LKBH § 107 – Bookning af vikar tider

2. semester eksamensprojekt (Datamatiker)

A

Jon Lundby Nielsen (09-08-1985)

GitHub username - JonLundby

Magnus Carstensen (19-09-1992)

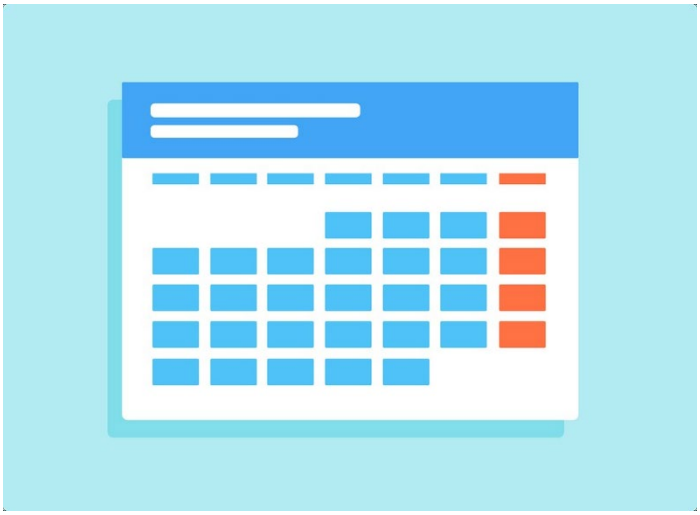
GitHub username - Elvasfar

Markus Ingerslev Olsen (20-04-2001)

GitHub username - MarkusIngeslev

Palle Jensen (02-06-1987)

GitHub username - PalleGregersJensen



GitHub frontend repository:

https://github.com/MarkusIngerslev/LKBH-semester-projekt-frontend

GitHub backend repository:

https://github.com/MarkusIngerslev/LKBH-semester-projekt-backend

Deployet frontend:

https://markusingerslev.github.io/LKBH-semester-projekt-frontend/

Backend REST-API

https://lkbh-semester-projekt-backend.azurewebsites.net

Gruppe – JMMP

Dat23V2 (Datamatiker)

Kea – Københavns Erhvervsakademi

Afleveret 15-12-2023

Indhold

Indledning inkl. problemformulering ... 3
Diagrammer ... 4
Usecases ... 4
Klasse-diagram ... 8
ER-diagram ... 9
Substitutes Table ... 9
Shifts Table ... 10
ShiftInterest Table ... 10
Analyser ... 10
Risikoanalyse ... 10
Risk-management strategi ... 10
Identificering af risici, sandsynlighedsvurdering af risici og risikofaktorernes konsekvens ... 11
Risikoovervågning ... 12
Udvidet risikotabel ... 12
Feasibility study ... 14
Teknisk gennemførlighed ... 14
Økonomisk gennemførlighed ... 14
Lovmæssig gennemførlighed ... 15
Operationel gennemførlighed ... 15
Planlagt gennemførlighed ... 15
Politisk gennemførlighed ... 16
Teknologier ... 16
Azure ... 16
Github ... 17
Beskrivelse af kode ... 17
Hjemmeside (Frontend) ... 17
NodeJS (Backend) ... 19

Indledning inkl. problemformulering

Firmaet LKbh beskæftiger sig med at tilbyde et botilbud under servicelovens §107 samt ADL-træning (almindelig daglig levevis). Deres målgruppe er voksne i alderen 18-50 år, der har diagnoser inden for ASF (Autisme Spektrum Forstyrrelser), OCD og angst.

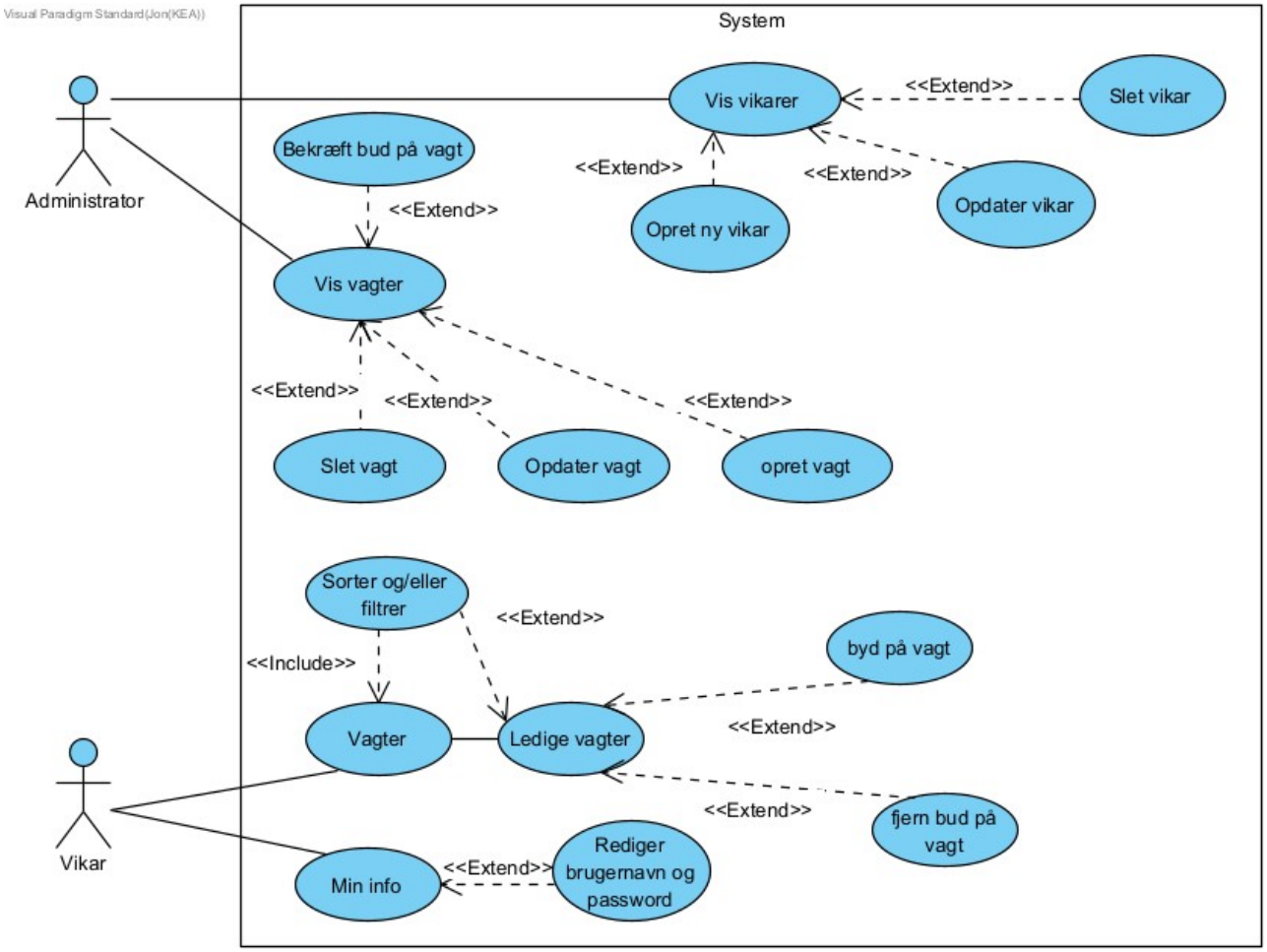
I dette projekt samarbejder vi med LKbh for at skabe en ny løsning – et bookingsystem for firmaets vikarer. I øjeblikket sendes der en SMS-besked ud til alle vikarer for at høre, om de har mulighed for at tage en ledig vagt. Vi vil implementere en løsning, hvor en ansvarlig driftsmedarbejder i stedet kan gå ind på en hjemmeside og indtaste de vagter, der i øjeblikket er ledige. Dette giver de forskellige vikarer i firmaet mulighed for at byde på de forskellige ledige vagter efter deres ønsker. Den driftsansvarlige har derefter mulighed for at godkende de forskellige vagter i overensstemmelse med vikarernes ønsker.

Formålet med denne løsning er at gøre det nemmere for både den driftsansvarlige og de forskellige vikarer at få et overblik over deres vagter samt identificere eventuelle tidspunkter, hvor der mangler dækning. Vi forventer også, at dette vil reducere den tid, der bruges på at planlægge vagter.

Diagrammer

Usecases

I projektet bruger vi usecases til at skabe os både et overblik, men også til at få en ide om hvad det er kunden godt kunne tænke at vores produkt gjorde. Nedenfor har vi et diagram for hvilke muligheder der er på hjemmesiden.



Vi har efterfølgende lavet en Use Cases (breif) for de forskellige tilfælde:

Use case titel / ID	Admin får vist liste af vikarer / UC001
fremgangsmetode	1. Admin logger ind på websiden. 2. Systemet verificerer admin. 3. Systemet fetcher data med vikarer og genererer liste 4. Systemet viser liste med vikarer. 5. slut.

Use case titel / ID	Admin opretter ny vikar / UC002
fremgangsmetode	1. Admin navigerer til visning af vikar liste. 2. Admin trykker på opret ny vikar. 3. Systemet viser opret ny vikar formular. 4. Admin udfylder formular og trykker opret/bekræft. 5. Systemet dobbeltjekker bekræftigelse. 6. Hvis formular er valid: systemet opretter ny vikar i database. 7. ellers: system viser fejl besked. 8. Slut.
Use case titel / ID	Admin får vist specifik vikar / UC003 (AFLYST)
Steps involved	1. Admin navigerer til vikar liste. 2. Admin vælger/trykker på specifik vikar. 3. System shows detail view. (AFLYST)
Use case titel / ID	Admin sletter specifik vikar / UC004
fremgangsmetode	1. Admin navigerer til specifik vikar. 2. Admin trykker slet 3. Systemet spørger om bekræftigelse 4. Hvis admin bekræfter: - Systemet sletter specifik vikar (baseret på id) 5. ellers: - Systemet viser fejlbesked 6. Slut.
Use case titel / ID	Admin opdaterer specifik vikar / UC005

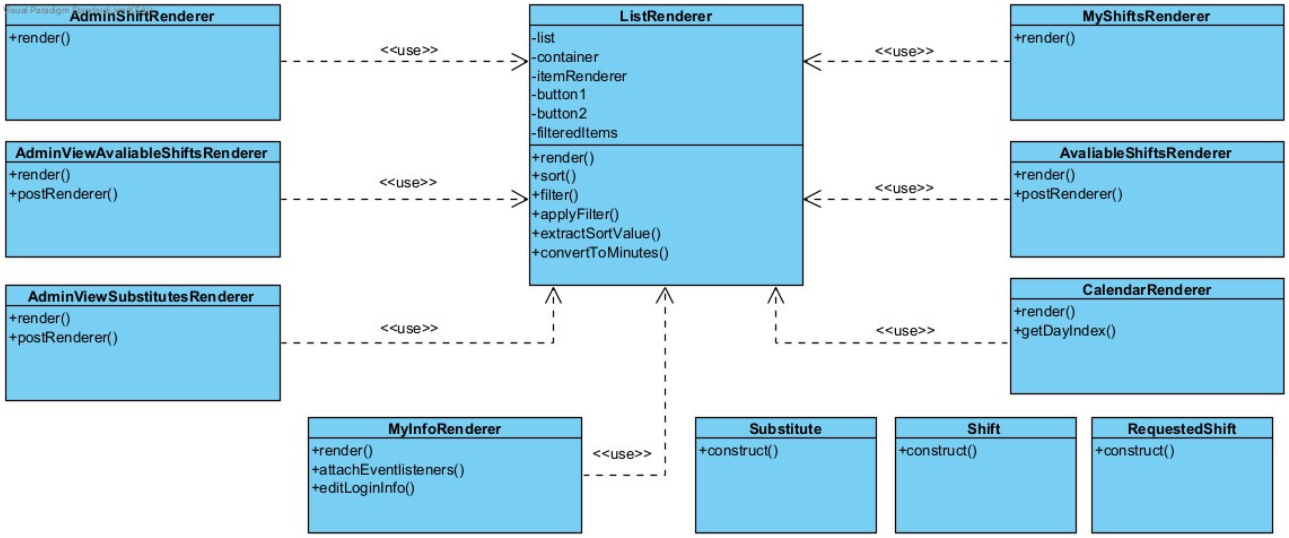
fremgangsmetodess	1. Admin navigerer til specifik vikar og trykker opdater. 2. Systemet viser opdaterings formular med detaljer fra den valgte vikar. 3. Admin laver ændringer i formular og trykker opdater. 4. Systemet spørger om bekræftigelse. 5. Hvis: formular er valid - Systemet skriver den opdaterede vikar til databasen. 6. ellers: - systemet viser fejlbesked. 7. End.
Use case titel / ID	Admin får visning af liste med vagter / UC006
Fremgangsmetode	1. Admin logger ind på websiden. 2. Systemet verificere admins log ind. 3. Systemet fetcher data for vagter og genererer liste 4. Systemet viser liste med vagter. 5. slut.
Use case titel / ID	Admin opretter ny vagt / UC007
fremgangsmetode	1. Admin navigerer til visning af vagt liste. 2. Admin trykker på opret ny vagt. 3. Systemet viser opret ny vagt formular. 4. Admin udfylder formular og trykker opret/bekræft. 5. Systemet dobbeltjekker bekræftigelse. 6. Hvis formular er valid: systemet opretter ny vagt i database. 7. ellers: system viser fejl besked. 8. Slut.
Use case titel / ID	Admin opdaterer vagt / UC008
fremgangsmetode	1. Admin navigerer til specifik vagt og trykker opdater. 2. Systemet viser opdaterings formular med detaljer på den valgte vagt. 3. Admin ændre formular og trykker opdater. 4. Systemet spørger om bekræftigelse. 5. Hvis: formular er valid

	- systemet gemmer opdateret vagt I databasen. 6. else: - systemet viser fejlbesked. 7. Slut.
Use case titel / ID	Admin sletter vagt / UC009
fremgangsmetode	1. Admin navigerer til specifik vagt og trykker slet. 2. Systemet spørger om bekræftigelse. 3. Hvis: admin bekræfter. - Systemet sletter vagt. 4. Ellers: - systemet viser fejlbesked. 5. Slut.
Use case titel / ID	Vikar får visning af vagter/ UC010
fremgangsmetode	1. Vikar logger ind på websiden. 2. Systemet verificere vikar. 3. Systemet fetcher data med vagter og genererer liste 4. Systemet viser liste med vagter. 5. Slut.
Use case titel / ID	Vikar får sortere + filtrere af vagter / UC011 + UC012
fremgangsmetode	1. Vikar navigerer til visning af liste med vagter. 2. Vikar vælger sortering eller filtrerings muligheder. 3. system viser listen på ny i sorteret og/eller filtreret tilstand 4. Slut.
Use case titel / ID	Vikar byder på ledig vagt / UC013
fremgangsmetode	1. Vikar navigerer til specifik vagt og trykker ”byd på vagt”. 2. System opdaterer vagten i vagt databasen. 3. Slut.

Use case titel / ID	Vikar fjerner bud på vagt/ UC014
fremgangsmetode	1. Vikar navigerer til specifik vagt og trykker fjern bud. 2. systemet opdaterer vagten i vagt databasen. 3. Slut.
Use case titel / ID	Vikar får visning af egen info/ UC015
fremgangsmetode	1. Vikar navigerer til ”min side” 2. system finder specifik vikar baseret på hvilket ID brugeren(vikaren) er logget ind med. 3. system viser vikarens egne kontakt informationer.
Use case titel / ID	Vikar redigerer brugernavn + password / UC016
fremgangsmetode	1. Vikar navigerer til ”min side” og trykker rediger brugernavn + password. 2. systemet viser formular. 3. Vikar laver ønskede ændringer og trykker ok. 4. systemet spørger om bekræftelse. 5. Vikar bekræfter. 6. System gemmer opdatering i vikar database. 7. Slut.

Klasse-diagram

Til vores frontend/hjemmeside anvendes flere forskellige klasser. Disse klasser er imidlertid ikke bygget på en måde, hvor de nedarver fra hinanden, men i stedet bruger forskellige klasser. Dette kan ses i vores klasse-diagram, hvor mange af de forskellige Renderer-klasser anvendes ”use” i vores ListRenderer-klasse. Klasse-diagrammet kan ses på næste side:



ER-diagram

I denne database har vi 3 tabeller. ”Substitutes”, der indeholder information om hver vikar i databasen med hver deres unikke ID. En tabel ”Shifts”, der indeholder praktiske informationer om alle vagter såsom dato, starttidspunkt, sluttidspunkt, medarbejderen, der har vagten og et unikt ID til hver vagt. ”Medarbejder” i tabellen er en foreign key fra ”Substitutes”- tabellen, hvor man dermed knytter en unik medarbejder fra Substitutes-tabellen til en unik vagt.
Derudover har ”Shifts”-tabellen en boolean værdi ”ShiftIsTaken”, der beskriver, hvorvidt en vagt er taget eller ej.

Den tredje tabel ”ShiftInterest”, er en såkaldt ”junction”-tabel, der indeholder par af ”ShiftID” og ”EmployeeID”. Denne tabel registrerer vikarer, der har udtrykt interesse for forskellige vagter.

Tabellerne og opsætning af data i databasen skal opfylde 3. normalform og vi vil nu beskrive, hvorfor vores tabeller gør lige det.

Substitutes Table

Alle kolonner afhænger af hele primærnøglen (EmployeeID).

Kolonnerne er direkte afhængige af EmployeeID, så dette opfylder 1NF og 2NF.

Shifts Table

Alle kolonner afhænger af hele primærnøglen (ShiftID).

EmployeeID er en fremmednøgle og afhænger kun af ShiftID, og ShiftID er en entydig identifikator for rækkerne i denne tabel. Dette opfylder 1NF og 2NF.

ShiftInterest Table

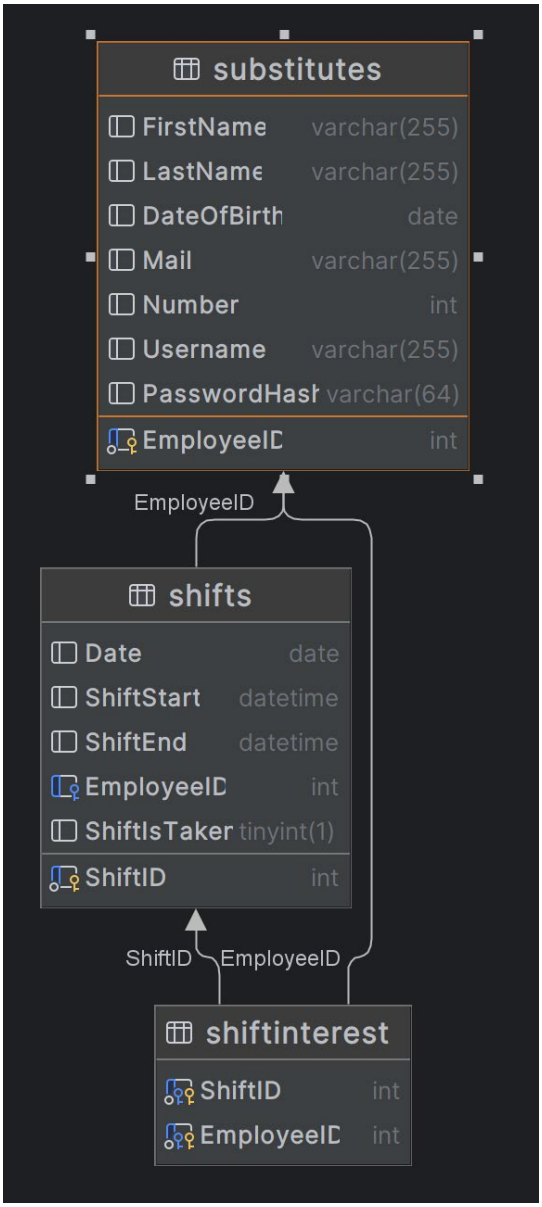
Alle kolonner afhænger af hele primærnøglen (ShiftID, EmployeeID).

Både ShiftID og EmployeeID er nødvendige for at identificere rækkerne i denne tabel, og begge kolonner er simple (ikke-sammensatte) nøgler. Dette opfylder 1NF og 2NF.

Derfor opfylder de tre tabeller 1NF og 2NF. For at bekræfte, om de opfylder 3NF, skal vi se på transitive afhængigheder, hvor ingen ikke-nøgleattribut af en tabel er transitivt afhængig af nogen supernøgle. I vores tabeller er der ikke nogen transitive afhængigheder, og alle attributter afhænger direkte af hele primærnøglen.

Derfor overholder de tre tabeller 3. normalform.

De 3 tabeller og deres relationer til hinaden kan ses illustreret til højre.



Analyser

Risikoanalyse

Vi har udarbejdet denne risikoanalyse for at identificere og lave en handlingsplan for risici forbundet med udviklingen af den løsning vi udvikler for ”LKbh”. Vi har primært haft fokus på de risici, der vedrører vores arbejde og udarbejdelse af løsningen.

Risk-management strategi

Vi benytter os af en proaktiv risikostrategi, hvor risici bliver identificeret inden projektets start og der bliver udarbejdet en aktionsplan for hver enkel risiko, således, at alle trusler for vores projekt er kalkuleret for og medregnet i vores projektplanlægning.

Identificering af risici, sandsynlighedsvurdering af risici og risikofaktorernes konsekvens

Vi vil nu vurdere, hvor stor sandsynligheden er for, at de forskellige risici indtræffer. Det er selvfølgelig relevant at have denne sandsynlighedsovervejelse in mente, når man som team skal vurdere, hvilke risikofaktorer man skal have fokus på.
Vi vil benytte os af denne model til at vurdere sandsynligheden af en risikofaktors skulle indtræffe i procent:

- Meget lav (<10%) værdien 1
- Lav (10%-25%) værdien 2
- Moderat (25%-50% værdien 3
- Høj (50%-75%) værdien 4
- Meget høj (>75%) værdien 5

Det er også vigtigt at vurdere, hvor stor en betydning eller konsekvens de forskellige risikofaktorer har for projektet. Hvis det er meget sandsynligt, at noget indtræffer, men at det ingen betydning har for arbejdsprocessen, det færdige produkt eller andre elementer af projektet, så er det ikke relevant at tage højde for.
Vi har taget udgangspunkt i denne model til at vurdere de forskellige risikofaktorers konsekvens:

- Ubetydelig værdien 1
- Tålelig værdien 3
- Alvorlig værdien 7
- Katastrofal værdien 10

Her er en liste over de risici vi mener er relevante for vores projekt med sandsynlighedsvurdering og konsekvens for projektet:

1. Et medlem vælger at forlade gruppen.
   - Dette er en personalerisiko, som anses som en risiko, da med en mand mindre i et team på 4 personer, så skal de tilbageværende arbejde væsentligt mere effektivt, for at nå i mål med projektet.
   - Vi har givet denne risikofaktor en sandsynlighedsværdi på 2.
   - Vi har givet denne risikofaktor en konsekvensværdi på 7.
2. Tidsplanen skrider.
   - Dette er en estimeringsrisiko, da vores estimering af opgaver/ userstories omfang ikke altid kan forventes at være præcise og i sidste ende kan resultere i, at vi ikke bliver færdig med vores projekt eller med alle de funktionaliteter som vi havde sat os for og aftalt med vores kunde.
   - Vi har givet denne risikofaktor en sandsynlighedsværdi på 4.
   - Vi har givet denne risikofaktor en konsekvensværdi på 7.
3. Krav fra kunde kan ikke imødekommes.
   - Dette er en produktrisiko, hvor det produkt som kunden forventer, at vi som team leverer, ikke kan leveres. Det er derfor en risiko ift. det færdige produkt og dermed en produktrisiko.
     Hvis ikke vi kan levere produktet, om det så grunder i manglende tid eller kodeevner, så må det kunne konkluderes, at vi ikke har fuldført opgaven.

- Vi har givet denne risikofaktor en sandsynlighedsværdi på 3.
- Vi har derfor givet denne risikofaktor en konsekvensværdi på 3.

4. Fejlestimering af opgaver
   - Dette er en estimeringsrisiko, hvor vores manglende erfaring som programmører også medfører en manglende evne til at estimere brugen af tid til diverse opgaver.
   - Vi har givet denne risikofaktor en sandsynlighedsværdi på 4.
   - Vi har givet denne risikofaktor en konsekvensværdi på 7.

5. Kunde ændrer i krav eller kommer med nye krav undervejs i projektet.
   - Dette er en produktrisiko, hvor produktet som kunden ønsker ændres. En ændring af produktet kan medføre, at vi som team udfører en masse spildt arbejde og bruger tid og ressourcer på noget som ikke skal bruges. I tilfældet med nye krav, der vil omfanget af produktet stige som vi ikke tidsmæssigt har estimeret for.
   - Denne risikofaktor tildeler vi en sandsynlighedsværdi på 1.
   - Vi har givet denne risikofaktor en konsekvensværdi på 7.

6. Sygdom i gruppe.
   - Dette er en personalerisiko. Foruden risiko for sygdom iblandt os i gruppen, så er vi to i gruppen, der har små børn. Vi er derfor ekstra udsat for, at sygdom i den nære familie vil kunne have indflydelse på, hvor meget arbejdskraft vi har til rådighed i projektperioden.
   - Denne risikofaktor tildeler vi en sandsynlighedsværdi på 6.
   - Vi har givet denne risikofaktor en konsekvensværdi på 3.

Risikoovervågning

Det er vigtigt, at alle risikofaktorer bliver overvåget løbende i den forstand, at vi som team i vores proces skal vurdere om risici har ændret sig. Er der kommet nye til? Er der eksisterende risikofaktorer, som ikke er aktuelle længere og kan fjernes fra vores risikoanalyse?
Derfor bør alle risikofaktorer ved afslutningen af hvert sprint genovervejes ift. deres sandsynlighed, aktionsplan og hvorvidt det er en reel risiko længere.

Vi sætter nu værdierne for konsekvens og sandsynlighed for de respektive risikofaktorer ind i vores udvidede risikotabel. Hvis produktet er over **10**, så er det en risikofaktor, der er værd at holde øje med og have en aktionsplan for.

Udvidet risikotabel

Hele den foregående risikoanalyse bygger op til denne udvidede risikotabel, hvor alt, hvad vi indtil nu har fundet ud af, kan plottes ind i tabellen.

Udvidet risikotabel						
Risikoindex	Sand-synlighed (1-5)	Konse-kvens (1-10)	Produkt	Præventive tiltag	Løsningsforslag	Ansvar

Gruppemedlem forlader gruppe	2	7	14	Sørge for, at alle teammedlemmer trives, føler at de bliver hørt og samtidig også kan omgås socialt.	Omstrukturering af arbejdet. Mere intensive sprints, hvor den tidsmæssige buffer er fyldt ud.	Marcus
Tidsplan skrider	4	7	28	Korte sprints. Daglige SCRUM-møder om vores udvikling. Burndown chart.	Fylde den tidsmæssige buffer ud. Evt, omstrukturere arbejdet.	Jon
Krav fra kunde kan ikke imødekommes	3	3	9			
Fejlestimering af opgaver	4	7	28	Korte sprints. Daglige SCRUM-møder om vores udvikling.	Fylde den tidsmæssige buffer ud. Evt, omstrukturere arbejdet.	Palle
Ændrede/nye krav fra PO	1	7	7			
Sygdom i gruppe	3	6	18	Alle kan lave alt. Holde hinanden opdateret om, hvordan opgaver kan løses.	Omstrukture arbejdet. Andre i gruppen overtager opgaver.	Magnus

Feasibility study

Mening med denne feasibility report er at vi vil gennemse de forskellige områder af projektet. Vi vil undersøge hvilke svagheder og udfordringer der kunne være ved et muligt projekt, men også hvilke styrker der ville kunne være.

Dette projekt har som formål at lave en hjemmeside med et skema, hvor en leder vil kunne oprette forskellige ledige vagter. Det er så muligt for forskellige medarbejder at mælde sig på de ledige tider. Inden projektet, så blev planlægningen gjort gennem sms beskeder, som hver medarbejder så kunne svare tilbage på, om de var ledige til at kunne tage en vagt. Mening med projektet er så at frigøre en stor mængde planlægnings tid, der så ville kunne bruges andetsteds.

Teknisk gennemførlighed

I denne del af vores feasibility study analyseres de tekniske udfordringer i vores projekt.

Vi arbejder med programmeringssprog og -koncepter, som vi er bekendte med. Vi benytter Javascript, som vi kender forholdsvis godt og har arbejdet en del med, så er vi på forholdsvis sikker grund i forhold til at kunne komme i mål med vores projekt. Risiciene for ikke at kunne lave projektet vil være større ved implementering af programmeringssprog og koncepter, som vi ikke er bekendt med eller har så meget erfaring med. Der kan være delelementer i projektet, som vi ikke har prøvet at lave før.

Det projekt, som vi skal lave, er ikke nyskabende i forhold til udvikling af ny teknologi/programmeringskoncepter. Der vil derfor være en del hjælp at hente til at lave diverse delelementer. Vi er fire personer i projektgruppen, så der er forholdsvis meget arbejdskraft til rådighed i projektperioden.

Vores webapplikation kræver ikke, at brugere har store it-kundskaber eller særligt hardware. Appen vil fungere som mange andre standardhjemmesider. Der kræves blot internetforbindelse.

Det kan være værd at notere sig, at et feasibility study og en SCRUM-proces ikke nødvendigvis går hånd i hånd. Hvis man skal vurdere, hvorvidt et projekt kan lade sig gøre, kræver det også grundlæggende viden om projektet. I en SCRUM-proces er de endelige krav ikke kendt på forhånd, og det er derfor svært at vurdere, hvorvidt et sådant projekt kan lade sig gøre/er hensigtsmæssigt at føre ud i livet.

Økonomisk gennemførlighed

Når det kommer til den økonomiske del af projektet, er der mange forskellige ting, der skal tages hensyn til. Før projektejeren kan få sit produkt, skal han først betale for udviklingen. Dette kan være i form af løn til udviklere og eventuelle materialer, som de kan få brug for under udviklingen. Efterfølgende vil der også være en pris for at holde produktet vedligeholdt/videreudviklet, hvor projektejeren også skal betale for eventuelle serveromkostninger samt løn til dem, der skal køre produktet.

I dette projekt er det os, der har kontaktet produkteejeren for at kunne lave vores projekt. Det er derfor os som udviklere, der gerne vil skabe et produkt, der kan være til gavn for produkteejeren, og vi vil derfor udføre dette projekt ulønnet.

Efter at projektet er færdiggjort, vil der kun være få ting, der kræver vedligeholdelse. Produkteejeren vil være nødt til at købe domænet, hvor hjemmesiden kommer til at køre. Det vil også være nødvendigt at have en person til at holde styr på databasen, hvor forskellige medarbejders personoplysninger er placeret, samt hvor tidligere vagter er gemt. Dette kan dog løses med en eller to personer.

Cost	Benefit
Målbar • Server • Vedligeholdelse • Løn ***Ikke målbar*** • Omstillings periode	***Målbar*** • Ingen udviklingsomkostninger • Hurtigere planlægningstid ***Ikke målbar*** • Bedre oversigt over tider

Lovmæssig gennemførlighed

I dette projekt vil der blive arbejder og behandlet forskellige personoplysninger. Men eftersom disse oplysninger allerede er blevet givet til arbejders, giver i forvejen, så vil der ikke være nogen brud på GDPR. I tilfælde af at en medarbejder skulle fratræde, så har virksomhed/arbejdes giver pligt til hurtigst muligt at få fjernet fratrådte medarbejders oplysninger.

Operationel gennemførlighed

Det operationelle aspekt af et feasibility study omhandler bl.a., hvorvidt det foreslåede projekt løser virksomhedens problemer og understøtter virksomhedens forretningsstrategi.

I øjeblikket ringes der rundt til vikarer for at dækket ledige vagter. Med vores løsning vil dette ikke længere være nødvendigt i samme omfang. Dette projekt skulle derfor gerne lette arbejdsbyrden for planlæggerne på bostedet og gøre det mere overskueligt og gennemskueligt for vikarer at booke nye vagter og se tidligere vagter. Virksomheden har en hjemmeside i forvejen (https://lkbh.dk/), så det kunne være oplagt at gøre bookingsystemet for vikarer til en del af deres egen hjemmeside.

Planlagt gennemførlighed

Deadline er sat til D. 15. december 2023. Denne deadline er ikke til forhandling da det er et eksamensprojekt. Det giver os i alt 4 arbejdsuger til at færdiggøre det endelige produkt. Vores gruppe består af 4 udviklere ud af 4 maksimalt tilladte, altså er vores arbejdskraft på

fuld styrke. Vi bruger scrum som arbejdsproces og planlægger vores opgaver derefter med et tilhørende scrumboard og burndown chart. Disse værktøjer giver os et løbende indblik i vores tidshorisont og hjælper os med at forudsige eventuelle forsinkelser i god tid. Gennem vores uddannelse har vi allerede stiftet bekendtskab med lignende opgaver og er derfor nogenlunde beredt på omfanget af denne eksamens opgave. Ydermere har vi mulighed for at få vejledning gennem vores undervisere hvis der skulle opstå komplikationer undervejs.

Den første uge bruger vi på et ”sprint 0” hvor vi vil udvikle vores indledende analyser, diagrammer og datastruktur. I sprint 1 vil vi fokusere på at få backend og frontend til at fungere. Der skal gerne kunne fetche data og udføres CRUD operationer via lokal database i datagrip og postman. Sprint 2 vil omhandle implementering af bruger-funktioner som sortering og filtrering samt opsætning af database på Microsoft Azure. Det sidste sprint 3 er tænkt som buffer til forventet opsamling samt finpudsning. Eventuelle risici for ekstraordinære komplikationer vil fremgå af vores risikoanalyse.

Politisk gennemførlighed

I dette projekt vil der ikke være politiske udfordringer, som der skal tages udgangspunkt i eller hensyn til

Teknologier

Vi har valgt at bruge to forskellige teknologier til dette projekt. Den første teknologi bruger vi til at hoste både vores MySQL-database og Node.js backend-app. Den anden teknologi bruger vi i vores frontend til at hoste og vise vores hjemmeside.

Azure

Teknologien til den generelle backend-del består af brugen af Microsofts hosting-tjeneste Azure. Nogle fordele ved at bruge Azure er den nemme integration med andre Microsoft-tjenester. En af disse er f.eks. Visual Studio Code, hvor alle medlemmer af vores gruppe har udviklet vores projekt. En anden fordel er skalerbarhed og global tilgængelighed, hvor Azure tilbyder nem justering af ressourceforbrug efter behov samt adgang til forskellige datacentre over hele verden. Dette gør det nemt at specificere forskellige regioner for at minimere latency for brugerne.

Nogle ulemper ved at bruge Azure er, at det kan være relativt dyrt sammenlignet med andre udbydere. En anden ulempe er kompleksiteten, som kan opstå på Azures platform. Dette skyldes de mange forskellige løsninger, som Azure tilbyder, hvilket hurtigt kan gøre det uoverskueligt for en ny bruger at danne sig et overblik.

Github

Til frontend (hjemmesiden) har vi valgt at bruge den teknologi, som GitHub tilbyder til projekter, der omhandler hjemmesider, nemlig GitHub Pages. Nogle fordele ved at bruge GitHub Pages er den gratis hosting, de tilbyder. Dette er specielt værdifuldt for mindre projekter, hvor omkostninger ved hosting andre steder kan blive et problem. En anden fordel er enkeltheden og nem implementeringen af selve GitHub Page-hjemmesiden, hvilket er særligt praktisk, når enkel hosting er tilstrækkelig til projektet.

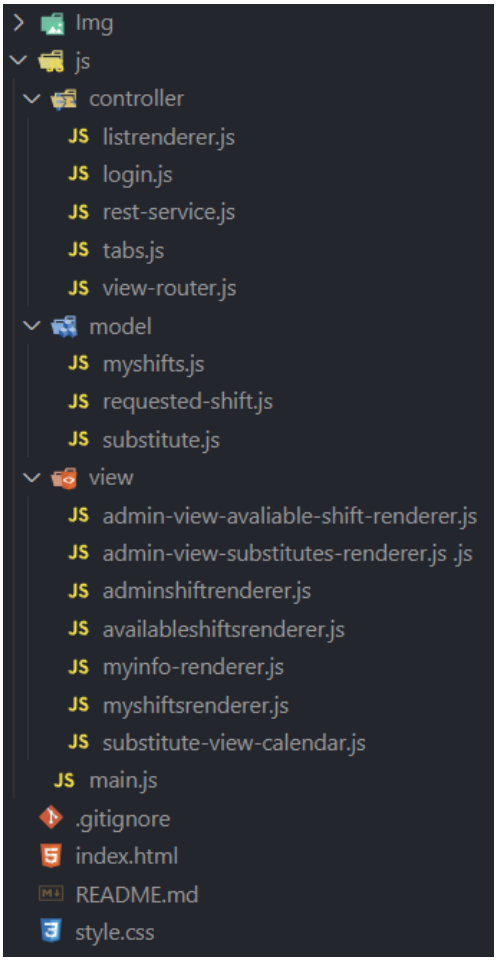
Nogle ulemper ved at bruge GitHub Pages alene er den begrænsede funktionalitet og manglen på databasestøtte. Med dette menes, at GitHub Pages bedst egner sig til statiske websider og derfor kun har begrænset funktionalitet sammenlignet med mere avancerede hostingtjenester. Dette er også grunden til, at vi i projektet har valgt Azure til både backend og MySQL-databasen.

Beskrivelse af kode

I dette projekt har vi valgt at opdele backend og frontend i to separate repositories, hvilket betyder, at både Node.js og MySQL-databasen håndteres i backend, mens hjemmesiden håndteres i frontend.

Hjemmeside (Frontend)

Strukturen for vores frontend kan ses på billedet til højre. Den generelle struktur består af en index.html, style.css, en 'img'-mappe, en README.md og en 'js' (JavaScript)-mappe. Alt dette er placeret i grundmappen, også kaldet 'root' af hjemmesiden.



Inden i JavaScript-mappen anvender vi konceptet MVC til at organisere filerne. MVC står for Model, View & Controller. I 'controller'-mappen har vi en fil kaldet 'main.js', hvorfra hele hjemmesiden startes op. Ud over denne fil har vi også andre filer og funktioner. Dette skyldes, at filerne i 'controller'-mappen er dem, der kan manipulere andre filer i programmet. Med dette menes, at det er de filer, der modtager inputs og kommandoer, som derefter bruges i model og view til at ændre, hvad der vises på hjemmesiden. For eksempel har vi en funktion i controller, der kaldes 'view-router'. Det, denne funktion gør, er at tjekke, hvilken del af hjemmesiden brugeren befinder sig på. Hvis brugeren klikker på et af de andre side-links, opfanger 'view-router' dette og fortæller derefter den relevante 'view-Renderer'-funktion, at den skal ændre, hvad der vises på hjemmesiden.
Vi benytter også flere forskellige klasser til at opbygge hvordan og hvad hjemmesiden skal kunne. Alle

klasserne kan ses på det tidligere billed for *klasse-diagram* delen. Som nævnt tidligere så har vi ikke en direkte nedarvning mellem vores klasser, men i stedet så har vi det således at en klasse bruger de andre klasser til hvad der skal vises på hjemmesiden. Det er denne klasse der sender hvad der skal vises fra model til view klasserne.

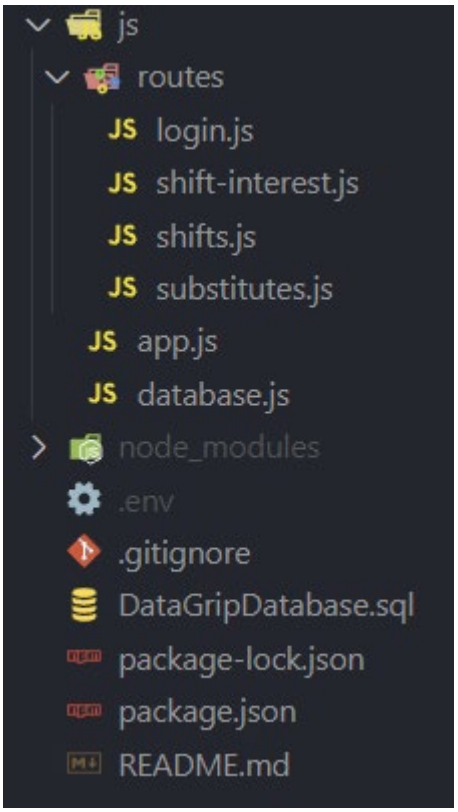
For eksempel, hvis en administrator er logget ind på hjemmesiden og gerne vil se, hvilke vagter nogle vikarer har budt på, samt hvilke der stadig står tomme, kan vedkommende klikke på 'Vagter'-linket i navigationsbjælken. Dette vil udløse kaldet til ListRenderer-funktionen sammen med andre nødvendige klasser for at oprette siden. I dette tilfælde bliver ListRenderer kaldt med følgende parametre:

```
// ADMIN: shifts, new instance of Listrenderer for shifts (admin view)
const shiftsAdminList = new ListRenderer(shifts, "#shifts-admin-tbody", adminShiftRenderer);
shiftsAdminList.render();
```

Klassen oprettes med listen over vagter, et ID til, hvor i HTML-koden det skal placeres, og til sidst specificeres, hvordan indholdet skal vises, i dette tilfælde som en tabel. Efter oprettelsen køres klassens renderer-metode for at generere selve koden og vise den på hjemmesiden. ListRenderer-klassen kan ses på billedet nedenfor.

```
export class ListRenderer {
    constructor(list, container, itemRenderer, buttonID1, buttonID2) {
        this.items = list;
        this.container = document.querySelector(container);
        this.itemRenderer = itemRenderer;
        this.buttonID1 = buttonID1;
        this.buttonID2 = buttonID2;
        this.filteredItems = []; // Initialize filteredItems array
    }

    render() {
        this.container.innerHTML = "";
        const filteredList = this.items.filter(
            (item) => this.filterValue === "all" || item[this.filterProperty] == this.filterValue
        );
        for (const item of filteredList) {
            const html = this.itemRenderer.render(item);
            this.container.insertAdjacentHTML("beforeend", html);

            if (this.itemRenderer.postRenderer) {
                const element = this.container.lastElementChild;
                const button1 = element.querySelector(this.buttonID1);
                const button2 = element.querySelector(this.buttonID2);
                this.itemRenderer.postRenderer(item, button1, button2);
            }
        }
    }

    sort(sortBy, sortDir) {···
    }

    filter(filterProperty, filterValue) {···
    }

    applyFilter(item) {···
    }

    extractSortValue(item, sortBy) {···
    }

    convertToMinutes(timeString) {···
    }
}
```

NodeJS (Backend)



Den generelle struktur for backend kan ses på billedet til højre. Billedet viser, at backenden grundlæggende består af en mappe til alt JavaScript samt en undermappe til de routere, som frontend ønsker at forbinde med databasen.

Derudover er der en "node_modules"-mappe, der indeholder de forskellige Node.js-biblioteker, som backenden bruger i programmet.

En .env-fil eller "dotenv" er en privat fil, som man opretter lokalt for at etablere forbindelse til sin egen server eller en anden server. Gitignore bruges til at specificere hvilke mapper/filer, der ikke skal inkluderes, når der udføres et "commit" på GitHub.

"DataGripDatabase.sql" bruges til at genstarte/nulstille databasen i tilfælde af, at der opstår fejl under "push". Den anvendes også til at skabe et fælles grundlag for, hvordan en lokal version af databasen skal konfigureres for, at programmet fungerer.

```
"name": "lkbh-semester-projekt-backend",
"version": "1.0.0",
"description": "Et booking system for LKHB",
"main": "./js/app.js",
"type": "module",
 Debug
"scripts": {
    "start": "node ./js/app.js",
    "dev": "nodemon ./js/app.js",
    "watch": "node --watch ./js/app.js"
},
"author": "JMMP",
"license": "ISC",
"dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "mysql2": "^3.6.5",
    "nodemon": "^3.0.1"
}
```

"Package-lock.json" og "package.json" er filerne, der specificerer hvilke "dependencies" eller pakker, der skal downloades, når man henter programmet.

På venstre side af billedet kan man se opbygningen af "package.json"-filen. Ud over de filer, der skal downloades som "node_modules" i "dependencies", angiver filen også grundlæggende informationer om programmet. Den specificerer, hvilken fil der skal startes fra "./js/app.js" og at det skal være af typen "module", så man kan eksportere og importere i forskellige .js-filer. Derudover angiver den også, hvordan programmet startes, ved hjælp af forskellige "scripts".