colorlinks=true, linkcolor=blue, filecolor=magenta, urlcolor=cyan, pdftitle=Overleaf Example, pdfpagemode=FullScreen,

DevOps afsluttende rapport

62582 - Komplekse systemer & DevOps - Fall 22

Website: https://probe.joecthomsen.dk/https://probe.joecthomsen.dk

Github: https://github.com/Joecthomsen/probe/https://github.com/Joecthomsen/probe

Johannes Claudius Thomsen (s190536)
Kasper Buur Vistesen (s195168)
Jonas Grelle (s205114)
Troels Engsted Kiib (s205492)
Annika Ehlers (s153667)

27. december 2023

Indhold

1 Indledning

PROBE er en startup virksomhed som vil lave en platform der kan hjælpe mulige deltagere i at finde kliniske studier som passer til dem. Behovet for en sådan platform kommer af at hospitaler ikke selv må reklamere med kliniske studier af etiske årsager. Gruppe 8 har ladet sig inspirerer af denne problemstilling og lavet en applikation, som skulle kunne matche sundhedsvæsnet med potentielle forsøgsdeltagere. Vi har taget PROBE's problemstilling, og udvalgt fem usecases med henblik på at de skal udgøre så funktionel en fullstack applikation som muligt. Vi har i processen valgt at arbejde uafhængigt af PROBE og har derfor ikke fået deres feedback på noget af vores arbejde. Vores applikation er udelukkende vores eget design og implementation.

Jonas (80%) / Johannes (20%)

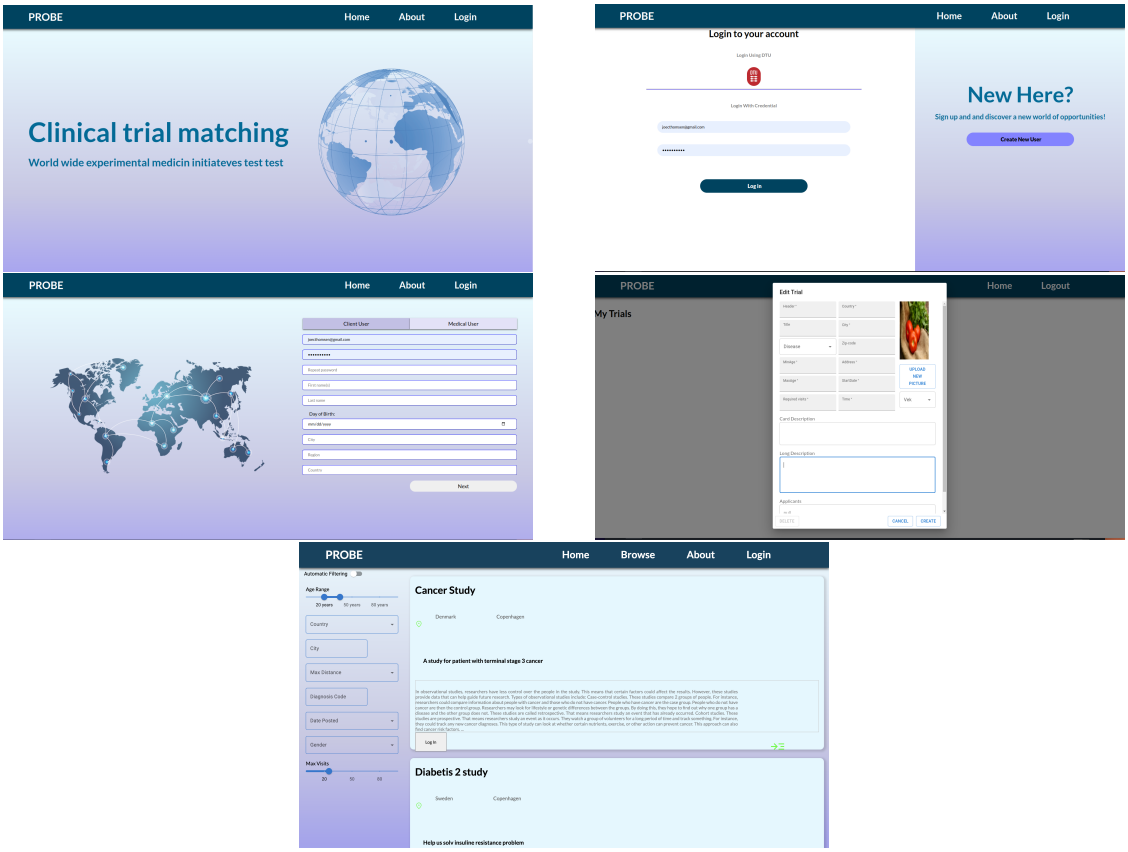
2 Overblik

Vores applikation er en single page application som består af en frontend der er skrevet i React, et rest API som backend der er implementeret via Springboot, og en MySQL database. Vores deployment pipeline foregår via Github actions som pusher kode fra vores master branch til en Docker container der er sat op med Caprover som platform der kører på en Azure server. Vi bruger gated commits og testing via Github actions som holder ikke-fungerende kode udenfor vores dev branch. Vi har monitorering via Prometheus og Grafana som også kører i Docker containere.

Jonas (100%)

3 Produktet

Produktet går i alt sin enkelthed ud på, at matche kliniske forsøg med potentielle forsøgsdeltagere. Lige nu er applikationen simpel. Det er muligt at oprette brugere, studier og gennemse relevante studier. Desuden er der en admin-side, hvor der er overblik over brugere. Produktet møder kravspecifikationerne som blev lavet den første uge.



Figur 1: Produktet

Johannes (80%) / Jonas (20%)

4 Design

4.1 Behovet til programmet

Da vores program skal fungere som en platform til deltagelse i kliniske studier, bliver vi først om fremmest nødt til at opdele programmet i en frontend og backend.

4.1.1 Frontenden

Når vi kigger på behovet for vores frontend, bliver vi nødt til at tænke i brugerens perspektiv. Brugeren i dette tilfælde vil primært enten være en udbyder af kliniske studier, eller en bruger der gerne vil deltage i kliniske studier. Den sidste bruger er relativt interessant i vores tilfælde, da denne bruger er den primære "bruger"af vores platform. Hvis vi prøver at analysere denne bruger lidt, må brugeren have nogle attributter, der giver brugeren mulighed for deltagelse i et klinisk studie, f.eks. kræft. Derfor kan brugeren allerede være stresset før de overhovedet tilgår vores frontend. Ud fra dette kan vi udtrække nogle behov til vores UX, som vi skal lægge den primære vægt på, såsom en hurtigt refresh tid imellem siderne samt en overskueligt UI. Dette bliver yderste relevant, da vores bruger, som skal deltage i kliniske studier, som sagt allerede kan være presset af sin helbredssituation og derfor have nemmere ved at droppe forsøget i at finde et klinisk studie og deltage i dette.

4.1.2 Backenden

Da platformen skal være tilgængelig på internettet, må der implementeres en hvis form for sikkerhed i programmet, således at brugernes følsomme information ikke bare ligger ude i det fri. Denne sikkerhed skal indholde SSL/TLS, således f.eks. at brugerens information ikke bliver sniffet. Løsningen burde også inkorporere en form for access control, der ikke lige kan udmanøvreres uden videre. Udover dette skal programmet også kunne håndtere mange brugere samtidig og derfor må der oprettes en måde at overvåge

platformens meta-data, således at man ville kunne opfange et evt. nedbrud i opstarten og have en ide om hvorfor nedbruddet skete. Denne overvågning af meta-dataen vil også give mulighed for at finde et evt. bruteforce attempt og på denne måde skabe endnu mere sikkerhed, hvis overvågningen bliver brugt.

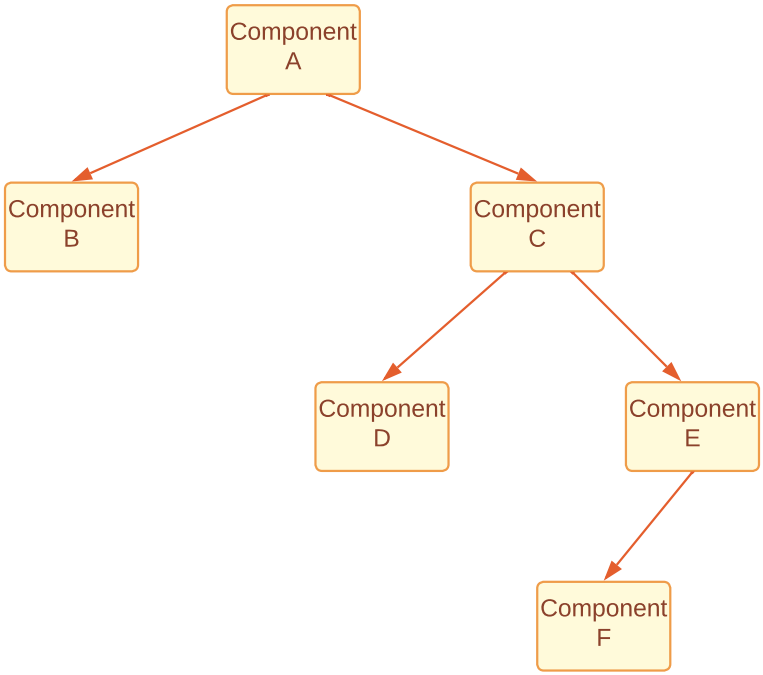
Troels (100%)

5 Implementering

5.1 Frontend

5.1.1 DOM

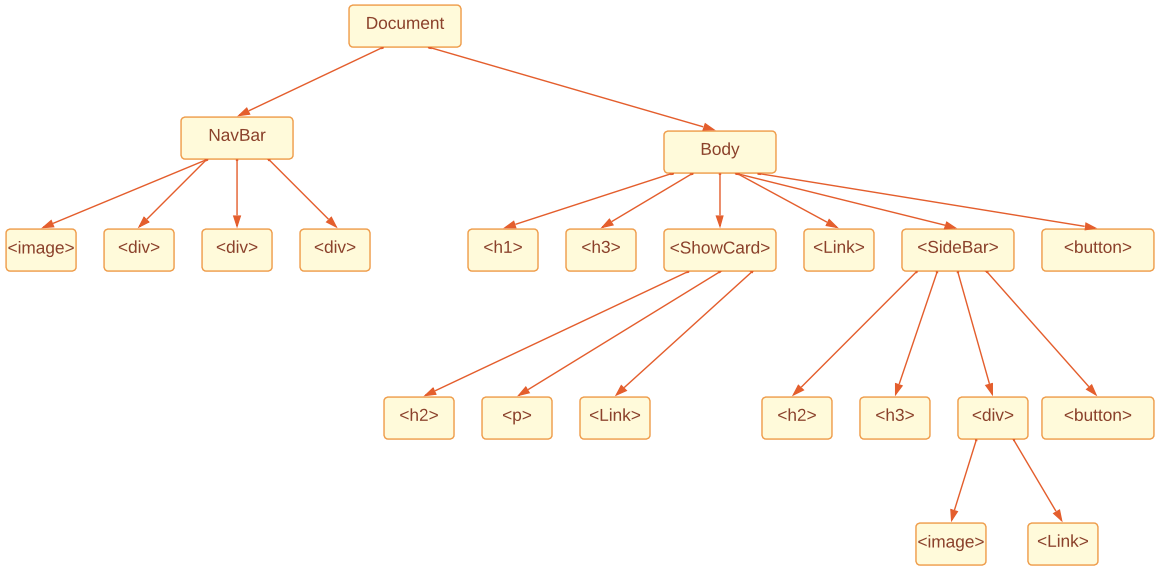
Centralt i en webapplikation bygger man i browseren det der hedder en Document Object Model (DOM). DOM'en repræsenterer alle de komponenter en webapplikation består af, og beskriver hvilken komponenter, der er børn af hvilken forældre. Nedenstående figur viser den træstruktur som en DOM fil benytter sig af.



Figur 2: Træstruktur i DOM fil

På ovenstående figur, kan det ses, at komponent E indholder F men ikke D og komponent A indholder naturligvis B, C, D, E og F

For at få et mere retvisende kontekst, er nedenstående figur tilføjet.



Figur 3: Træstruktur i DOM fil med HTML elementer

De samme principper fra figur 2 på side 5 er naturligvis stadigvæk gældende.

Johannes (100%)

5.1.2 React

Hele frontend applikationen er bygget med et bibliotek til Java Script kaldet React.
React virker på den måde, at det manipulerer DOM filen, så når elementer skal opdateres, så opdateres kun det komponent der faktisk er blevet ændret i og dette komponents børn. Hvis vi betragter figur 2 på side 5 og vi opdaterer komponent D, så er det kun D som reelt bliver opdateret og re-renderet. Hvis vi derimod opdaterer komponent C, bygger React også komponenterne D, E og F igen, såfremt der er ændringer. React holder hele tiden styr på state af komponenter, og opdaterer kun, hvis der sker en ændring.

Johannes (100%)

5.1.3 SPA

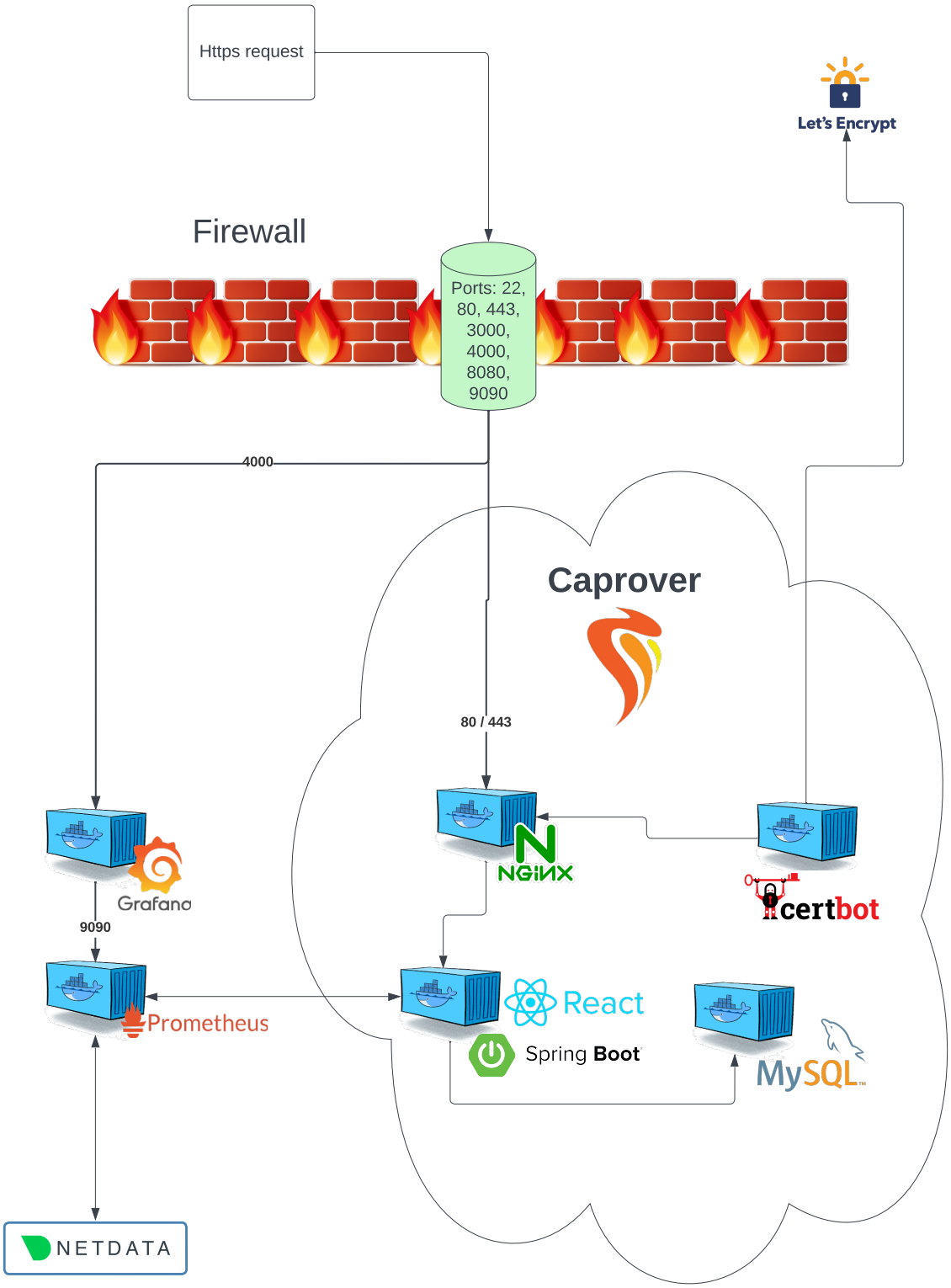
Single page applikation betyder, at man modtager hele applikationen, første gang man indleder en interaktion med den. Det betyder i praksis, at når man modtager det første view, som beskrevet i afsnittet om REST API, så modtager man faktisk de andre views som hele applikationen skal bruge. Det vil sige, at den første request man laver, der vil responset være noget tungere, fordi at den faktisk skal indholde hele skellettet til applikationen.
Fordelen ved dette er, at man som bruger får en mere native applikations følelse, altså følelsen af, at man sidder med en applikation, som er installeret på enheden. Dette skyldes, at når man skifter view, så skal man ikke længere fetche, fordi det blev jo gjort allerede da man fik den første respons. Der er selvfølgelig stadigvæk dynamisk indhold, som skal fetches, men det gøres asynkront og ødelægger derfor ikke følelsen af, at når man trykker på en knap, så responderer den med det samme.

Johannes (100%)

5.2 Backend

5.2.1 Overblik

Nedenstående figur illustrerer arkitekturen i backenden, som vil blive uddybet i de efterfølgende afsnit



Figur 4: Deployment diagram

Johannes (100%)

5.2.2 Caprover

Caprover er et opensource værktøj, som kan bruges til at pakke sin applikation ned i containere, med en proxy server i form af NGINX og en certbot container, som automatisk fornyer SSL/TLS certifikater. (Se figur 7 på side 11 for reference)
Caprover kan altså ses som en form for abstraktion, som i bund og grund automatiserer containerization af en applikation og håndterer certifikater.

Johannes (100%)

5.2.3 NGINX

Caprover håndterer deployeringen, så man faktisk ikke behøver at konfigurere NGINX selv. Det er dog vigtigt at forstå, hvad der forgår under motorhjelmen, både i forhold til at forstå ens server bedre, men også i forhold til, hvis man skal debugge en server.
I dette tilfælde bliver NGINX brugt som en reverse proxy server, altså en server der står foran resten af applikationen og vidrestiller indkommen trafik. Dette bliver rigtig smart, når man ønsker at at skalere et produkt. Her kan man blot referere flere instanser i nginx.config og på magisk vis, er de blevet en del af setup.
En fordel ved NGINX er også muligheden for, at simpelhen tage en server ud af drift, såfremt denne måtte præstere dårligt. En NGINX server i kombination med en solid overvågning af ens server, er altså et virkeligt stærkt værtøj, til at tilsikre at din server er sund.
En sidste ting, som burde nævnes når snakken går på NGINX. NGINX har mulighed for at fungere som load balancer. Det vil i praksis sige, at man har en række muligheder for at konfigurere, hvilken algoritme man ønsker at bruge, og herefter vil NGINX fordele belastningen ud på de servere som er til rådighed - selvfølgelig under hensynstagen til den konfigureret algoritme.

Johannes (100%)

5.2.4 Docker

Docker er en container teknologi, som muliggøre at pakke sine services ned i containere, som kører, uafhængigt af det underliggende operativ system og den underliggende hardware. I dette projekt har opsætningen af Docker været relativt simpel, fordi Caprover har håndteret det i forhold til selve applikationen.
Derfor, så er applikationen kørende i de docker containere (Se figur 4 på side 7) som Caprover har pakket den ned det. Ydermere er monitoreringen med Prometheus og Grafana også pakket ned i Docker containere, dog uden Caprovers indblanding.

Johannes (100%)

5.2.5 REST API

Muligheden for at interagere med backenden forgår gennem en REST API, som er bygget i Spring Boot. Spring Boot giver mulighed for, at relativt nemt og enkelt implementere en REST API som omverden kan kommunikere med.
Når brugeren navigerer rundt på siden, er der forskellig data, som denne ønsker at se. Hertil bruger vi et REST API, hvor vi sender requests fra frontenden til backenden. Dette kan både bruges til bare at vise data, men man kan også ændre, tilføje og slette data (CRUD). Dataen sendes i JSON format. Dette giver altså mulighed for dynamisk indhold på hjemmesiden.

Johannes (100%)

5.2.6 ORM

ORM, eller Object Relation Mapping, er en smart måde, hvorved man helt kan undgå at skrive SQL queries. I stedet for at skrive queries, så kan man mappe et objekt til en database. Det spare en del tid, specielt hvis man ønsker at tilføje eller fjerne kolonner i en tabel, så forgår processen smertefrit, simpelthen ved at blot redigere objektet.

Måden det i praksis udføres på, er ved at lave en "model"class, indholdende de elementer, som beskriver den tabel man ønsker at lave.

Johannes (100%)

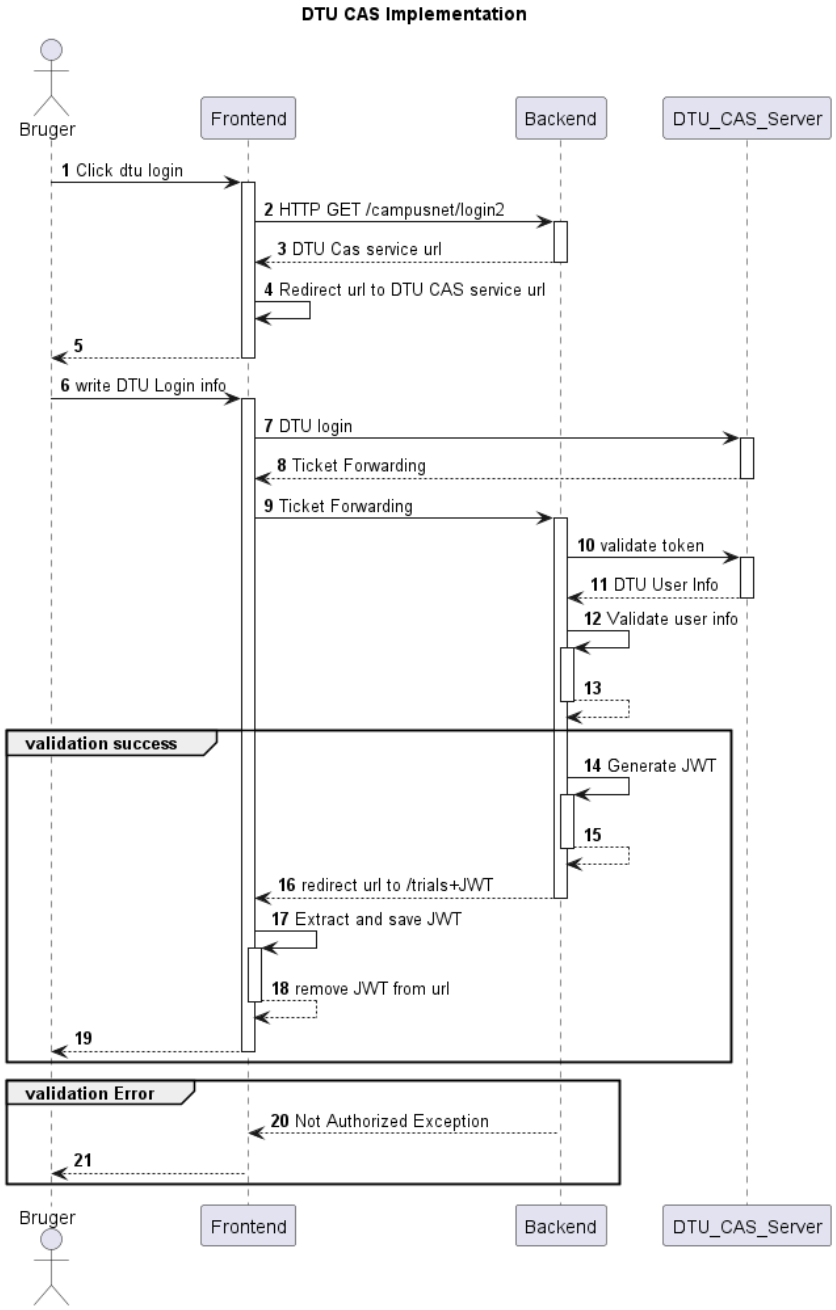
5.2.7 Certbot

Hvis man ønsker at folk skal føle sig trygge på ens hjemmeside, er man nødt til at anskaffe sig et SSL/TLS certifikat, så folk har mulighed for krypteret kommunikation med ens applikation. Dette gør man kan benytte https, hvilke for browseren til ikke at advare brugeren om at siden er usikker. I stedet for, at bruge en masse penge på at anskaffe sig et certifikat, så kan man få et gratis via letsencrypt.org.
Denne CA returnere et certifikat til brugeren sammen med en krypteret session key, og dermed kan web appen og brugerens kommunikation være krypteret.
Problemet opstår, når certifikater skal fornyes. Alle certifikater har en udløbdato, og når denne er overskredet, så kan man ikke længere etablerer sikker forbindelse. Dette er nemt at glemme, så derfor findes Certbot som automatisk kan forny certifikatet. Hvis Certbot imod forventning ikke kan gøre det automatisk, bliver man advaret før certifikater udløber - også automatisk.

Johannes (65%) / Jonas (35%)

5.2.8 Access Control

Vi har inkorporeret access control i vores system, således at når man opretter en bruger, bliver brugerens password hashet og saltet via. HMac og SHA512, og derefter bliver digest og salt gemt i databasen. Der bliver det password brugeren indtaster hashed med saltet, og sammenlignet med det digest der ligger i databasen. Brugeren har også mulighed for at logge ind med sit DTU-login da vi har implementeret DTU CAS, som følger CAS 2 protokollen. Efter et succesfuldt login, giver vi brugeren en JSON-Web-Token, der er computed med vores secret og giver os mulighed for at udføre et integritets check af brugerens information. Denne funktionalitet er dog uheldigvis kun implementeret under /edittrial/* og /user/authorize/* siderne. Der kan ses et diagram over login proccessen i vores system på figur 12 der kan også ses et sekvensdiagram over dtu cas login implementation på figur: 5



Figur 5: DTU cas implementation

Troels (65%) / Jonas (35%)

6 Pipeline

En vigtig del dette projekt er den pipeline som er blevet lavet undervejs.

Når man er færdig med sin kode, puller man den ind på vores dev branch, på dette pull bliver der kørt vores automatiske tests af vores kode. Derefter når featuren er klar til at blive deployet på det færdige miljø, opretter man et pull request på fra dev til master, for der bliver kørt automatiske regression tests, dette er et gated og kræver et review. Når testene består og reviewet bliver approved bliver koden deployet automatisk til et testmiljø vha. dockerhub, hvor der automatisk bliver kørt vores e2e test med cypres på. Til slut bliver koden automatisk deployet på vores produktionsmiljø.

Troels (50%) / Johannes (50%)

7 Operation

7.1 Idriftsættelse på et nyt system

Hvis systemet skal opsættet på et nyt system, skal man først og fremmest oprette en server, dette kunne f.eks. være en linux server. På denne server skal der nu installeres al den overordnede arkitektur via docker: Prometheus, Grafana, Caprover, Netdata og en mysql Server (Denne kan også oprettes igennem Caprover senere). Disse elemtenter skal nu containeriseres og gøres tilgængelig på overordenede porte som ses på figur: 4.

Dernæst skal der bare oprettes en app på caprover, og deploye vores docker image. I denne app der er blevet deployet på caprover, skal der sættes følgende miljø variabler på figur: 6. Man kan også opsætte MySQL server via. en oneclick action på caprover og deraf bruge caprover som en PaaS.

DB_HOST	probe-db-db.joecthomsen.dk
DB_NAME	probe_db
DB_PASSWORD	1234
DB_PORT	4567
DB_USERNAME	root
JWT_KEY	MegaHaemli
SERVER_PORT	8080

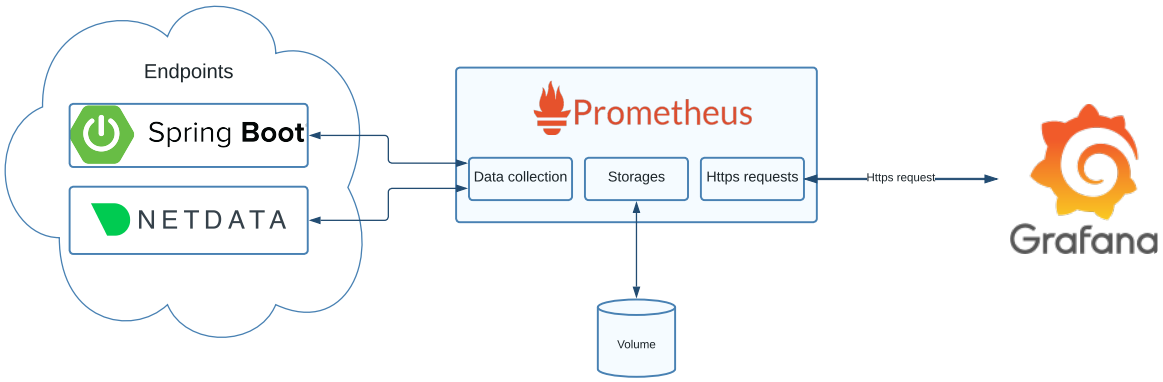
Figur 6: Miljø variabler der skal sættes

Troels (100%)

7.2 Monitorering

7.2.1 Overordnet

Prometheus og Grafana bliver brugt sammen med spring boot, således at spring boot udsteder et endpoint, som prometheus scraper (Samle data fra). Ydermere er netdata blevet installeret på serveren, hvis foremål er, at monitorer serverens generelle helbred. Ligesom med Spring Boot, udsteder netdata et endpoint, som prometheus kan opsamle data fra.
Grafana bliver brugt til at udstille data som prometheus har samlet sammen. Det betyder at prometheus ligeledes udsteder et endpoint, hvorfra Grafana kan opsamle data.
Nedenstående figur forsøger at klarlægge denne arkitektur



Figur 7: Monitorering overblik

Johannes (100%)

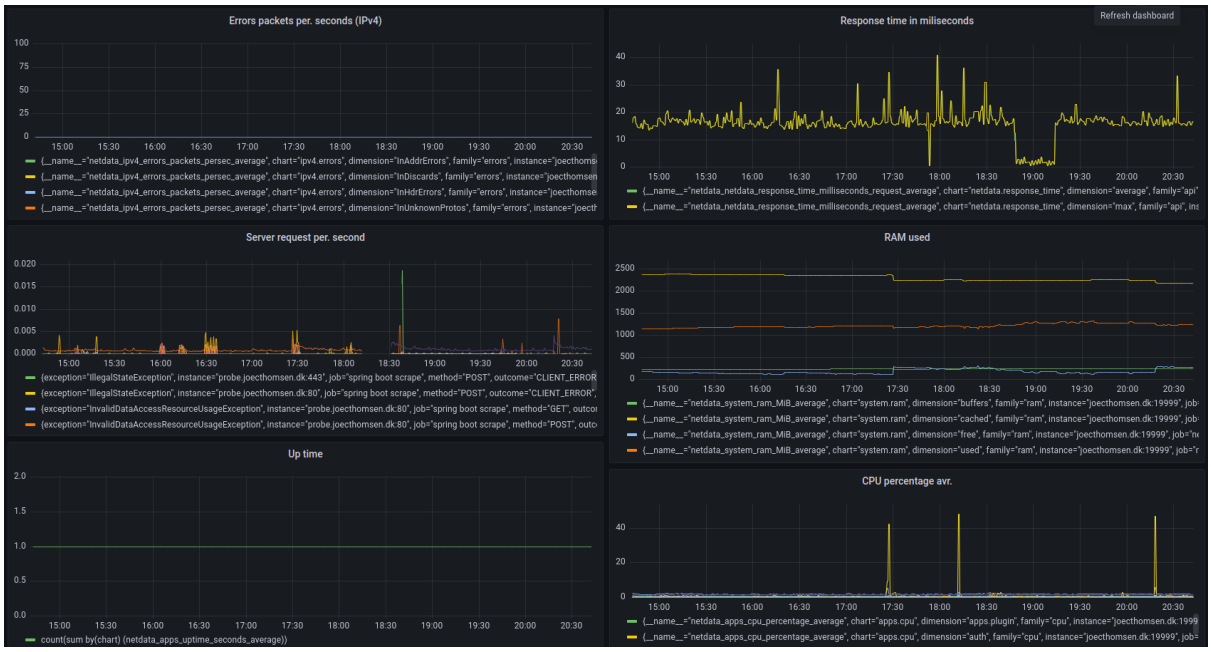
7.2.2 Formål

Formålet med overvågning er, at opdage uregelmæssigheder og fejl, før at de bliver kritiske. Selv hvis en fejl bliver kritisk, og man står over for et nedbrud, kan man, hvis man har lavet en god monitorering implementering, nemmere finde fejl, fordi man ikke længere leder i blinde, men tværtimod har et solidt datagrundlag, som kan indkredse fejlkilden.

Johannes (100%)

7.2.3 Servertilstands monitorering

Nedenstående figur viser, hvordan overvågning i Grafana ser ud. Som det ses, er der 6 metrics som bliver overvåget. Alle metrics er scrapet fra Netdatas endpoint (Se figur 7 på side 11)



Figur 8: Monitorering med Grafana

Dette er naturligvis blot et udpluk af, hvilke metrics man kunne overvåge på serveren. Netdata giver tusindvis af metrics i det øjeblik det bliver installeret på serveren og er derfor et godt værktøj til at overvåge det generelle helbred på en server.

Johannes (100%)

7.2.4 Eksempel af monitorering

Grundet generelt tidspres og en stejl indlæringskurve, lykkedes det ikke at udnytte de mange tusind metrics, men vi måtte i stedet udvælge de steder, hvor vi fandt det relevant at overvåge.
Et af de steder, der blev udvalgt til monitorering i dette projekt, var login delen. Her blev der lavet brugerdeffineret metrics, som i alt sin enkelthed består at en counter, som bliver inkrementeret i service laget. Dette bliver så tilføjet i spring boot endpoint (Se figur 7 på side 11) og kan derigennem overvåges via Grafana

Figur 9: Monitorering med Grafana

Johannes (100%)

7.2.5 Alarmering

En af fordelene ved at monitorere er, at man har mulighed for at oprette alarmer. Der er naturligvis også lavet alarmer i dette projekt. Bland andre bliver uptimeRobot brugt. UptimeRobot er en meget simpel alarm, som blot pinger serveren hver 5. minut og såfremt at serveren ikke svare, så modtager man en e-mail
En anden type alarmer, er Grafana alarmer. Disse er et stærkt værtøj, da man kan sætte dem, så man muligvis kan opdage at en servers tilstand er dårlig før den går ned. Det kunne bland andet være i de metrics som allerede er oprettet, f.eks hvis response tiden pludselig bliver meget lang, eller hvis der pludselig er mange uautoriseret login forsøg på serveren, så kunne dette tyde på, at et hacker angreb imod serveren er i gang.

7.3 Logging

Et hvert system, som er i i brug, burde have en log. En log, hvor alt vedrørende driften af ens system bliver skrevet ned og gemt. Ud over de auto genereret logs som Spring Boot genererer, så bør man implementerer en - til projektet - tilpasset log. Et af de værktøjer som man kan bruge, er log4j2, som giver en mulighed for at tilpasse loggen præcis som man måtte ønske dette.
log4j2 består af to abstrakte elementer, en logger og en appender, hvor loggeren er ansvarlig for at generere log meddelelser fra kilden og appenderen er ansvarlig for, at finde den korrekte log, og tilføje disse logs i filen. Dette kunne nok blive en stor fil efter et par måneders drift, derfor kan man implementere en "rolling appender"som kun gemmer loggen inden for et forud defineret vindue, f.eks 48 timer.
Data som man kunne finde interessant at gemme, kan grundlæggende deles op i data som omhandler selve servicen, såsom fejl, hvilken bruger der logger ind hvornår, transaktionshastighed og http statuskoder. Yderligere vil man typisk også logge data som omhandler "forretningslaget". Det vil sige, at man f.eks logger click-dept og conversion. Uanset hvad, så er logning en vigtig del af en sundt webapplikation og gør det nemmere at debugge og finde fejl i systemet. Grundlæggende er der nogle forskellige niveauer af logning, alt efter hvor kritisk en log besked måtte være.

- Trace
- Debug
- Info
- Warning
- Error
- Fatal

Med disse niveauer kan man nemmere finde de relevante log, da det ellers hurtigt bliver anstrengende at gennemgå tusindvis af linjer af godt og blandet logning.

Johannes (100%)

8 Usecases

8.1 Johannes' usecase

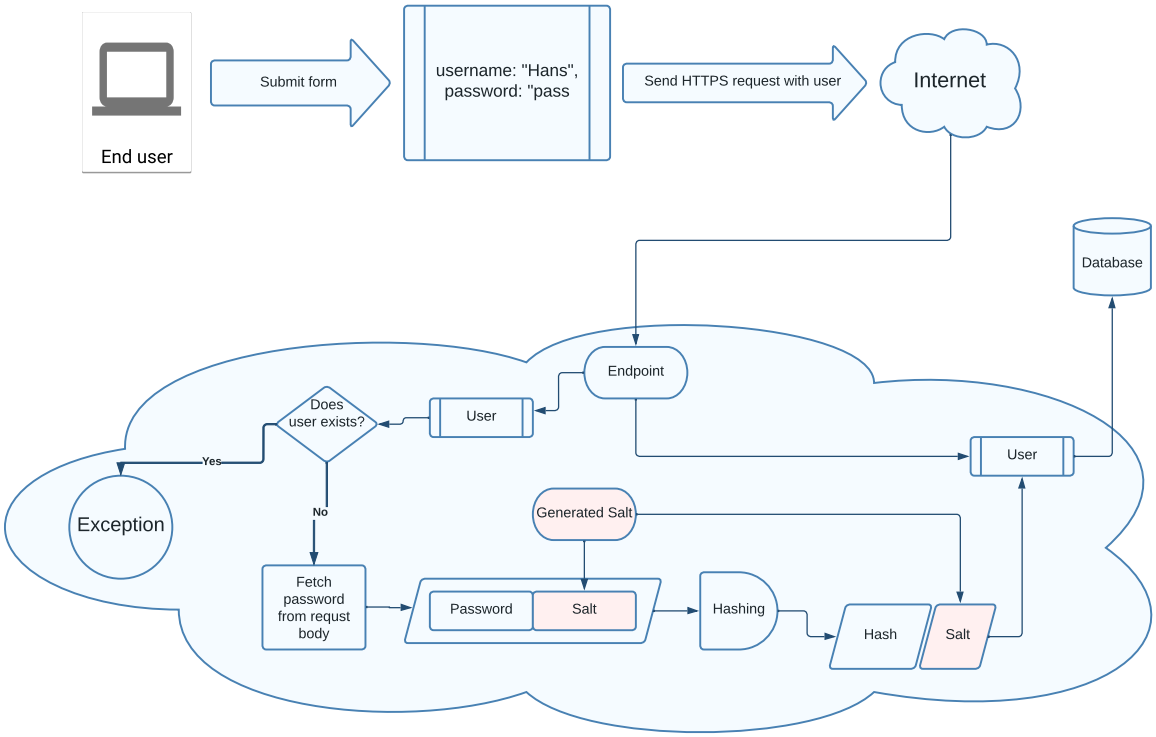
8.1.1 Landingpage

Landingpage er den første side en bruger lander på, når han indtaster domænet i sin browser. På landingpagen er der ikke umiddelbart nogen funktionalitet, men hvis brugeren scroller ned, vil han se en karrusel med udvalgte studier.

Johannes (100%)

8.1.2 Create user

Systemet kan håndtere oprettelse af brugere med forskellige roller. Man kan oprette brugere, som leder efter et studie, eller brugere som har et studie og søger deltagere til deres dette. Rent praktisk fungerer det på den måde, at frontend tager den data, som brugeren har indtastet i formen og sender som https request med brugeroplysningerne til et endpoint i backend. Når brugeren bliver modtaget i backenden, bliver det kontrolleret om der allerede er en tilsvarende bruger i databasen og kodeordet bliver hashet med et salt, som er en tilfældig streng man tilføjer, for at øge kompleksiteten i kodeordet og undgå at kodeordet kan udledes ved hjælp af rainbow table.



Figur 10: Simplificeret oprettelse af bruger og hashing af password

Johannes (100%)

8.1.3 Login with credentials

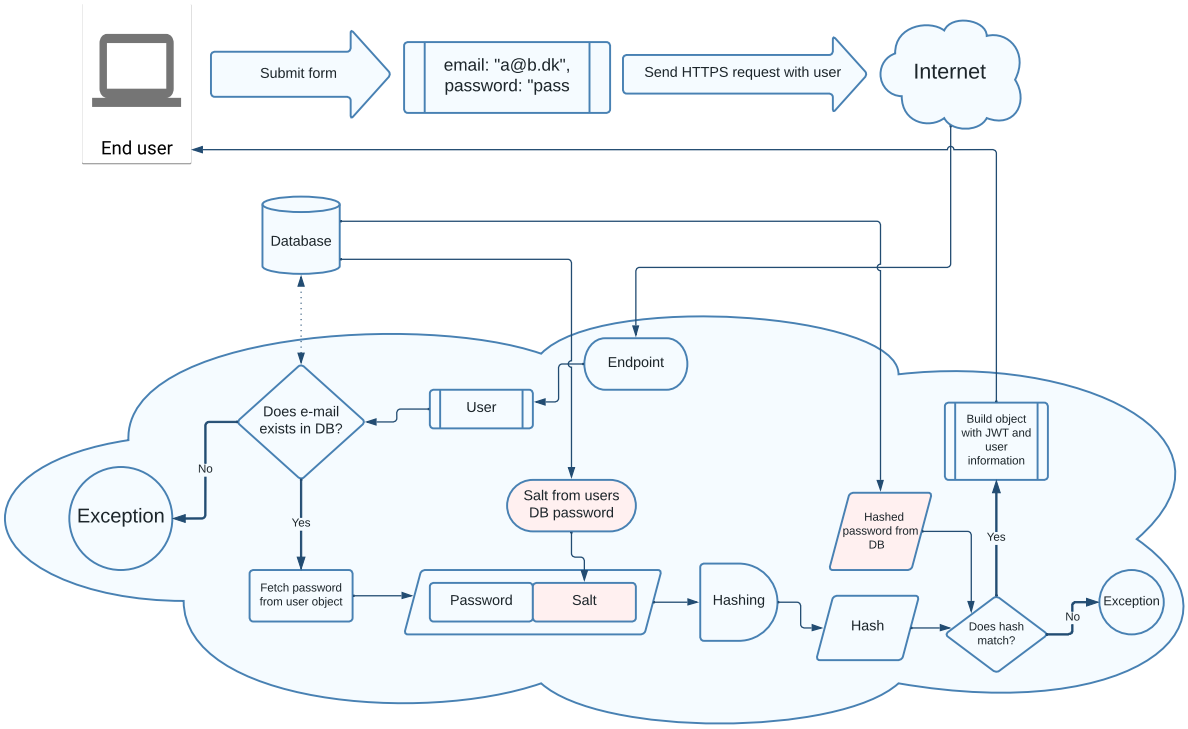
Når en bruger er blevet oprettet, kan man ligeledes logge ind på siden ved at navigerer til "Login"i navigations baren. Herfra indtastes blot brugerens e-mail og kodeord, hvorefter der trykkes på knappen

"Login".
Herefter bliver en https request send til backenden med credentials hvorfra disse bliver kontrolleret, og hvis alt går vel, bliver et objekt indholdene en signeret JWT token med e-mail, rolle og udløbstid. Resten af brugerens informationer, såsom adresse, alder, vægt og så videre, bliver blot tilføjet som key/value pairs i samme objekt, altså ikke signeret af JWT token.
Årsagen til, at det netop er disse værdier som bliver signeret i JWT token, er at disse variabler er kritiske. E-mail bliver brugt som id, da den er unik, derfor nytter det ikke noget, hvis en bruger har mulighed for, at ændre denne variabel i frontenden. Det samme gør sig gældende for rollen, her nytter det ikke, hvis man giver sig selv en administrator rolle, og herefter kan komme uberettiget ind i kritisk infrastruktur. Sidst er det naturligvis vigtig, at den udløbstid som token har fået, bliver signeret sammen med de andre variabler, da den i sagens natur ligeledes er kritisk. Lige nu, er en JWT tolen gyldig i 48 timer og man kunne argumentere for, at denne gyldighedsperiode var for lang. I stedet kunne der implementeres en kort gyldighedsperiode, men med mulighed for at fornye sin token løbende.


```
"token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.
    eyJyb2xlcyI6Ilt7XCJpZFwiOjEsXCJyb2xlTmFtZVwiOlwiQ0xJTklDQUxfVVNFUlwifV0iLCJpc3MiOiJQcm9iZURlbHV4ZSIsImlkIjoiXCJKb2VjdGhvbXNlbkB
    nbWFpbC5jb21cIiIsImV4cCI6MTY2OTcyNjY5NX0.
    uI162_zTnf4HGZRfWPbCsAqrL1wm8DkqiSL-C9ZSkkJRkfkFRD9lvRFCUXI_pcSJ0_mfmKlWWGeHm1XYNqmmtA",
"email": "joecthomsen@gmail.com",
"sex": "male",
"firstName": "Johannes",
"lastName": "Thomsen",
"dob": "1985-11-10",
"weight": "83",
"chronicDisease": "Covid-19",
"phoneNumber": "+45 22267892",
"streetName": null,
"doorNumber": null,
"zipCode": null,
"city": "København N",
"region": "Hovedstaden",
"country": "Denmark",
"roles": [
    {
        "id": 1,
        "roleName": "CLINICAL_USER"
    }
]
```


Figur 11: Ved succesfuld login, bliver dette objekt med JWT token til brugeren returneret

Nedenstående figur viser, hvordan login processen fungerer.

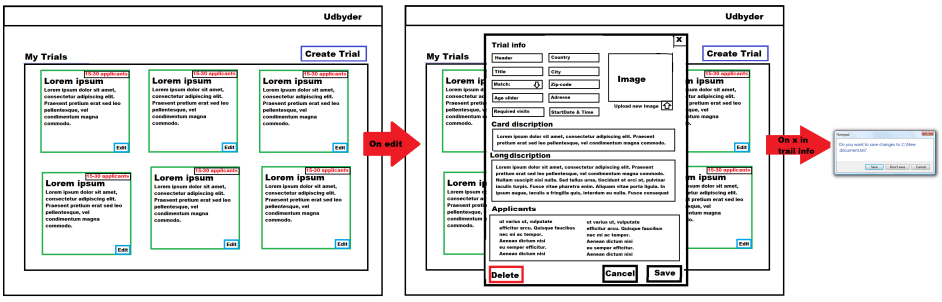


Figur 12: Flow chart over login processen

Johannes (100%)

8.2 Troels' usecase

Da de kliniske studier skal være oprette før en bruger kan tilmelde sig dem, bliver der nødt til at være en side dedikeret til udbyderen, hvor udbyderen har alle CRUD rettigheder på netop sine trials. Til denne side blev der først lavet en hurtig mockup af hvordan siden skulle se ud, som blev nogenlunde fuldt, mock-uppen ses på figur : 13.



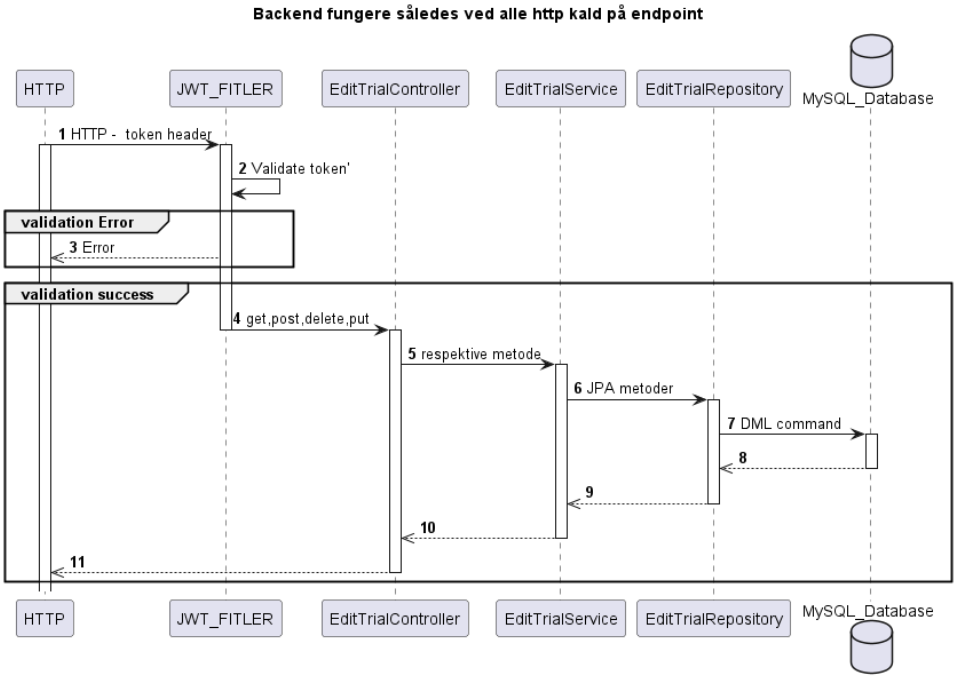
Figur 13: EditTrial UI mockup

Ud fra udveksling med virksomheden Prope, har vi fået nogle platform specifikation som vi nogenlunde har overholdt, til hvad et klinisk forsøg skulle bestå af. De steder hvor vi afviger er for at spare på tid, eller fjerne redundant information, da den ikke skal bruges andre steder.

Forsøgsbeskrivelse

Variabel	Datatype	Beskrivelse
Titel**	text	Navnet på studiet
Deltagerinformation	list/(text)	Information på ønskede deltagere (denne foretrækkes implementeres som vedhæftet fil, med mulighed for visning af resume direkte)
Kort annonce**	text	Resume af studiet til feed visning
VEK godkendelse **	ja/nej	Kræver studiet VEK godkendelse
Patientkategorier	Valgmuligheder (flere valg tilladt)	Hvilken type sygdom/diagnose studiet henvender sig til
Inklusionskriterier	list	Kriterier der tillader deltagelse
Eksklusionskriterier	list	Kriterier der ekskluderer deltagelse
Honorar	float	Størrelsen på udbetalt honorar i DKK
Kørselsgodtgørelse	ja/nej	Om transport er dækket for deltager
Transport	ja/nej	Stilles transport til rådighed
Tabt arbejdsfortjeneste	ja/nej	Godtgørelse for tabt arbejdsfortjeneste
Studiets varighed	int	Hvor lang tid forløber studiet over i antal dage
Antal besøg	int	Hvor mange besøg påkræves deltager
Længden af besøg	int	Varighed for hvert besøg i antal minutter
Slutdato for studiet	datetime / datetimeoffset	Hvilken dato afsluttes studiet
Svartid	int	Forventet svartid efter fuldendt tilmelding i antal dage

overordnet fungere siden som vist på sekvens diagrammet på figur: 14, og de underordnede metoder bliver forklarer yderligere senere. Al funktionaliteten er låst bagved en JWT, så dvs. at selvom man godt kan tilgå siden, har man ingen af funktionaliteten hvis man ikke er logget ind og valideret igennem backenden. Senere har vi tænkt os at gøre denne acces control mere rolle baseret, da vi har opsat systemet til at fungere med roller, dog bare ikke haft tid til at implementere det ordentligt.

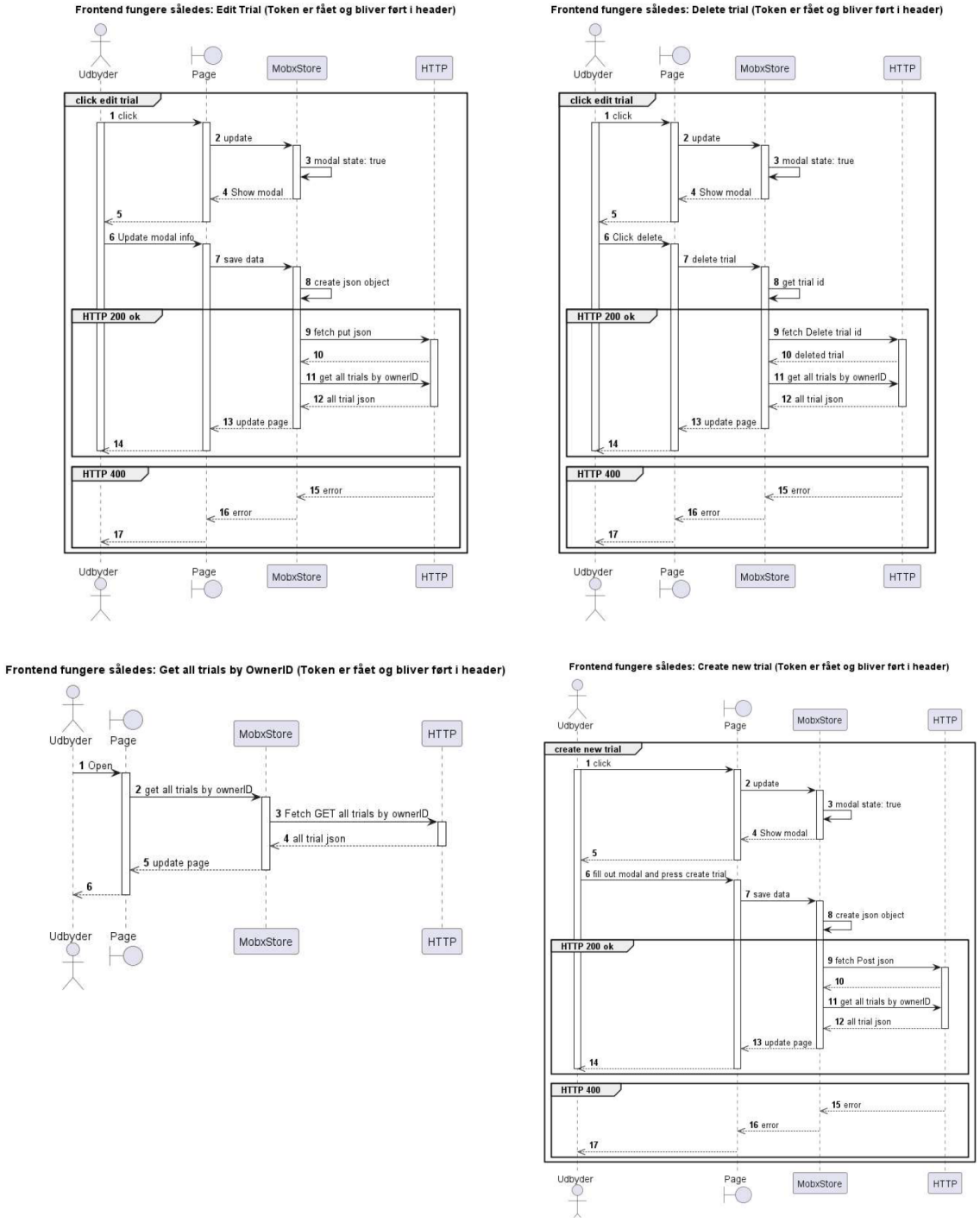


Figur 14: Overordnet funktionalitet af EditTrial

Ellers fungere de underordnede funktioner således:

- Create : Man klikker på Create Trial knappen, udfylder al informationen og create, og så ville den trial blive oprettet i systemet
- Read : Når siden åbner, refresher eller de forskellige trials ændre sig, vil der blive hentet et JSON objekt med alle trials tilhørende den kliniske udbyder.
- Update : Man klikker på Edit knappen på et trial card, ændre den information man har lyst til at ændre og klikker på update knappen, hvorefter det trial men den tilhørende trial id bliver opdateret med den information som man har ændret.
- Delete : Man klikker på Edit knappen på et trial card og klikker på delete knappen, hvorefter den respektive trial man har klikket på bliver slettet fra systemet.

Disse funktionaliteter kan ses yderlig uddybbet i sekvensdiagrammer på figur: 15



Figur 15: Underordnet funktionalitet af EditTrial

Troels (100%)

8.3 Kaspers usecase

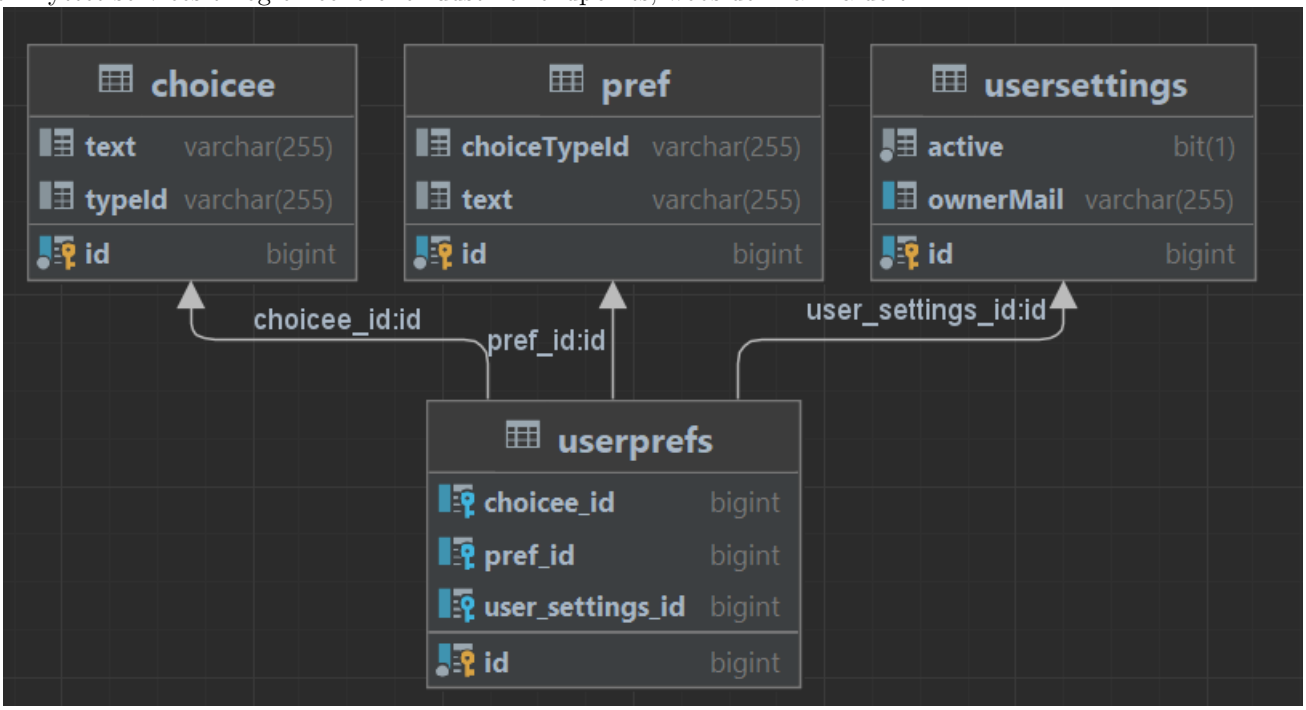
8.3.1 Kort introduktion til usecasen

UserProfile/UserPrefereneces/UserSettings dækker over samme usecase, det handler om at de kliniske brugere, altså dem som potentielt vil tilslutte sig et klinisk studie, skal sætte nogle indstillinger, som kan bruges i matchingen af studier. Aktuelt er det sat op, så brugeren kan sætte sig som aktivt søgenede eller melde sig ikke aktivt søgende, derudover kan brugeren vælge præferencer fra en afgrænset liste af

muligheder, hver mulighed har afgrænsede svarmuligheder. Dette sikrer at brugeren vælger emner og svar, som vil kunne matches med indstillinger på de udbudte studier.

8.3.2 Implementationen på afleveringstidspunktet

Integrationen mellem frontend og backend er ikke fuldført på afleveringstidspunktet. Frontenden fungerer for sig selv, hvor det er muligt at ændrer status for om man er aktivt søgende og hvilke præferencer man vil vælge til og fra. Valgene bliver dog ikke persisteret i databasen, da kaldene til backenden ikke er sat op. Frontenden bruger to lister for "preferences"og "choices"som er skrevet direkte ind i mobX storen. De valg brugeren vælger bliver kun opbevaret i state og kommer altså ikke videre til databasen. Frontenden filtrerer og tilpasser det der vises til brugeren udfra hvad der er i state.
Backenden er implementeret med fire Tables, som er relateret til hinanden, som det ses af vedhæftede billede (som er autogeneret via programmet DataGrip fra Jetbrains). Vores projekt benytter Hibernate som ORM, hibernate implementerer JPA, der er derfor oprettet repositories for de relevante entities, der er tilknyttet services til og en controller udstiller endpoints, websiden kan kalde til.



Knappen "Click me to get from API"laver et kald til backenden, som modtager Userprefs indenholdende settings, prefs og choices. Dette skrives til consollen, men er ikke nået at blive sat op, så frontenden fremviser det til brugeren. For at kaldet skal være succefuldt, så brugeren været logget ind med en mailadresse, da den fungerer som foreign key i UserSettingstabellen, userprefs tabellen har en reference til "sin"usersettings og derigennem en reference til "sin"user.

8.3.3 Hvordan kobles frontenden og backenden

Frontenden er klar til at blive koblet til backenden og de fleste funktionaliter er klar i backenden, database strukturen er sat op og der er oprettet relevante repositories og services, som let kan udvides for at implementer manglende funktioner. De to lister, som frontenden på nuværende tidspunkt benytter til at vise muligheder til brugeren kan erstattes med lister fra backenden. Der er oprettet mulighed for at tilføje flere muligheder til disse lister, dette demonsterers i ved opstart af serveren, hvor tabellerne seedes med eksempler på indhold. Strukturen i frontenden er sat op til at matche databasen, så der vil relativt let kunne laves konverteringer i mellem de to.

8.3.4 Test af nuværende implementation

Der er desværre mangelfuld test af nuværende implementation, det ville være relevant at lave automatiske test til hele systemet. Det vil være relevant med unit test af både frontend og backend, frontenden kan fx testes med jest og/eller React test library. Backenden med jUnit. Og så vil der kunne laves end-to-end test med Cypress.

Kasper (100%)

8.4 Jonas' usecase

Det er på denne side at brugeren som leder efter kliniske studier kan se de studier som er tilgængelige, og finde det studie som passer til dem. Det meste af siden bliver brugt til visningen af hvilke kliniske studier der er tilgænglige. Her kan brugeren scrolle igennem studierne og se en række nøgle informationer om studierne som de kan bruge til at danne sig et overblik over hvilke studier der er interresante for dem. Brugeren kan tilmelde sig studier de er interesserede i gennem "join trial"knappen.

8.4.1 Trial View

De højre 80% af skærmen viser kort med de forskellige kliniske studier der er tilgængelige. Studierne er blevet oprettet af "Medical Users". Studierne bliver trukket fra databasen med et fetch kald, som er rettet mod service ViewTrialController's mapping /getall som henter alle studier i EditTrial modellen. Ud over studier i databasen vises noget mock data på nuværende tidspunkt for at demonstrerer funktionaliteten.

8.4.2 Filter

På venstre side af skærmen er der en række filter muligheder. Den første mulighed er at slå automatisk filtrering til og fra. Automatisk filtrering er at de klinisker studier bliver sorteret baseret på brugerens instillinger. Hvis brugeren ikke føler de kan finde nogle studier som giver mening for dem baseret på den automatiske filtrering er kan denne mulighed slås fra, og så bliver resten af mulighederne tilgængelige.

8.4.3 Join Button

Denne knap sidder på hver kort der representerer et klinisk studie. Hvis brugeren er logget ind lader den dem ansøge om at deltage i studiet. Hvis brugeren ikke logget ind redirecter den dem til login-siden.

Jonas (100%)

8.5 Annikas usecase

Et stort system er nødsaget til at indeholde én eller flere administratorer, der kan holde styr på brugere. Når man har en åben login-form, kan man risikere, at der bliver oprettet SPAM-brugere, som skal fjernes eller lignende.

Vi har valgt at have en administrator, som kan se en liste af medicinske brugere samt rette og slette disse. Administratoren kan gå ind på hver enkelt bruger for at se detaljer om disse. Administratoren kan ændre data for hver bruger, hvilket kan være praktisk i en situation, hvor brugeren ikke selv kan finde ud af det. Dermed kan brugeren sende en besked om ting, der skal rettes (dette flow er ikke implementeret). Meningen var, at administratoren også skulle kunne se studier og rette disse. Dette er dog ikke implementeret.

En anden vigtig ting, når man har en administrator, er, at dennes side skal være sværere at tilgå. Den almindelige login-form for brugere, kan tilgås af alle og dermed kan man igangsætte brute-force attacks. Derfor er der implementeret en separat login-side som tilgår en separat tabel i databasen. Siden tilgås vha. routen admin. Hertil kunne man også sætte flere sikkerhedsforanstaltninger op såsom at den route kun kan tilgås fra en specifik IP. Ydermere, burde routen heller ikke hedde admin, men blev kaldt det i mangel på bedre. Der bliver logget, hver gang en administrator fejler sit login, hvor det vises, hvilket brugernavn, der prøver at logge ind.

Ligesom med almindelige brugere bliver administratorens password hashet med en salt. Hvis nogen opsnapper passwordet, vil det ikke være læsbart og kan dermed ikke bruges direkte. For at opnå dette, har jeg brugt SpringBoots implementation af BCrypt. Det smarte ved denne implementation er, at man ikke selv skal stå for at gemme saltet, da implementationen har en måde at gemme dette direkte i det hashede password.

Der kan ikke oprettes administratorer gennem frontenden, men en testbruger med brugernavn og password admin, admin er sat up.

Annika (100%)

9 VCS

9.1 Distribueret versionkontrol

For at sikre vores kodebase og give os selv bedre mulighed for at eksperimentere med nye implementeringer bruger vi version control (VCS). Ved brug af VCS vil det altid være muligt at se hvilke ændringer, som er lavet til koden og hvis ændringerne har haft en uønsket effekt på produktet, så kan der vendes tilbage til tidligere versioner.
For at kunne samarbejde omkring udvikling af produktets kodebase, har vi brug for et fælles repository som vi alle har adgang til så vi kan dele ændringer med hinanden. For fleksibilitetens skyld og mulighed for at have adgang til kodebasen hvor end man er vil vi gerne have et online repository. For at kunne arbejde effektivt og sikre koden hos hver enkelt udvikler vælger vi at benytte distribueret versions kontrol, hvor hver enkelt udvikler cloner online repositoriet ned og efterfølgende holder det opdateret ved at pulle ændringer fra online repositoriet. Hver udvikler kan så arbejde med branches fra sit lokale repository og lave løbende commits undervejs uden behov for at "skulle forbi"online versionen. Udvikleren kan så når branchens formål (bugfixing, ny feature eller andet) er opfyldt pushe til det fælles repository, så alle andre udviklere har adgang til det.
Vi har valgt at benytte git til vores distribuerede VCS, da det er det mest udbredte samt er meget anerkendt og nemt at finde hjælp og vejledninger til. Vi har valgt at benytte gitHub til at hoste vores online repository, da det ligeledes er meget udbredt og let at finde vejledning til. Vi benytter et public repository, da det giver gratis mulighed for blandt andet at benytte GitHub Actions, som vi benytter til automatiserede builds, test og deploys. Vi benytter GitHub Actions som vores CI/CD værktøj, da det giver et tydeligt overblik over hvad der sker, når CI/CD pipeline beskrivelserne ligger sammen med resten af kodebasen.

9.2 Branch strategier

En risiko ved distribueret VCS er at udviklernes lokale branches kommer til at afvige fra hinanden, hvilket kan give tidskrævende merge conflicts og gøre integration imellem forskellige features svær. Vi har derfor fokus på at holde development life-cycle nede og "hurtigt"få lokale branches merged tilbage til den fælles. Det er derfor vigtigt at lave små fixes og features i hver branch, dette mindsker kompleksiteten og risikoen for problemer med integration med de andre udvikleres arbejde. Samtidig vil det give mulighed for hyppige deploys og derved mulighed for tidlig feed-back på nye features eller ændringer.
Vi har valgt at have to gennemgående branches i vores online fælles repository, master og dev. Dev er den branch, som vi som udviklere hyppigt merger udfra og tilbage til. Master er den branch, som bliver deployet og derved kommer ud til brugerne. På disse to branches er der opsat forskellige regler - styret fra settings i GitHub - samt actions - styret fra Workflows i GitHub.
Da projektet udføres som studieprojekt, hvor vi kan arbejde med det på mange forskellige tidspunkter, så aftalte vi at det skulle være let at få ændringer ind på dev uden at skulle afvente andre. Der er derfor ingen krav om review eller approval på dev. For at sikre at der ikke kommer kode ind på dev, som kan ødelægge funktionalitet, så benytter vi pull request, hvor der kører automatiske test af builds og de medfølgende test filer. Hvis disse test bliver bestået, så kan udvikleren selv merge ind på origin/dev, lukke sin branch og pulle origin/dev ned for at kunne branche ud fra en opdateret local/dev. Hvis testene ikke består, så retter udvikleren til og lader det teste igen. Inden hver pull request er det vigtigt at pulle og merge origin/dev ind i sin egen branch, hvis der kommer merge conflicts så skal de resolves med respekt for det som allerede ligger på origin/dev. Dette sikrer at man ikke overskriver andres ændringer med mindre det virkeligt er intentionen.
Da master bliver frigivet til brugerne har vi sat strengere krav op for at merge ind til denne. Merges til origin/master kan kun ske via et pull request, som vi altid vil lade være fra origin/dev. Grundet tidligere beskrevet strategi for origin/dev, så ved vi at den kode som kan komme ind på master kan builde og har bestået vores tests. Pull request til master kræver approval fra minimum en anden udvikler. Dette sikrer at minimum to udviklere har gennemset den kode som bliver frigivet. Samtidig så sikrer reviews også at vi understøtter læring ved at se hinandens kode og være åben over for kommentarer og forslag til ændringer. Optimalt set blev læringen mere understøttet på dev branchen, men grundet arbejdstidsforskellene har vi valgt ikke at gennemse hinandens kode på dev.

Kasper (100%)

10 CI/CD

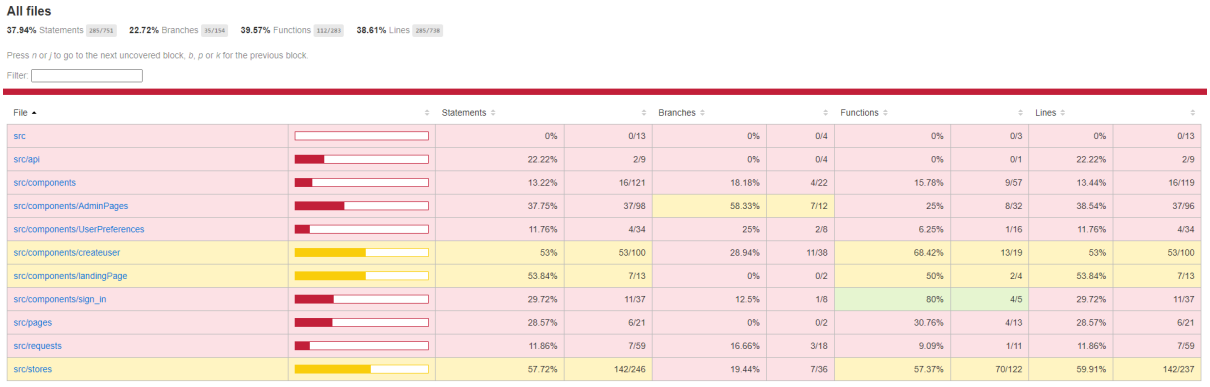
Et kernekoncept når man snakker om DevOps er Continuous Integration (CI) and Continuous Delivery (CD) Dette koncept er forsøgt efterlevet igennem de 13 uger dette projekt er blevet udviklet i.
Continuous Integration er en praksis som omhandler at man bliver ved med at lave små kode commits til sin master branch igennem hele projektets levetid. I praksis snakker vi mange små kode commits om dagen og jo støre ens team bliver, jo flere commits vil man typisk lave. Et commit vil herefter starte en automatisk test, som vil evaluere koden og en succesful test vil trigger et nyt build af koden.
En fordel ved at merge små ændringer med en høj frekvens er, at det minimerer risikoen for konflikter med andres kode (merge conflicts), og skulle dette alligevel ske, er det noget nemmere at overskue, hvad der er galt og hvordan man løse det, da man ikke kigger på flere tusind linjer kode, men i langt nemmere grad kan isolere problemet.

Continuous Delivery kan ses som en forlængelse af CI. Når de automatiske tests og builds er færdige, overtager CD. Her handler det om, at CD gør klar til at deployerer koden, og klargøre serveren med de dependencies der nu måtte være. Det kunne f.eks være, at pakke den builded kode ned i en docker container, og pushe det til docker hub, hvorfra den server, hvor ens applikation kører på, nemt kan hente det nye image og deployere det på serveren.
Ved at have en fuld automatiseret deployment pipeline, kan man deployere når som helst. Ultimativt betyder det, at den tid der går fra idé til deployering, bliver kraftigt reduceret, og selve opgaven med at deployerer bliver en rutine opgave, som alle udviklere på holdet er i stand til at varetage.

Johannes (100%)

11 Evaluering af produktet

Under implementationen har vi haft meget fokus på at få inkorporeret alle teknologier. På denne måde har vi fået udviklet et system, som har alle elementer, der skal til for at lave en fullstack hjemmeside samt sørget for at vi kan have hjemmesiden i produktion. Dette inkluderer en masse forskellige værktøjer og teknologier, vi har skulle sætte os ind i at bruge. Derfor har der også været nogle ting, hvor vi ikke er nået helt i mål. Dette inkluderer eksempelvis Test Coverage. Her er det ideelt med en coverage på 80% . Dog har vi en coverage på 38% i frontenden og 26% i backenden. Dette kan ses på hhv. figur 16 og 17.



All files

37.94% Statements 285/751 **22.72%** Branches 35/154 **39.57%** Functions 112/283 **38.61%** Lines 285/738

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File		Statements		Branches		Functions		Lines	
src		0%	0/13	0%	0/4	0%	0/3	0%	0/13
src/api		22.22%	2/9	0%	0/4	0%	0/1	22.22%	2/9
src/components		13.22%	16/121	18.18%	4/22	15.78%	9/57	13.44%	16/119
src/components/AdminPages		37.75%	37/98	58.33%	7/12	25%	8/32	38.54%	37/96
src/components/UserPreferences		11.76%	4/34	25%	2/8	6.25%	1/16	11.76%	4/34
src/components/createuser		53%	53/100	28.94%	11/38	68.42%	13/19	53%	53/100
src/components/landingPage		53.84%	7/13	0%	0/2	50%	2/4	53.84%	7/13
src/components/sign_in		29.72%	11/37	12.5%	1/8	80%	4/5	29.72%	11/37
src/pages		28.57%	6/21	0%	0/2	30.76%	4/13	28.57%	6/21
src/requests		11.86%	7/59	16.66%	3/18	9.09%	1/11	11.86%	7/59
src/stores		57.72%	142/246	19.44%	7/36	57.37%	70/122	59.91%	142/237

Figur 16: Code coverage i frontend

Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	26.2% (11/42)	14.2% (31/218)	13.3% (57/427)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.probe.probe_springboot.CampusNet	0% (0/2)	0% (0/7)	0% (0/29)
com.probe.probe_springboot.authentication	0% (0/7)	0% (0/43)	0% (0/104)
com.probe.probe_springboot.controller	0% (0/4)	0% (0/25)	0% (0/27)
com.probe.probe_springboot.exceptions	0% (0/6)	0% (0/21)	0% (0/22)
com.probe.probe_springboot.filters	0% (0/3)	0% (0/8)	0% (0/34)
com.probe.probe_springboot.exceptions.EditTrials	16.7% (1/6)	16.7% (1/6)	16.7% (1/6)
com.probe.probe_springboot.service	40% (2/5)	24.1% (7/29)	21.7% (23/106)
com.probe.probe_springboot.model	83.3% (5/6)	20.3% (12/59)	21.3% (13/61)
com.probe.probe_springboot	100% (1/1)	50% (3/6)	40% (8/20)
com.probe.probe_springboot.model.UserPreferences	100% (2/2)	57.1% (8/14)	66.7% (12/18)

Figur 17: Code coverage i backend

Annika (100%)

12 Ændringer undervejs

12.1 Ændring af server

Da projektet i sin spæde start blev deployeret for første gang, blev det delpoyeret på Github Pages, som egner sig fint til mindre statiske sider. Det skulle dog hurtigt vise sig, at applikationen voksede sig større og derfor var det nødvendigt at deployere den på en anden server struktur. Derfor blev Azure valgt, og det gik fint - i hvert fald til at starte med.
Som projektet skred frem, skulle det også her vise sig, at den mængde RAM og CPU kraft som der her blev stillet til rådighed ikke rakte. Response tiden på http request lå mellem 12 og 20 sekunder, hvis man vel og mærket var heldig at få en response. Omkring uge 12 gik serveren helt ned, så i stedet for at forsøge at genstarte serveren, blev det besluttet at sadle om og deployere applikationen på en helt ny server - købt og betalt uden om DTU.
Dette viste sig at være en god beslutning, da applikationen nu virker efter hensigten, med en rimelig respons tid.

Johannes (100%)

13 Ny-erhvervet viden

Vi har stiftet bekendskab med en lang række moderne værktøjer som er nyttige til at udvikle en fullstack web applikation. De er beskrevet undervejs i rapporten. Derudover har vi udviklet forståelse for en masse ideer og principper som kan forbedre udviklingsprocessen når det system man arbejder med når en kompleksitet der gør at det ikke er muligt at opretholde fuld forståelse for hele systemet på individuel

basis. Vi har lært om hvorfor det er vigtigt at der er kommunikation mellem development og operations når noget går galt. Vi har fået et indblik i hvordan arbejdsprocessen reelt ser ud på virksomheder.

Jonas (100%)

14 Videreudvikling

14.1 Dataindsamling og Algoritmer

Formålet med vores platform er at parre potentielle deltagere med udbydere af kliniske studier, og derfor er en af de vigtigste steder hvor der altid er plads til forbedring vores evne til at vise de bedste studier først for en hver given bruger. Hvis applikationen blev released og fik en voksende brugerbase vil der være rig mulighed for at samle og bruge data for de brugeres brugsmønstre og bruge det til at implementere algoritmer hele tiden kan blive bedre til at forudsige hvilke studier en bruger vil være interesseret i. På nuværende tidspunkt har vi kun frontend og backend frameworket til at filtrerer og sorterer visningen af studier, men ingen reel brugerdata til at implementerer en algoritme ud fra. Det er et oplagt sted at videreudvikle vores løsning fordi det har en høj indvirken på kvaliteten af brugeroplevelsen.

Jonas (100%)

14.2 E-mail konfirmation ved oprettelse af bruger

Når en bruger bliver oprettet, vil det være en god ide, at implementerer en service, som sender en konfirmations e-mail ud til den angivet e-mail adresse, som brugeren skal bekræfte og derigennem kan man verificere, at brugerens e-mail adresse eksisterer.

Johannes (100%)

14.3 E-mail notificationer

Derudover er email også oplagt at bruge til at sende notifikationer til brugere. Når der bliver oprettet nye studie som kan være relevante for brugere eller ændringer med studier som de er interesserede i.

Jonas (100%)

14.4 Mit ID implementation

For at sikre at brugeren virkelig er den som vedkommende påstår at han er, ville Mit-ID være en oplagt implementation - specielt taget projektets formål i betragtning, der er det vigtigt, at man kan stole på, at de oplysninger, som er angivet, rent faktisk er korrekte.

Johannes (100%)

14.5 Import af medicinsk journal

Med tilstrækkelig samtykke fra brugeren er det potentielt muligt at hente journaler fra sundhedsvæsenet som kan bruges til at hjælpe brugeren med nærmere at finde et studie. Dette kan være en stor hjælp eftersom det kan afhjælpe behovet for at potentielle deltagere er afhængige af selv at indhente og forholde sig til information der kræver teknisk ekspertise at forstå.

Jonas (100%)

14.6 Rollebaseret acces control

Som vores program fungere lige pt. har vi ingen rolle baseret acces control, vi har dog programmeret det meste af det allerede, vi må bare konkludere at den sidste del af opgaven blev glemt og derfor ikke færdiggjort, samtidig med at den fungerende JWT access control ikke nåede at blive implementeret på alle use-cases. Derfor har vi valgt ikke at færdiggøre resten af den rollebaserede access control, således at folk stadig har mulighed for at implementere den fungere JWT acces control. Opgaven dog næsten færdig, da vi allerede registrerer roller i systemet, fører dem med i vores JWT Token og også router en lille smule efter dem, så der mangler bare lige at blive lavet et overordnet filter som router/kontrollere rettigheder ud fra rollerne, således at man ikke kan komme ind på sider man ikke har rollen til.

Troels (100%)

14.7 Udvidelse af logger

Et af de steder, der er brug for en indsats er ift. logning. Den nuværende logning er langt fra fuldt implementeret, og er i sin nuværende form mere et proof of concept.

Johannes (100%)

14.8 Overvågningsudvidelse

Ligesom med logning, er der rigelig med plads til at udvide overvågningen af applikationen, så den kommer til at overvåge større dele af applikationen.

Johannes (100%)

15 Konklusion

Vi har udviklet en del af en større løsning der er uden for et 5-points kursus scope, og derfor er der rige muligheder for at viderudvikle og udvidde vores løsning. Authentication og notifikationer via email, Mit-ID authenticaiton, sorterings algoritmer, import af medicinske journaler og meget mere. Desuden er der rig mulighed for at udvide eksisterende features ved hjælp af feedback fra eksperter i hvordan kliniske studier fungere og potentielle brugere.
Vores pipeline er fuldt automatiseret og fungerer i det store hele godt. En forbedringsmulighed er et staging miljø hvor vores dev branch kan blive deployed på samme måde som vores master branch gør i production.
Vi har automatiseret vores pipeline, deployment og testing, vi har holdt batch size på vores commits lille, og vi har generelt holdt vores deployment lead time lille og det har gjort at vi har kunne skrive mere kode der kan bruges hurtigere. Devops er industristandarden for god software udvikling og det er det fordi det virker.

Jonas (100%)

A Bilag

A.1 Arbejdsfordeling

A.1.1 Johannes´ opgaver

- Landingpage
- Kort karrusel på landingpage
- Oprettelse af brugere frontend og backend
- Login frontend og backend
- JWT handler
- Monitorering med Prometheus og Grafana
- Test af oprettelse af bruger
- Test af bruger login
- Dele af deployeringslinjen med Git Actions
- Authentication filter
- Custom Exceptions til Create user og User login
- Indledende logning med log4j2

A.1.2 Troels´ opgaver

- Edittrial Siden
- EditTrial repo osv.
- Cors filter
- DTU CAS implementation
- Test af editTrial
- E2E test af editTrial
- Exceptions til editTrial
- Dele af deployeringslinjen med Git Actions

A.1.3 Kaspers opgaver

A.1.4 Annikas opgaver

- Admin login + admin side frontend + backend
- Ret bruger, frontend + backend

A.1.5 Jonas´ opgaver

- View trial frontend
- View trial backend integration
- Trial view filtreringsmuligheder