nogle gode ideer vi havde til sprint 3, men han var også overbevist om, at det ikke var funktionalitet, der var nødvendige for at systemet ville være perfekt. På baggrund af denne viden valgte vi at droppe sprint 3, og i stedet fokusere på, at vores nuværende system fungerer på den mest optimale måde.

Sprint retrospective

Vi kunne i 2 sprint godt mærke, at vi havde lært noget af vores fejl i sprint 1. Vi havde dermed blevet bedre til at specificere vores user stories, så vi undgik forvirring og fejl i arbejdsprocessen. Dog var der en enkelt gang, hvor to user stories overlappede så meget,

at vi endte med at være to udviklere på den samme opgave, dette gjorde at vi blev nødt til at kommunikere effektivt, og være påpasselige når vi lavede vores merges, så der ikke opstod konflikter. Men vi kom i mål med opgaven og hele sprintet endte med at blive en success, uden de store konflikter.

Konklusion på scrum process

Vores scrum process har været særdeles effektiv, da vi har formået at afslutte hvert sprint hurtigt og med få fejl. Gruppen har arbejdet som et team, hvor vi sad sammen i den første del af projektet. Dette var en nødvendighed, da det var nemmere at få alles mening om, hvordan vores user stories skulle opbygges, og hvad de skulle indeholde.

Da gruppen så gik i gang med at programmere, så sad vi i de første par dage sammen, da vi så nemt kunne spare med hinanden, om hvordan vi ønskede at opbygge vores kode, og man kunne nemt hente hjælp fra hinanden.

Da vi så stødte på det problem, at vi manglede et godt sted at sidde sammen, så skiftede vi til at køre vores gruppearbejde online. Dette fungerede også fint, da vi kunne bruge en platform kaldet discord til at snakke sammen, og man kunne dele sin skærm, hvis man havde behov for det. Hver morgen før vi gik i gang med at kode, så afholdt vi et “daily scrum”, dette var effektivt, da man så fik et indblik i hvad gruppens medlemmer havde fået færdiggjort, og hvilke problemer de havde stødt på.

Gruppens medlemmer kunne bedst koncentrere sig, når de arbejdede selvstændigt, men havde de andre med på linjen, så de kunne vejlede hinanden. Med dette i mente opbyggede vi vores user stories samt tasks, så de kunne laves af én person. Dette fungerede fint gennem projektet, men gik galt en enkelt gang, da to user stories overlappede hinanden, og de to gruppemedlemmer måtte derfor arbejde sammen.

Vi brugte fibonacci tal, da vi estimerede vores task og user stories, dog var det ikke helt nemt at lave korrekte estimering, da vi kun har arbejdet med et lignende projekt en enkelt gang før. Vores task estimering var til tider for små og enkelte gange for store, dog endte vores user stories med at være estimerede korrekt.

Da vores sidste sprint review var færdigt, så blev gruppen enige om, at vi godt kunne have struktureret vores scrum board på en mere effektiv måde. Vi havde nemlig opbygget scrum boardet med user stories kort og så lavede vores tasks kort som sub issues. Det gjorde at

der var over 150 forskellige kort med user stories og task, hvilket var meget svært at finde rundt i.

Det var til tider også frustrerende, at hver gang man havde lavet en task, så skulle man sætte den fra “in progress” til “testable”, så i “testing” når man testede den, og til sidst efter den var testet færdigt kunne man lægge den i “done”. Denne proces kunne til tider godt ødelægge det gode flow man havde, når man sad og arbejde, da man hele tiden skulle bruge en masse tid og energi med at rette på scrum boardet.

For at opsummere på vores arbejdsprocess, så har der været et par enkelte ting, som vi som gruppe kunne arbejde på, men generelt har processen været god, og det er blevet mere åbenlyst, hvordan et værktøj som scrum hjælper til at organisere og strukturere et større projekt som dette. Vi har derfor lært en masse vigtige ting i dette projekt, indenfor projektstyring og organisering, og disse ting vil helt sikkert være med til at forbedre vores næste projekt.

CI/CD (Continuous Integration / Continuous Deployment) og Deployment

For at sikre kontinuerlig integration og udvikling har vi opsat en CI/CD-pipeline i vores Github ved hjælp af Github Actions. Dette workflow sørger for, at ny kode automatisk testes og valideres, inden det kan blive en del af ‘main’. Målet er at undgå konflikter, fejlbehæftet kode og ikke testede funktioner i produktionen.

Vores workflow er defineret i en eller flere .yml-filer under .github/workflows/. Disse filer beskriver præcist, hvilken arbejdsproces der skal udføres - fx test, build og deployment. I vores tilfælde er workflowet konfigureret til at køre mvn test, så vi sikrer, at alle automatiserede test bliver kørt ved hvert pull request mod main.

For at understøtte dette har vi opsat **Branch Protection Rules**⁴ i Github. Det betyder at:

- Ingen kan merge til main, medmindre alle test er bestået
- Der kræves en code review fra et-to teammedlem
- Merge kan kun ske via pull request.

Arbejdsprocessen følger et klassisk Git flow, se bilag 6.

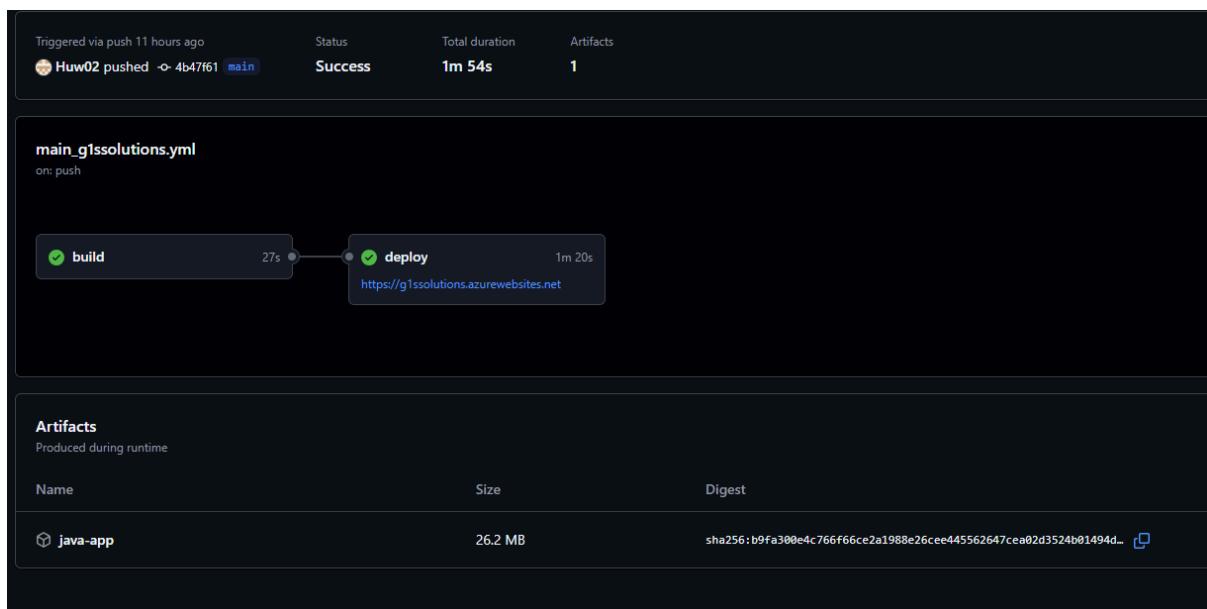
⁴ [Managing a branch protection rule - GitHub Docs](#)

1. En udvikler fra teamet opretter en feature branch
2. Arbejder på sin funktion og laver lokale test
3. Når funktionaliteten er klar, oprettes et pull request mod main
4. Github Actions kører automatisk testpipeline
5. Ved godkendelse fra review og bestået test, kan ændringer merges.

For at gøre det nemt for nye udviklere at følge denne proces, har vi udarbejdet en [CONTRIBUTE.md](#), som beskriver retningslinjer og steps for korrekt brug af vores workflow.

For at demonstrere vores CI/CD-setup har vi inkluderet to skærmbilleder fra vores Github Actions-forløb. Bilag 7 viser et eksempel, hvor en pull request fejlede under testkørslen og derfor ikke kunne merges til main - og dermed heller ikke blev deployed til produktionen.

Omvendt viser figur 49 et eksempel, hvor hele workflowet gennemførte alle test og blev godkendt via review, hvilket betød, at pull request kunne merges og automatisk blive deployet til vores produktionsmiljø.



Figur 49) Github actions succes kørsel

Continuous Integration (CI)

Som vist i ovenstående eksempler bliver ændringer kun merged og deployet, når både test og code review er gennemført succesfuldt. Denne proces bliver styret af vores **Github Actions workflow**, som beskrevet tidligere.

For at give indblik i, hvordan denne automatiserede proces er sat op, har vi inkluderet et af vores workflow-filer i bilag 8. Denne .yml-fil definerer præcis, hvornår workflowet bliver udløst, hvilke trin der køres, og hvilke krav der skal være opfyldt, før koden kan integreres i produktion. På den måde sikrer vi, at vores CI/CD-pipeline passer til vores behov, og at man ikke ved en fejl kan deploye ustabil eller utestet kode, som kan skade produktet.

Ved at kigge på bilag 9, kan man se hvordan workflowet er opbygget. Det aktiveres ved både push og pull request til 'main'. Jobbet kører i en virtuel Linux Ubuntu-miljø hvor følgende steps udføres:

1. Checkout: Koden hentes ned fra Github
2. Setup Java: Der installeres en Java 21 version via Temurin.
3. Build og test: projektet bygges og testes via mvn - b test.

Dette sikrer, at ny kode altid gennemgår et komplet testforløb, inden det kan blive en del af produktionskoden.

Continuous Deployment (CD)

Som en del af vores CI/CD-setup har vi opsat automatisk deployment til Azure, som håndterer overførslen af applikationen til produktionsmiljøet. Deployment-processen styres af en Github Actions workflow, defineret i bilag 8.

Workflowet aktiveres automatisk, når en pull request bliver mergeret til main, og alle test bestået, som beskrevet tidligere i CI. Herefter bygges projektet, og applikationen bliver automatisk deployet til vores Azure Web App, hvor den er offentligt tilgængelig.

Vi har konfigureret tre Spring Bot Profiles - dev, test, og prod - som gør det muligt at skelne mellem udviklings-, test- og produktionsmiljø. Dette håndteres via forskellige application-*.properties-filer som muliggør miljø specifikke indstillinger som fx databaseforbindelser og credentials.

Denne deployment-proces sikrer, at den nyeste stabile version af applikationen altid udrulles automatisk, uden manuelle fejl eller forsinkelser.

Gennemgang af workflow-konfiguration

Bilag 8 viser den fulde konfiguration af vores main_g1ssolutions.yml, som styrer deployment-processen. Selvom workflowet er autogenereret af Microsoft, har vi vurderet, at det er relevant at gennemgå i rapporten.

For at vores Spring Boot profiles kan fungere korrekt i produktionen, har vi konfigureret nødvendige Github repository secrets, som blandt andet indeholder:

- Prod-database credentials
- API-nøgle
- Azure App Service credentials

Disse secrets bliver brugt i workflowet og er også opsat som environment variables i Azure.

Workflowet er struktureret som følger:

1. Checkout: Koden hentes ned fra Github
2. Setup Java: Installeres Java 21 fra Microsoft
3. Build with Maven: Kører ‘mvn clean install’, hvilket fjerner tidligere build, kompilerer koden, kører test og pakker det til en ‘.jar fil’.
4. Upload artifact: ‘.jar-filen’ gøres klar til deployment
5. Deploy environment: En virtuel Linux Ubuntu server gøres klar til at modtage deployment
6. Download artifact: Den pakkede ‘.jar-fil’ downloades til serveren.
7. Login: Logger ind på Azure via credentials defineret i secrets.
8. Deploy: Applikationen deployes automatisk til Azure Web App.

Se bilag 8 for den komplette ‘main_g1ssolutions.yml’ workflow fil.

Azure setup og Miljøer

Efter at have beskrevet vores CI/CD pipelines, vil vi nu forklare opsætningen af vores Azure-miljø, som danner grundlaget for produktionen og tilgængeligheden af systemet.

For at kunne imødekomme både kundens krav om tilgængelighed og KEA’s krav om deployment til cloud, har vi valgt at benytte Azure services. Azure gør det muligt for os at levere en løsning, der både skalerbar, stabil og tilgængelig globalt.

Vi har implementeret følgende:

- Azure Web App - bruges til hosting af vores applikation
- Azure Database for MySql - fungerer som vores produktionsdatabase
- Spring Boot Profiles - skelner mellem dev, test og prod, så vi nemt kan tilkoble de korrekte miljøer.

For at sikre optimal performance og lav latenstid mellem server og bruger, har vi anvendt Azure Speed Test, som mäter latency til forskellige Azure-datacentre. På den baggrund har vi valgt det bedst placerede datacenter til vores løsning. Se resultatet i bilag 10.

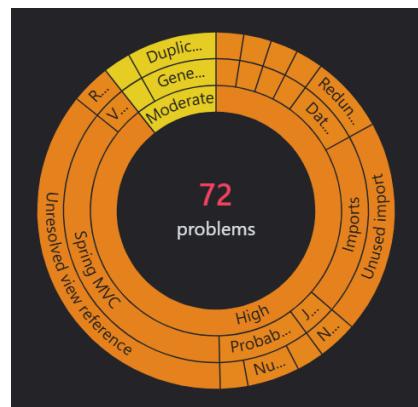
Både Web App og MySQL database er blevet deployet via Azure portal og bindes sammen i CI/CD-processen via application-prod.properties og credentials gemt i Github Secrets.

Derudover er nødvendige environment variables blevet sat i Azure Web App, så Spring automatisk benytter de rette profiler. Se bilag 11 og 12 for screenshots af Web App og MySQL-opsætningen.

Qodana

Qodana er et værktøj fra JetBrains, som bruges til at analysere og kontrollere kode i et projekt. Qodana er en statisk kodeanalyse, der kan finde fejl, dårlig kodestil og andre forbedringer. Vi har integreret Qodana i vores Github Actions CI-pipeline, så den automatisk tjekker koden igennem, hver gang man laver ændringer på projektet.

Ved at bruge Qodana får vi igennem Qodana.cloud et hurtigt overblik over vores kodekvalitet og hvad vi kan forbedre på koden. Det hjælper med at holde koden ren, læsbar, vedligeholdelsesvenlig samt mindske risikoen for fejl i fremtiden.



Figur 50) Statisk kodeanalyse fra qodana.cloud

Som det kan ses på figur 50, så kan vi ud fra den statiske kodeanalyse nemt og hurtigt finde vores fejl. Fejlene opdeles først i en alvorlighedsgrad. I vores kode har Qodana fanget nogle moderate og høje fejl. Efterfølgende opdeles fejlene i kategorier, som f.eks. Spring MVC, Probable Bugs, Imports, Data Flow osv., hvor den til sidst beskriver fejlen. Eksempelvis har vi under kategorien Imports 16 ubrugte importeringer.

```

    Unused import
    import apiasignment.alphasolutions.rowmapper.SkillRowMapper;
    JAVA
    src/main/java/apiassignment/alphasolutions/repository/B1SRepository.java
    8
    9 import apiasignment.alphasolutions.rowmapper.ProjectRowMapper;
    10 import apiasignment.alphasolutions.rowmapper.SkillRowMapper;
    11 import org.springframework.dao.DataAccessException;
    12 import org.springframework.dao.EmptyResultDataAccessException;
    Open file in IntelliJ IDEA More actions Copy link
    Reports redundant import statements.
    Read more
  
```

Figur 51) Udsnit fra listen af problemer i qodana.cloud (ubrugt import)

På figur 51 ses et eksempel på et problem fra qodana.cloud. Qodana stiller et værktøj til rådighed, hvor den viser og kan føre os direkte til den del af koden, som er problemet. På den måde har Qodana været et virkelig effektivt og nemt værktøj at bruge til at få orden på vores kode. Disse figurer er øjebliksbilleder og koden er blevet rettet siden. Vi har anvendt Qodana Code Analysis direkte i IntelliJ til at rette så mange problemer som muligt. Disse fejl omhandlede redundant kode, ubrugte imports, forkert håndtering af exceptions, mulige bugs osv.. Disse problemer er blevet håndteret som det sidste og dermed vil nogle af kodeeksemplerne i denne rapport ikke stemme overens med koden. Status på Qodana static code analysis kan ses i bilag 13.

Brug af LLM (Large language models) som sparringspartner

Under analyse-, design- og udviklingsprocessen har vi aktivt anvendt en LLM (Large Language Model) - herunder ChatGPT - som faglig sparringspartner. AI'en har spillet en rolle i både vores tekniske forståelse, implementering og dokumentation.

I designfasen blev modellen brugt til at opnå dybere indsigt i softwarearkitektur og designmønstre ved at kombinere egne kilder med forklaringer og eksempler fra AI'en. Dette styrkede vores evne til at vælge egne mønstre og begrunde dem i rapporten.

Under implementeringen blev AI anvendt til fejlsøgning og debugging, blandt andet ved analyse af log-filer, forståelse af white-label error-sider og optimering af kode. Modellen har desuden hjulpet med at generere genanvendelige komponenter - eksempelvis styling og sql mock-data - hvilket har effektiviseret den gentagne del af udviklingen.

AI har også spillet en aktiv rolle i opbygningen af rapportens struktur. Den har bidraget til at sikre sammenhæng, klar overgang mellem emner og konsistens i den tekniske dokumentation. Især i skrivefasen har LLM'en fungeret som en værdifuld sparringspartner, der kunne omskrive, forenkle eller formulere komplekse afsnit mere klart.

Den største værdi har vi oplevet ved netop den løbende sparring. Det har dog været afgørende at anvende AI med en kritisk sans. Modellen har ikke kendskab til vores konkrete projekt, faglige niveau eller intentioner - og foreslår derfor løsninger, der ikke altid er relevante eller sikre at implementere. Derfor har det været nødvendigt at indhente viden selvstændigt, stille præcise spørgsmål og validere alle forslag fagligt, før de blev brugt.

Sammenfattende har AI været en hjælp til både læring, struktur og udvikling - men kun i kraft af vores egen stillingtagen, filtrering, holdninger og kritiske vurderinger.

Kørselsvejledning

Vores applikation er deployet til Microsoft Azure, hvor både webapplikationen og databasen (MySQL) hostes. Webapplikationen kan tilgås via følgende link:

<https://g1ssolutions.azurewebsites.net/>.

Ved første besøg kan det tage et par sekunder at loade siden, da applikationen skal initialiseres og browseren endnu ikke har cachet indholdet. Efterfølgende vil load-tiden være kortere.

Afgang til systemet kræver, at man enten har fået tildelt en bruger, oprettet af en administrator eller requested en bruger som skal godkendes af admin. Uden login oplysninger kan man ikke tilgå systemets funktioner.

Lokal opsætning

Hvis man ønsker at køre applikationen lokalt - eksempelvis for at teste performance og udvikle nye features - kan dette også lade sig gøre. Der er en grundig vejledning i projektets:

[README.md](#) - Generel opsætning og teknologioversigt.

[Contribute.md](#) - Retningslinjer for bidrag og miljøopsætning.

Herunder følger en kort opsummering:

Krav:

- En IDE af eget ønske (IntelliJ IDEA, VS eller Eclipse)
 - Dog er det nødvendigt at de understøtter Java, Spring og Thymeleaf.
- MySQL server
 - Applikationen er konfigureret til at køre med MySQL. Det er muligt at anvende andre databaser, men det kræver manuel tilpasning af application.properties.

Sådan kører du applikationen lokalt:

1. Klon github projektet
 - a. git clone "url"
2. Importer projektet i din IDE.
3. Kør SQL-scripts fra src/main/resources/sql for at oprette databasen
4. Konfigurer application.properties med dine lokale databaseoplysninger
5. Kør applikationen fra AlphaSolutionsApplication.java
6. Åbn browser og tilgå <http://localhost:8080>

Konklusion

Formålet med dette projekt har været at udvikle et digitalt værktøj, der understøtter Alpha Solutions i deres behov for mere struktureret og effektiv projektstyring. Gennem hele udviklingsprocessen har vi arbejdet tæt med virksomhedens krav og brugerbehov, hvilket har givet os en solid forståelse for både tekniske og forretningsmæssige aspekter af projektet.

Vi har arbejdet med alle faser af systemudviklingen - fra kravspecifikationen og design til implementering, test og deployment. Systemet er opbygget i Spring Boot med en lagdelt arkitektur og håndterer blandt andet oprettelse og nedbrydning af projekter, estimeringer af tid og tildeling af medarbejdere. Vi har anvendt en MYSQL-database med fuld normalisering, og databasen er deployet sammen med webapplikationen i Microsoft Azure.

Derudover har vi arbejdet professionelt med DevOps-principper, herunder CI/CD via Github Actions, automatiserede test med Maven og integration af statisk kodeanalyse via Qodana. Disse værktøjer har hjulpet os med at sikre høj kvalitet og stabilitet i systemet.

Projektet er løst i et Scrum setup, hvor vi har dokumenteret, arbejdet gennem sprints, backlog, standups og reviews. Undervejs har vi tilpasset løsningen i takt med feedback og nye indsigt - herunder konkrete ønsker fra PO og Alpha Solutions.

Konkluderende mener vi, at vi har leveret en løsning, der både lever op til kundens behov og til de faglige krav, der stilles i et tværfagligt eksamensprojekt. Vi har udviklet et system, der er teknisk robust, forretningsmæssigt relevant og veldokumenteret - og som danner et godt grundlag for videreudvikling.

Litteraturliste

Indledning

Alpha Solutions, n.d. *Digital forretningsrådgivning*. <https://www.alpha-solutions.com/da>
[Lokaliseret d. 25.04.2025]

IT forretningsforståelse

Feasibility Study:

Mind the Graph, n.d. *Fra idé til innovation: Hvad er en feasibility-undersøgelse i forskning*.
<https://mindthegraph.com/blog/da/hvad-er-en-feasibility-undersogelse-i-forskning/>
[Lokaliseret d. 28.04.2025]

ProjectManager.com, n.d. *How to Conduct a Feasibility Study*.
<https://www.projectmanager.com/training/how-to-conduct-a-feasibility-study>
[Lokaliseret d. 28.04.2025]

JetBrains, n.d. *Buy IntelliJ IDEA Ultimate*.
<https://www.jetbrains.com/idea/buy/?section=commercial&billing=yearly>
[Lokaliseret d. 28.04.2025]

Europa-Parlamentet og Rådet, 2016. *EUROPA-PARLAMENTETS OG RÅDETS FORORDNING (EU) 2016/679 af 27. april 2016*.
<https://eur-lex.europa.eu/legal-content/DA/TXT/PDF/?uri=CELEX:32016R0679&from=DA>
[Lokaliseret d. 28.04.2025]

Datatilsynet, 2018. *Behandlingssikkerhed og databeskyttelse gennem design og standardindstillinger*.
https://www.datatilsynet.dk/Media/637689328983143992/Behandlingssikkerhed%20og%20databeskyttelse%20gennem%20design%20og%20standardindstillinger_2018.pdf
[Lokaliseret d. 28.04.2025]

Systemudvikling/teknologi

Den Store Danske, n.d. *Systemudvikling – it*. https://lex.dk/systemudvikling_-_it
[Lokaliseret d. 29.04.2025]

Domain model

Thoughtworks, n.d. *Domain Modeling: What you need to know before coding.*

<https://www.thoughtworks.com/insights/blog/agile-project-management/domain-modeling-what-you-need-to-know-before-coding>

[Lokaliseret d. 30.04.2025]

GeeksforGeeks, n.d. *What is Domain Class in UML?.*

<https://www.geeksforgeeks.org/what-is-domain-class-in-uml/>

[Lokaliseret d. 30.04.2025]

Codeburst, n.d. *Domain Modeling by Example.*

<https://codeburst.io/rule-your-domain-model-d4beae6806c>

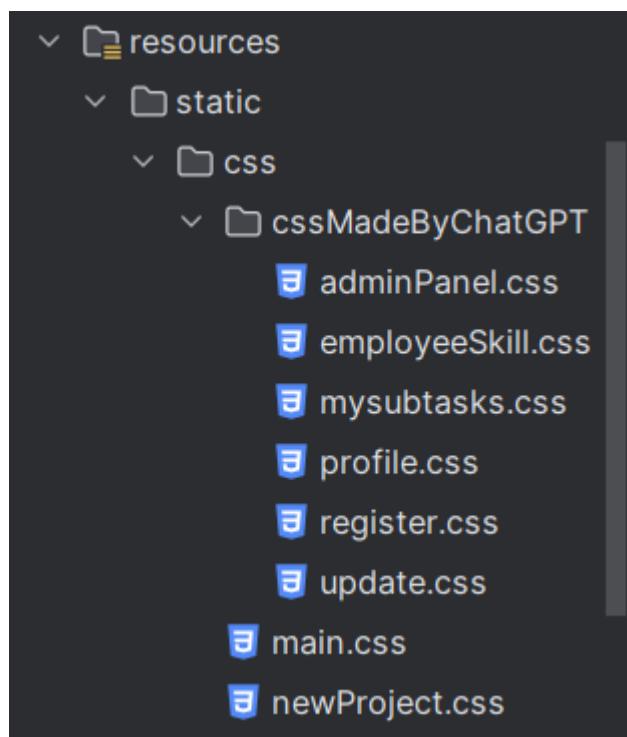
[Lokaliseret d. 30.04.2025]

Pearce, C., n.d. *Domain Modeling.*

<https://www.cs.sjsu.edu/~pearce/modules/lectures/ooa/analysis/DomainModeling.htm>

[Lokaliseret d. 30.04.2025]

Kode



Som det kan ses på ovenstående billede, så er noget af css'en i "main" og "newProject" lavet af gruppen selv, hvorimod de resterende css filer ligger i mappen "cssMadeByChatGPT", dette skyldes, at den css har ChatGPT hjulpet med at generere ud for prompts og billeder af vores Figma.

```
41      //følgende to metoder overrider måden den sammenligner objekter på
42      //dette skulle gøres, så man kunne se en liste over skills, så var brugerens skills allerede tjekket af
43      @Override ▲ huw02
44      public boolean equals(Object o) {
45          if (this == o) return true;
46          if (!(o instanceof Skill)) return false;
47          Skill skill = (Skill) o;
48          return skillId == skill.skillId;
49      }
50
51
52      @Override ▲ huw02
53      public int hashCode() {
54          return Objects.hash(skillId);
55      }
56  }
```

Ovenstående to metoder har vi også fået hjælp til af ChatGPT, da vi stødte på problemer, da vi skulle indlæse en brugers skills i listen over alle skills. Der forklarede ChatGPT, at det kun kunne fikses ved hjælp af følgende metoder, hvor den sammenligner to objekter. Disse to metoder befinder sig i klassen Skill.

Yderligere er ChatGPT også blevet brugt som en effektiv sparringspartner i tilfælde, hvor gruppen ikke har haft mulighed for at hjælpe hinanden.

Reference til den AI der er blevet brugt:

OpenAI. (2025). *ChatGPT (GPT-4o)* [Stor sprogmodel]. Hentet fra <https://chat.openai.com/>

Ovenstående er den gratis version af ChatGPT.

Database

freeCodeCamp, n.d. *Database Normalization – Normal Forms 1NF 2NF 3NF Table Examples.*

<https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/>
[Lokaliseret d. 02.05.2025]

GeeksforGeeks, n.d. *Normal Forms in DBMS*.

<https://www.geeksforgeeks.org/normal-forms-in-dbms/>

[Lokaliseret d. 02.05.2025]

User interface design

Figma, n.d. *What is Wireframing?*.

<https://www.figma.com/resource-library/what-is-wireframing/>

[Lokaliseret d. 06.05.2025]

Levende Streg, n.d. *Gestalt principper*. <https://levendestreg.dk/gestalt-principper/>

[Lokaliseret d. 06.05.2025]

Interaction Design Foundation, n.d. *User Interface Design Guidelines: 10 Rules of Thumb*.

<https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb>

[Lokaliseret d. 06.05.2025]

Scrum

Mannaz, n.d. *Hvad er scrum?*

<https://www.mannaz.com/da/kurser-og-uddannelser/projektledelse/scrum>

[Lokaliseret d. 30.04.2025]

CI/CD

GeeksforGeeks, n.d. *What is CI/CD?*. <https://www.geeksforgeeks.org/what-is-ci-cd/>

[Lokaliseret d. 19.05.2025]

Qodana, n.d. *Qodana Code Quality Report*.

<https://qodana.cloud/projects/kwMVD/reports/40RV1o?genericFilters=N4XyA>

[Lokaliseret d. 22.05.2025]

Viden

BOKBH, n.d. *Sådan skriver du en god problemformulering – Eksempler og skabelon.*
<https://bokbh.dk/saadan-skriver-du-en-god-problemformulering-eksempler-og-skabelon/>
[Lokaliseret d. 28.04.2025]

Studiekorrektur, n.d. *Bilag – Hvordan anvender man bilag i en opgave.*
<https://www.studiekorrektur.dk/bilag>
[Lokaliseret d. 12.05.2025]

Mindrot, n.d. *jBCrypt – strong password hashing for Java.*
<https://www.mindrot.org/projects/jBCrypt/>
[Lokaliseret d. 19.05.2025]

DZone, n.d. *Password Encryption and Decryption Using jBCrypt.*
<https://dzone.com/articles/password-encryption-and-decryption-using-bcrypt>
[Lokaliseret d. 19.05.2025]

Twilio, n.d. *SendGrid Docs.* <https://www.twilio.com/docs/sendgrid#send-your-first-email>
[Lokaliseret d. 15.05.2025]

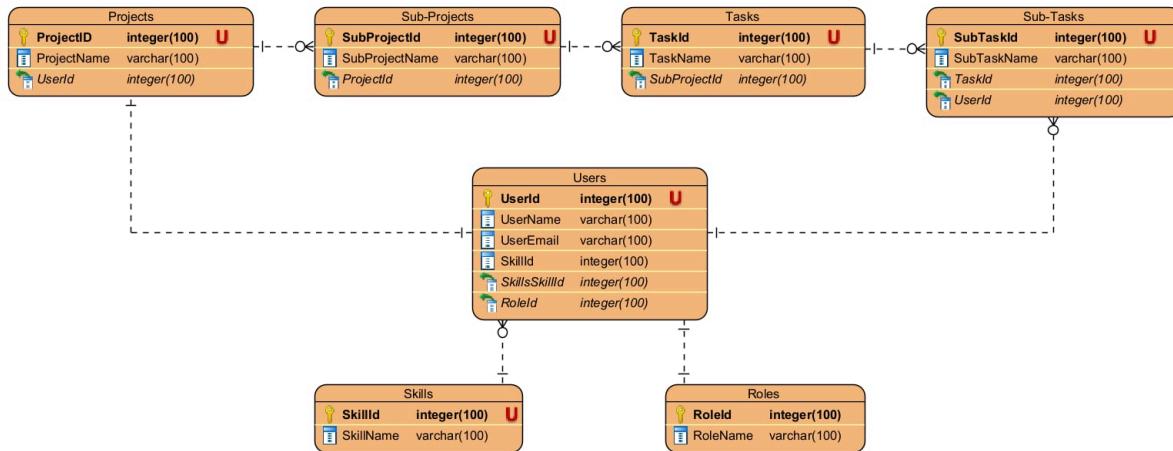
Dan Vega, n.d. *danvega/sendgrid: Java + SendGrid.* <https://github.com/danvega/sendgrid>
[Lokaliseret d. 15.05.2025]

YouTube, 2022. *How to Send Emails in Java Using SendGrid (Beginner Tutorial).*
<https://www.youtube.com/watch?v=i8Hvvo4ZITg>
[Lokaliseret d. 15.05.2025]

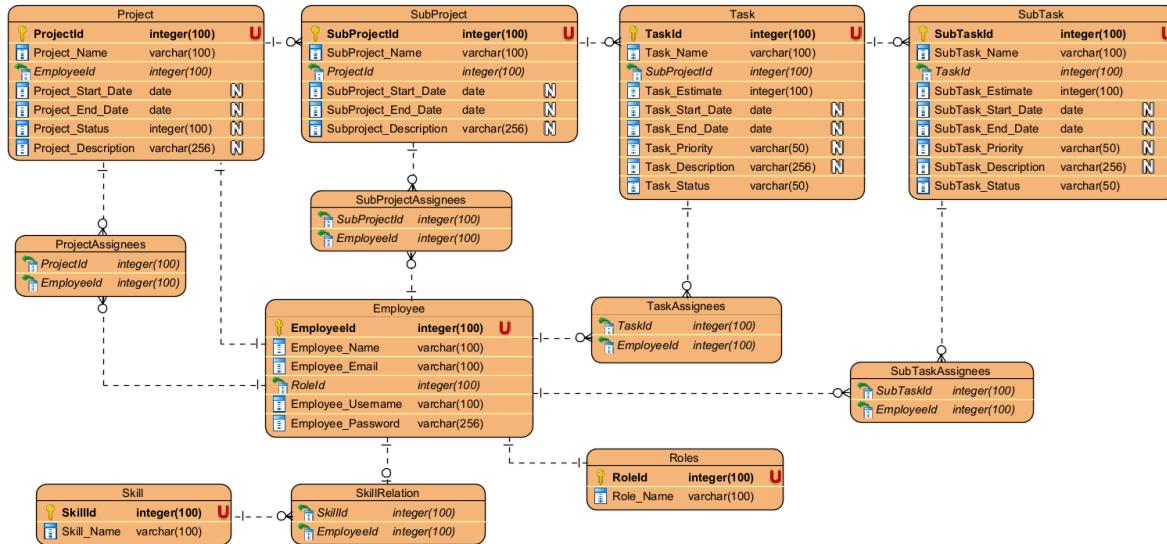
YouTube, 2023. *Spring Boot Environment Variables Explained (Secure Your API Keys!).*
<https://www.youtube.com/watch?v=rQV76dufxz4>
[Lokaliseret d. 12.05.2025]

Bilag

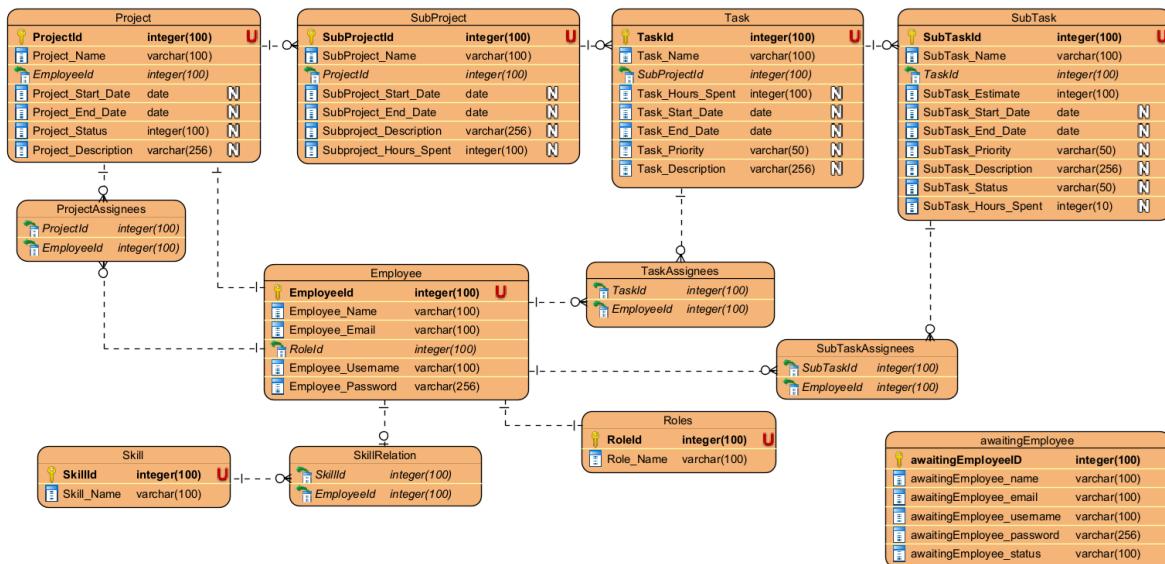
Bilag 1) Tidlig ER-model



Bilag 2) ER-model under sprint 1



Bilag 3) ER-model efter sprint 2



Bilag 4) User stories

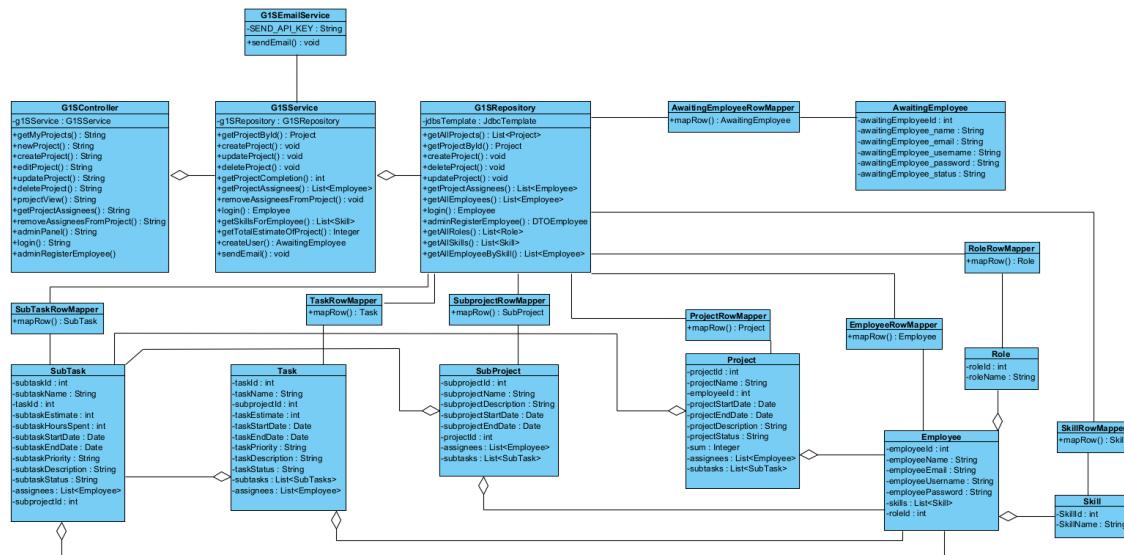
<p>Som projektleder skal jeg kunne se, oprette, slette og redigere projekter for at kunne få et bedre overblik over de projekter jeg har.</p>	<p>Jeg skal trykke på en tilføj knap og skal kunne skrive det ønskede navn på projektet.</p> <p>Jeg skal kunne trykke på en slet knap på mine projekter.</p> <p>Jeg skal kunne trykke på en rediger knap ud fra mine projekter.</p>
<p>Som projektleder skal jeg kunne se, oprette, redigere og slette sub-projekter, så jeg har overblik og struktur over de projekter, jeg har oprettet</p>	<p>Jeg skal trykke på en tilføj knap og skal kunne skrive det ønskede navn på sub-projektet.</p> <p>Jeg skal kunne trykke på en slet knap på mine sub-projekter.</p> <p>Jeg skal kunne trykke på en rediger knap ud fra mine sub-projekter</p>

Som projektleder skal jeg have et overblik over hvor mange timer der er til rådighed, brugt og tilbage i et projekt, så jeg ved om projektet overholder dets dedikerede antal timer.	Jeg skal have mulighed for at se antallet af brugte timer på mine projekter uden at skulle regne på det selv.
Som medarbejder vil jeg kunne oprette, redigere og slette tasks til sub-projekterne samt angive og ændre status, så de givne tasks altid er opdaterede.	<p>Efter at tilgå sub-projektet skal jeg kunne trykke på en knap hvor jeg kan oprette task med navn etc.</p> <p>Der skal være en knap som jeg kan trykke på der sletter task.</p> <p>Der skal være en knap, jeg kan trykke på som redigere en task.</p> <p>Der skal være en status, knap jeg kan trykke på som redigerer status på den given task.</p>
Som medarbejder vil jeg gerne kunne oprette, redigere samt angive og ændre status på sub-tasks, så jeg bedre kan få et overblik over, hvad status er på en sub-task.	<p>Efter at tilgå sub-projektet skal jeg kunne trykke på en knap hvor jeg kan oprette sub-task med navn etc.</p> <p>Der skal være en knap som jeg kan trykke på der sletter sub-task.</p> <p>Der skal være en knap, jeg kan trykke på som redigerer en sub-task.</p> <p>Der skal være en status knap jeg kan trykke på som redigere status på den given sub-task</p>

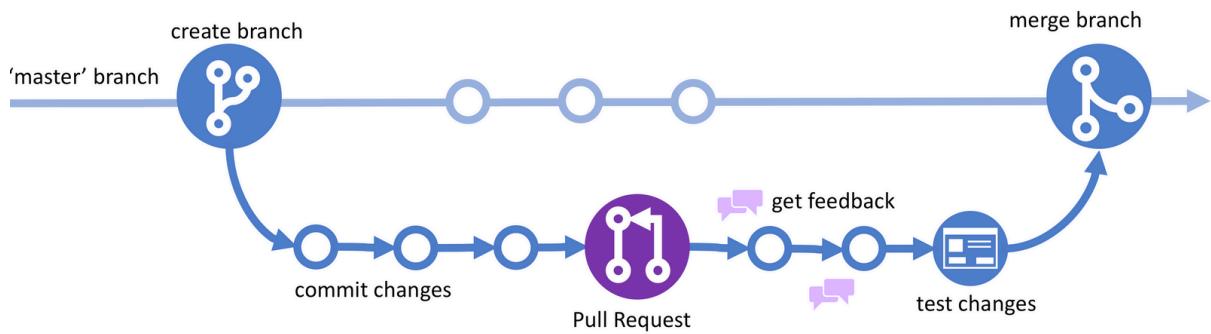
Som admin skal jeg kunne tilføje og slette projektledere og medarbejdere, så de givne brugere kan gøre deres arbejde.	Ved at trykke på opret bruger knap, kan der oprettes nye medarbejdere.
Som admin skal jeg kunne slette projekter, sub-projekter, task og sub-tasks, så jeg ikke har unødvendige opgaver i projektet.	Der skal være en knap som der kan trykkes på som viser alle projekter der er oprettet i systemet. Der skal være en knap ud fra alle givne projekter som kan slette dem. Når der trykkes på knappen slettes projektet.
Som projektleder vil jeg kunne tilføje en medarbejder til et projekt, så de kan arbejde med projektet.	Der skal være en knap som der kan trykkes på ud fra hvert projekt som sender hen til en side med medarbejdere der kan tilføjes til projektet.
Som projektleder vil jeg kunne se en liste over medarbejdere sorteret ud fra kompetencer, så jeg hurtigere kan finde den rette kandidat til en given opgave.	Der skal være en knap jeg kan trykke på som sortere medarbejderne ud fra den ønskede kompetence jeg leder efter. Der skal være en tilføj til et projekt knap som tilføjer den ønskede medarbejder til projektet.
Som projektleder vil jeg gerne kunne sortere mine sub-projekter, tasks og sub-tasks ud fra prioritering, så jeg bedre kan få et overblik over de forskellige opgaver.	Der skal være en knap, jeg kan trykke på som sortere liste med opgaver ud fra prioritering.
Som projektleder vil jeg have et overblik over, hvornår mine projekter nærmer sig deadline, så vi kan holde projektet indenfor dens tidsmæssige rammer.	Der skal implementeres en funktionalitet, så man kan give opgaver en deadline for, hvornår de skal være færdige.

Som projektleder vil jeg kunne se et gantt diagram over mit projekt, så jeg kan se hvilke opgaver der tager flest ressourcer	Der skal implementeres en 'se diagram' knap, som sender en hen til en side, hvor der bliver dannet et gantt diagram ud fra et givent projekts data
Som projektleder vil jeg have et overblik over, hvor mange kilowatt samt CO2 et givent projekt udleder, så vi kan se, hvor mange træer der skal plantes.	Der skal implementeres funktionalitet, som på baggrund af mængden af timer brugt på et projekt fortæller hvor mange kilowatt el og CO2, der er blevet brugt på det givne projekt.
Som projektleder vil jeg kunne se, hvor mange træer der skal plantes, for at et projekt har været neutralt i forhold til CO2 udledningen, så vi kan holde et CO2 neutralt projekt.	Der skal implementeres en funktionalitet der beregner antallet af træer der skal plantes for, at man beholder et neutralt CO2 udslip, dette baseres på antal timer brugt på projektet.

Bilag 5) Klassediagram



Bilag 6) Github branch feature set up



Bilag 7) Github Action fejl

Summary

Triggered via push 3 days ago
Huw02 pushed → d9421f5 main

Status: Failure | Total duration: 23s | Artifacts: -

main_g1ssolutions.yml

on: push

build 19s → deploy 0s

Annotations

1 error

build: Process completed with exit code 1.

Bilag 8) CI/CD maven fil deploy til azure

AlphaSolutions / .github / workflows / main_g1solutions.yml

Code Blame 66 lines (54 loc) · 1.89 KB

Raw ▾

```
3
4 name: Build and deploy JAR app to Azure Web App - g1solutions
5
6 on:
7   push:
8     branches:
9       - main
10    workflow_dispatch:
11
12 jobs:
13   build:
14     runs-on: ubuntu-latest
15     permissions:
16       contents: read #This is required for actions/checkout
17
18     steps:
19       - uses: actions/checkout@v4
20
21       - name: Set up Java version
22         uses: actions/setup-java@v4
23         with:
24           java-version: '21'
25           distribution: 'microsoft'
26
27       - name: Build with Maven
28         run: mvn clean install
29
30       - name: Upload artifact for deployment job
31         uses: actions/upload-artifact@v4
32         with:
33           name: java-app
34           path: '${{ github.workspace }}/target/*.jar'
35
36 deploy:
37   runs-on: ubuntu-latest
38   needs: build
39   environment:
40     name: 'Production'
41     url: ${{ steps.deploy-to-webapp.outputs.webapp-url }}
42   permissions:
43     id-token: write #This is required for requesting the JWT
44   contents: read #This is required for actions/checkout
45
46   steps:
47     - name: Download artifact from build job
48       uses: actions/download-artifact@v4
49       with:
50         name: java-app
51
52     - name: Login to Azure
53       uses: azure/login@v2
54       with:
55         client-id: ${{ secrets.AZUREAPPSERVICE_CLIENTID_B690F78BC13147B6B63CB036F529AB07 }}
56         tenant-id: ${{ secrets.AZUREAPPSERVICE_TENANTID_9022D62112A648839C55F73A775CB380 }}
57         subscription-id: ${{ secrets.AZUREAPPSERVICE_SUBSCRIPTIONID_81B907A4828C4DFB812FAC4EA881071 }}
58
59     - name: Deploy to Azure Web App
60       id: deploy-to-webapp
61       uses: azure/webapps-deploy@v3
62       with:
63         app-name: 'g1solutions'
64         slot-name: 'Production'
65         package: '*.jar'
```

Bilag 9) Java CI with maven

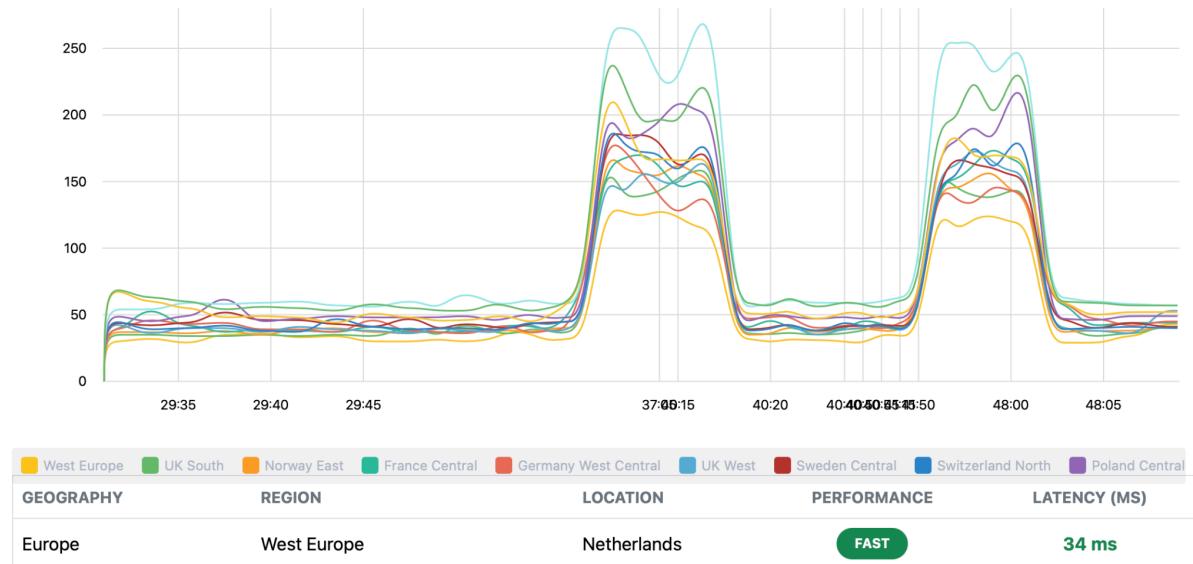
```
Code Blame 25 lines (19 loc) · 404 Bytes

1   name: Java CI with Maven
2
3   on:
4     push:
5       branches: [ "main" ]
6     pull_request:
7       branches: [ "main" ]
8
9   jobs:
10    build:
11
12      runs-on: ubuntu-latest
13
14    steps:
15      - uses: actions/checkout@v4
16      - name: Set up JDK 21
17        uses: actions/setup-java@v4.3.0
18        with:
19          java-version: '21'
20          distribution: 'temurin'
21          cache: maven
22      - name: Build with Maven
23        run: mvn -B test
24
```

Bilag 10) Azure latency test:

<https://www.azurespeed.com/Azure/Latency>

Azure Latency Test Results



Bilag 11) G1Ssolution web app

The screenshot shows the Azure portal interface for the 'g1solutions' web application. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Recommended services (preview), Resource visualizer, Deployment (Deployment slots, Deployment Center), Settings (Environment variables, Configuration, Authentication, Identity, Backups, Custom domains, Certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), WebJobs), and Application Insights.

Essentials

Resource group (move)	: AlphaSolutions	Default domain	: g1solutions.azurewebsites.net
Status	: Stopped	App Service Plan	: ASP-AlphaSolutions-bfbf (F1: 1)
Location (move)	: West Europe	Operating System	: Linux
Subscription (move)	: Azure for Students	Health Check	: Cannot determine health check status while app is in a "Stopped" state.
Subscription ID	: aed11d41-cd7e-4af1-be01-e82b1c89b6e9	GitHub Project	: https://github.com/Zahawis/AlphaSolutions

Tags (edit)

Properties

Name	g1solutions	Deployment Center
Publishing model	Code	Deployment logs
Runtime Stack	Java 21 SE	Last deployment

Domains

Default domain	: g1solutions.azurewebsites.net	Application Insights
Custom domain	: Add custom domain	Name

Hosting

Plan Type	App Service plan	Networking	
Name	ASP-AlphaSolutions-bfbf	Virtual IP address	: 20.105.232.33
Operating System	Linux	Outbound IP addresses	: 50.85.7.197, 108.141.66.113, 57.153.132... Show More
Instance Count	1	Additional Outbound IP addresses	: 50.85.7.197, 108.141.66.113, 57.153.132... Show More
SKU and size	Free (F1) Scale up	Virtual network integration	: Not supported

Bilag 12) G1Solution mySQL database

The screenshot shows the Azure portal interface for the 'g1solutions' Azure Database for MySQL flexible server. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Learning center, Resource visualizer, Settings (Compute + storage, Connect, Server parameters, Replication, Maintenance, High availability, Backup and restore, Advisor recommendations, Locks), Power Platform, Security, Monitoring, Automation, and Help.

Essentials

Subscription (move)	: Azure for Students	Server name	: g1solutions.mysql.database.azure.com
Subscription ID	: aed11d41-cd7e-4af1-be01-e82b1c89b6e9	Administrator login	: g1solutions
Resource group (move)	: AlphaSolutions	Configuration	: Bustable_B1ms_1 vCores, 2 GiB RAM, 20 storage, 350 IOPS
Status	: Stopped	MySQL version	: 8.0
Location	: France Central	Availability zone	: --
		Created on	: 2025-05-23 08:39:39.8671222 UTC

Tags (edit)

Properties

Getting started	Properties	Recommendations	Monitoring	Tutorials
-----------------	------------	-----------------	------------	-----------

Compute + storage (Change)

Pricing tier	Burstable	Networking (Change)
Compute size	Standard_81ms (1 vCore, 2 GiB memory, 640 max iops)	Connectivity method
Storage	20 GiB	Virtual network
IOPS	Auto scale IOPS	Public access (allowed IP addresses)
Accelerated Logs	Disabled	Not configured
Storage autogrow	Enabled	

Backup (Change)

Backup type	Locally redundant	High availability (Change)
Retention period	7	High availability
Earliest restore point	2025-05-23 08:49:39.8671222 UTC	Not enabled

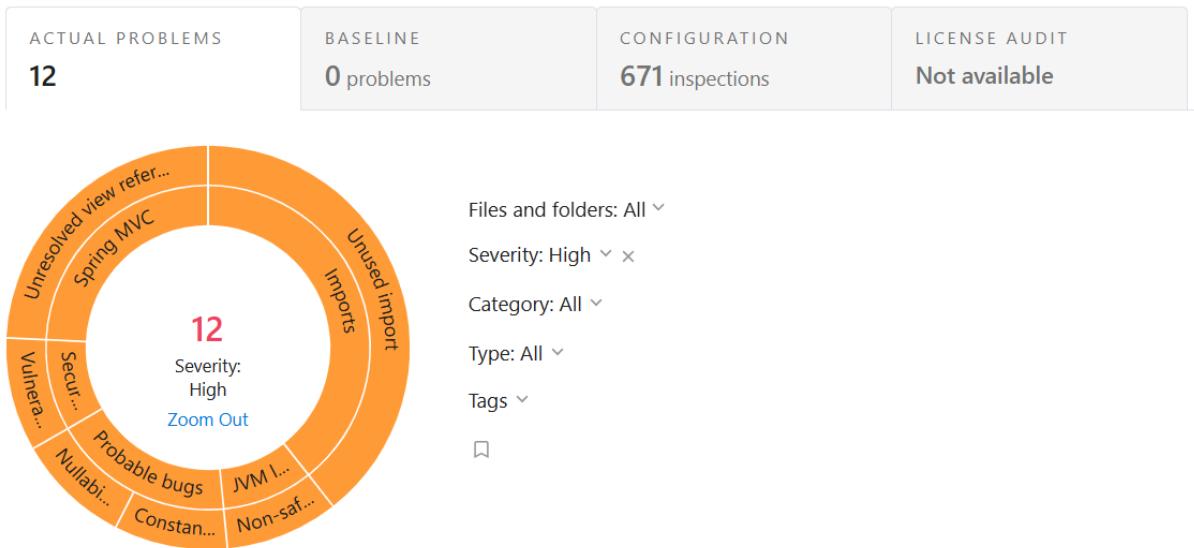
Maintenance (Change)

Maintenance	System-managed schedule
-------------	-------------------------

Replication (Change)

Replication role	None
Replicas	Not configured

Bilag 13) Qodana static code status



Daily scrum

05-05-2025

Daily scrum rapport - Simon:

• **Hvad jeg lavede sidst:**

Sidste gang vi arbejdede, startede jeg på user story #4, hvor jeg skulle lave et CRUD program for projekterne. Jeg har lavet koden så kun den projektleder, som opretter projektet samt de collaborators han/hun vælger til projektet, kan se selve projektet, når de logger ind på deres profil. Det er også muligt at slette og opdatere projekter.

• **Hvad jeg skal lave i dag:**

I dag skal jeg lave koden lidt om, så det kun er projektleder, som kan slette og redigere projekterne han/hun har lavet. Jeg skal også lave de tilhørende HTML filer om samt tilføje CSS, så de ligner de ønskede sider. Derefter skal det merges ind i mainen med en pull request.

• **Forhindringer:**

Der har været små problemer ind i mellem. Men ikke noget, som jeg ikke har kunnet løse selv ved at debugge.

Daily Scrum Rapport - Hannibal

- **Hvad jeg lavede sidst:**

Jeg arbejdede sidste gang med at få lavet et login siden, hvor jeg endte med at blive færdig med alt backend, så der mangler kun at blive stylet med css.

Jeg gik sidste gang også i gang med at lave admin panel og dets CRUD metoder.

- **Hvad jeg skal lave i dag:**

Jeg vil i dag arbejde videre med admin panelet og lave dets CRUD metoder færdigt.

Jeg skal også få merget mit arbejde med main.

- **Forhindringer:**

Der har ikke været nogen forhindringer endnu.

Daily Scrum Rapport - Victor

- **Hvad jeg lavede sidst:**

Oprettet "home.html" og udviklet "dashboard.html" som er et fragment vi vil bruge på alle html-sider som navigationsværktøj. Dette fragment er rart at have færdigt hurtigt under udviklingen så vi nemt og hurtigt kan navigere rundt.

- **Hvad jeg skal lave i dag:**

Implementere de korrekte endpoint paths i dashboardet når vi nærmer os dagens afslutning. Der har folk formentlig pushet og oprettet flere html-dokumenter, så dashboardet kan linkes op til dem og bliver funktionelt. I mellemtiden skal der arbejdes videre med user story/issue #7.

06-05-2025

Daily scrum rapport - Simon:

- **Hvad jeg lavede sidst:**

Jeg har lavet backenden til at kunne tilføje collaborators til et projekt. Man kommer ind på en liste over alle medarbejdere samt deres skills, hvorefter man afslutter og medarbejderne optræder på oversigten over projektet. Jeg har også fået lavet html og css til oprettelse af et nyt projekt.

- **Hvad jeg skal lave i dag:**

I dag skal jeg lave html og css til listen af projekterne og for listen af medarbejdere, som man ønsker at tilføje til et givent projekt.

- **Forhindringer:**

Vores mockup på Figma for at lave et nyt projekt ønskede vi at kunne tilføje collaborators direkte i formen. Men det har ikke kunne lade sig gøre, som vi forestillede os uden at bruge JavaScript

Daily Scrum Rapport - Hannibal

- **Hvad jeg lavede sidst:**

Jeg lavede admin panel og dets crud metoder færdigt, så user story login og admin panel er begge færdige. Jeg fik også merget det med main.

- **Hvad jeg skal lave i dag:**

style mine sider, når det er færdigt, så starter jeg på at kunne tilføje medarbejder til et projekt

- **Forhindringer:**

Daily Scrum Rapport - Victor

- **Hvad jeg lavede sidst:**

Flere tilføjelser til dashboard og /subproject/id funktionaliteter samt frontend og design.

- **Hvad jeg skal lave i dag:**

Videre med CRUD funktionalitet på task og subtasks inde i et subprojekt

- **Forhindringer:**

Daily Scrum Rapport - Zahaa

- **Hvad jeg lavede sidst:**

Jeg fik sat test database, database og prod database, indsat mock data og tilføjede det til databasen. Fulgt min user story og lavet subproject entity, lavet metoder og SQL queries.

- **Hvad jeg skal lave i dag:**

Skal fortsætte med at lave mine task for min user pages.

- **Forhindringer:**

Jeg har heldigvis ikke haft nogle udfordringer med mine user stories eller task.

Det er alt sammen kørt relativt smurt og godt.

07-05-2025

Daily scrum rapport - Simon:

- **Hvad jeg lavede sidst:**

Jeg fik lavet my.projects.html filen færdig med css. Derudover fik jeg lavet en addRoleAttributesToModel metode i controlleren, som sender roleId til html filerne. Dermed kan man på dashboardet filtrere mulighederne ud fra roleId på den bruger, som er logget ind.

- **Hvad jeg skal lave i dag:**

I dag skal jeg få et overblik over scrumprocessen indtil videre og dokumentere det, så det kan tilføjes til rapporten.

- **Forhindringer:**

Daily Scrum Rapport - Hannibal

- **Hvad jeg lavede sidst:**

Jeg gik i gang med at arbejde på at kunne sortere i medarbejder ud for skills

- **Hvad jeg skal lave i dag:**

Jeg skal lave funktionen færdig, hvor man sortere efter skills og tilføjer medarbejder til et projekt, derudover skal jeg lave test

- **Forhindringer:**

Daily Scrum Rapport - Victor

- **Hvad jeg lavede sidst:**

Fixet delete funktionalitet, UI, implementeret “cascading delete” i vores SQL database, så delete funktionalitet bliver nemt når en tabel har foreign key constraints.

- **Hvad jeg skal lave i dag:**

Videre med edit task/subtask funktionalitet.

- **Forhindringer:**

Daily Scrum Rapport - Zahaa

- **Hvad jeg lavede sidst:**

Jeg fik færdiggjort min user story og alle task under det. Det er blevet implementeret og pushed til main. Derudover har jeg opdateret mine assigned task under projects, så det er ajourført.

- **Hvad jeg skal lave i dag:**

I dag skal jeg køre unit test af diverse endpoint controller for at se om vores controllere er oprettet korrekt og er testable.

- **Forhindringer:**

Jeg har heldigvis ikke haft nogle udfordringer. Det har bare kørt som det skulle.

12-05-2025

Daily scrum rapport - Simon:

- **Hvad jeg lavede sidst:**

Siden sidst har jeg lavet integrationstest på alle metoderne tilknyttet project.

- **Hvad jeg skal lave i dag:**

I dag skal jeg starte på mine user stories til sprint 2. Jeg skal blandt andet lave en metode, som automatisk opdatere status på et projekt ud fra status på sub-projekt, task og subtask. Jeg skal også gøre det muligt at kunne tilføje assignees til et projekt.

- **Forhindringer:**

Daily Scrum Rapport - Hannibal

- **Hvad jeg lavede sidst:**

Jeg lavede alt funktionalitet færdig, i forhold til at tilføje folk til et projekt og sortere dem ud for skills

- **Hvad jeg skal lave i dag:**

i dag skal jeg gøre så man kan indsætte skills på folk når man opretter og opdatere dem, derudover skal jeg arbejde videre med test og styling

- **Forhindringer:**

Daily Scrum Rapport - Victor

- **Hvad jeg lavede sidst:**

Edit og create task/subtask funktionalitet færdig samt styling.

- **Hvad jeg skal lave i dag:**

Integrations- og unittest, samt begynde på implementering af progress bars.

- **Forhindringer:**

Daily Scrum Rapport - Zahaa

- **Hvad jeg lavede sidst:**

Jeg fik færdiggjort alt der skulle implementeres og testes af mine metoder og klasser og opdateret mine user stories og task.

- **Hvad jeg skal lave i dag:**

I dag skal jeg lave min user story færdig og få testet, om de virker.

- **Forhindringer:**

Jeg har heldigvis ikke været ude for nogle forhindringer, der har påvirket mit arbejde som sådan. Der har været nogle små ting, som jeg lige skulle læse lidt om før jeg kunne færdiggøre det.

13-05-2025

Daily scrum rapport - Simon:

- **Hvad jeg lavede sidst:**

Siden sidst har jeg fået lavet det meste af mine user stories færdigt i sprint 2. Man kan nu tilføje medarbejder til et projekt, hvor man efterfølgende kan tilføje de medarbejdere til en task eller subtask under det specifikke projekt.

- **Hvad jeg skal lave i dag:**

I dag skal jeg lave de sidste detaljer færdigt. Jeg skal blandt andet lave en side, hvor man kan se alle medarbejderne, som er tilknyttet et projekt.

- **Forhindringer:**

Daily Scrum Rapport - Hannibal

- **Hvad jeg lavede sidst:**

I går fik jeg lavet min user story, hvorpå man kunne tilføje skills til en employee, når man oprettede dem og når man opdaterede deres profil. Derudover arbejdede jeg videre med styling af de forskellige sider

- **Hvad jeg skal lave i dag:**

I dag skal jeg lave mere styling, arbejde på rapport og måske begynde på min nye user story

- **Forhindringer:**

Daily Scrum Rapport - Victor

- **Hvad jeg lavede sidst:**

Integrationstest på create/read/delete task/subtask, mangler update.

Omrokeret noget logik i programmet. Det er nu kun subtask som indeholder estimate, status og actual_hours, men task, subproject, project bruger disse data til at vise akkumulerede værdier af disse.

- **Hvad jeg skal lave i dag:**

- **Forhindringer:**

Dokumentering af daily scrum

Efter vores daily scrum, den 13-05-2025, begyndte vi at undlade at skrive vores scrum møder ned. Dette skyldtes, at vi følte der blev afsat for meget tid på at dokumentere vores møder. Vi mente i stedet, at det var nok blot at tale om dem og ikke dokumentere dem.