Danmarks Tekniske Universitet

It og kommunikation
62581

IT delopgave 2

Gruppe 4
Andreas Bach Berg Nielsen s205869
Emilie Bracht Nielsen s205484
Julie Falsig Valeur s205485
Naveed Imam Shah s205491
Troels Engsted Kiib s205492
Jesper Clement Aaløse s205488

December 27, 2023
Github : `https://github.com/Troels21/IT3-Delopgave-2`
Hjemmeside: `http://grp4.it3.diplomportal.dk:8080/EPJ`

Abstract

In this report, the task to add backend to the existing code was given. That was accomplished using Javascript, Java-servlets, a SQL-database and API. Furthermore, we will be explaining and discussing the process of making this a part of our program. As well as discussing the challenges that arose and what we will be adding in the future.

Indholdsfortegnelse

1 Introduktion

I denne rapport vil der blive videreudviklet på programmet fra delaflevering 1. Gennem processen til at udvikle koden til denne delaflevering, har vi indført Javascript, en SQL-database og en API. Dette har givet os mulighed for, at tilføje en masse nye funktioner, såsom login-validering og exceptions, som begge vil blive gennemgået i senere afsnit. Derudover vil der blive diskuteret, hvilke problemer vi er stødt ind i, gennem processen, og hvilke mulig forbedringer der kunne indføres i fremtiden.

1.1 Formål

Formålet med denne delaflevering er, at yderligt forbereder og udvikle vores kode. Dette skal opnås, ved at tilføje backend til koden, i form af javascript, en SQL-database og api. Der udføres en rapport, for at forstå designet og analysen af programmet, selve implementeringen og en afprøvning af programmet.

2 Kravspecifikation

Til denne gang ønskes der et minimalt fungerende system på hjemmesiden. Det betyder, at der en form for logind-validering. Når en aftale oprettes, skal dataen lægges på, og kunne hentes fra serveren, og det skal findes frem af en bruger, ud fra kalenderen.

3 Analyse

I dette afsnit vil vi analysere de ændringer, vi gerne vil foretage til vores program, og hvilken indflydelse det har på programmet. Der vil blive drøftet den betydning det har, at vi nu implementere javascript og en backend i vores program. Derved vil alle de statiske elementer i vores mock-up fra sidste rapport, nu blive dynamiske, som har noget funktionelt, at gøre brug af. Dette er f.eks at vi nu skal inkorporere en database, hvor vi gemmer konsultationerne og login brugerne. Ved at gøre det, kan vi hente alle konsultationerne, og derved vise dem i vores hjemmeside. Ved at gemme login oplysningerne, kan vi lave noget sikkerhed, ved at validere, om brugeren har de korrekte oplysninger. Så ved implementering af javascript, API og databaser, vil vi kunne lave et funktionelt program, der kan vise, og lave konsultationer.

3.1 Klassediagram

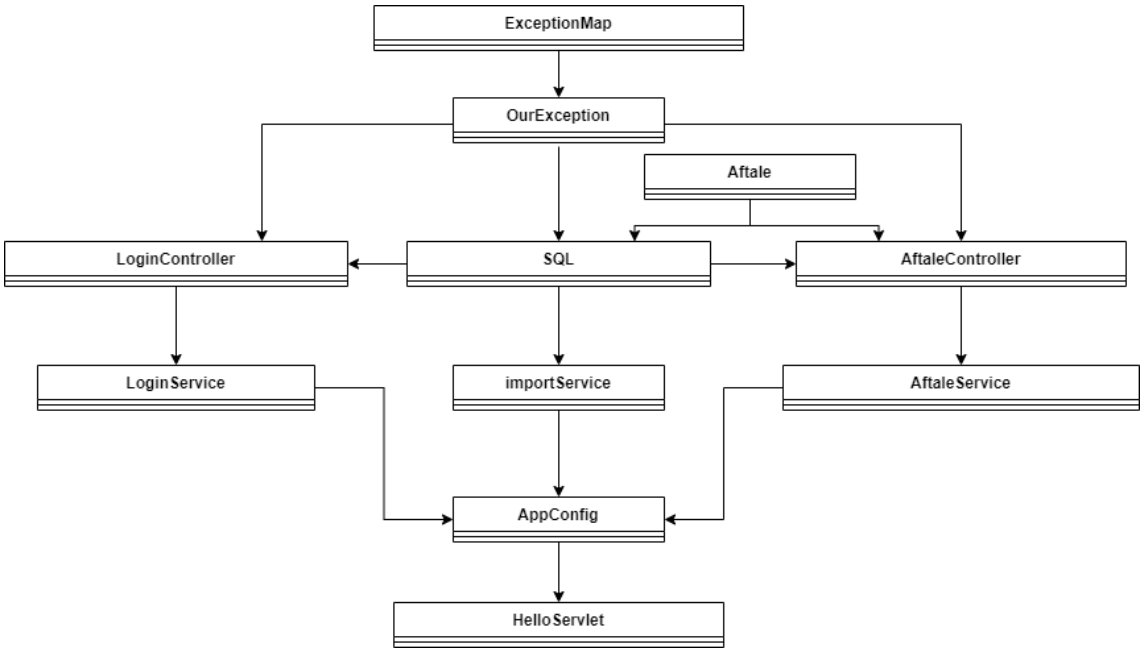

Figure 1: Klassediagram

Dette er hvordan vi tænker, vores API skal opdeles. Her har vi controllere til vores services, og SQl til at håndtere forbindelsen til databasen. Til sidst har vi vores Exception klasse, som er en klasse, hvor vi laver vores egen exception, som kan blive catched.

3.2 Kalendar

Vi har i første udkast, lavet et meget stort skema, hvor der er en masse felter, som fungere som tider på de forskellige dage. Dette fungere dog meget dårligt, da der ikke er noget javascript implementeret. Ved at tilføje javascript, kan vi nemlig gøre kalenderen dynamisk, ved at bruge date() objektet. Derfor skal vi lave hele kalenderen om. Det kunne være muligt, at implementere det på samme måde som før, men dette er upraktisk. Der ville være en masse tomrum, og være betydeligt sværere at skifte dato. Derfor har vi valgt at gøre dem separat. Dette gøres ved at dele kalenderen og skemaet op i to, hvor vi har kalenderen, der kun er til at vælge dato med, og skemaet der kun er til at vise konsultationer med.

3.3 Journal

I vores mock-up var journalen en helt side for sig selv. Dette medførte at man gik væk fra startsiden, over i en ny side. Dette er ikke den bedste løsning, da det giver mere brugervenlighed, at vi ikke laver en hel ny side, men en lille pop-ud, som vores journal. Det vil sige, at hvis der bliver klikket tilføj, så kommer vi ikke ind i en ny side, men i et popud. Vi ved, at vores brugere er både ældre og unge. Derfor er denne løsning mere enkel at tage holdning til. Da hvis den åbnede en hel ny side, kunne de ældre blive bange for, at der er åbnet noget, som ikke skulle, eller hvordan man vender tilbage. Der er heller ikke nogen grund til at bruge en hel side, til at indskrive data, som alligevel skal fremvises på startsiden.

3.4 Aktivitetsdiagram

Aktivitetsdiagrammet er ændret, således at det nu stemmer overens med de ovennævnte tanker.

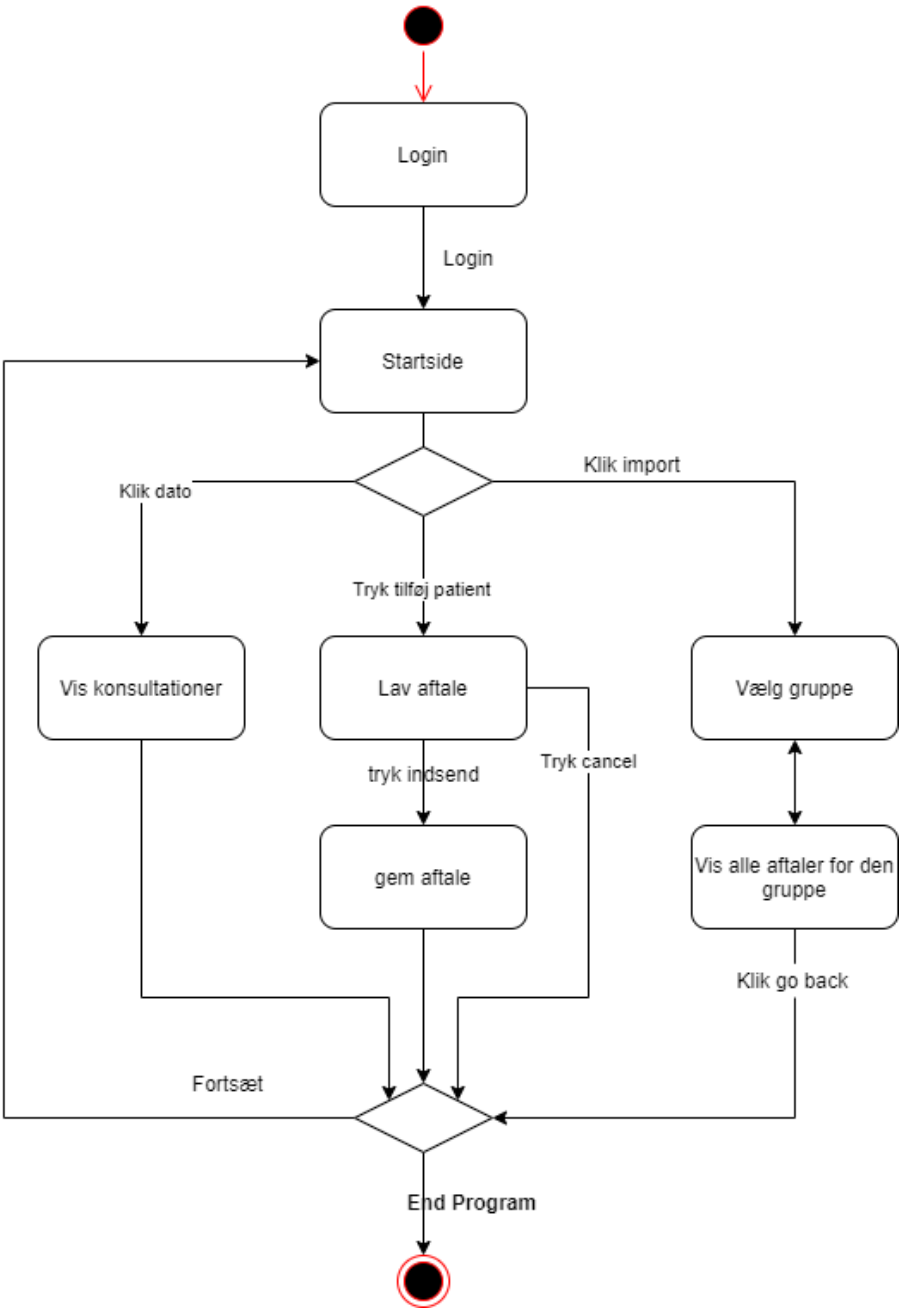

Figure 2: Aktivitetsdiagram

Det viser, hvordan de forskellige forløb er, og skal foregå. Her kan der tydeligt ses, at simpliciteten er steget, i forhold til sidste rapport, ved at have journalen som en popup, istedet for en side for sig selv. Det viser også hvor simpelt det nu er, at tjekke konsultationer. Der er ikke nogen "scroll", af hundrede tider for at finde en enkel, men blot at klikke på en dato, også få fremvist de tilsvarende konsultationer.

3.5 Tilstandsdiagram

Tilstandsdiagrammet er lavet, for at afdække de eksisterende processor og protokoller, der er i programmet, hvilket angiver den tilstand, brugeren er i.

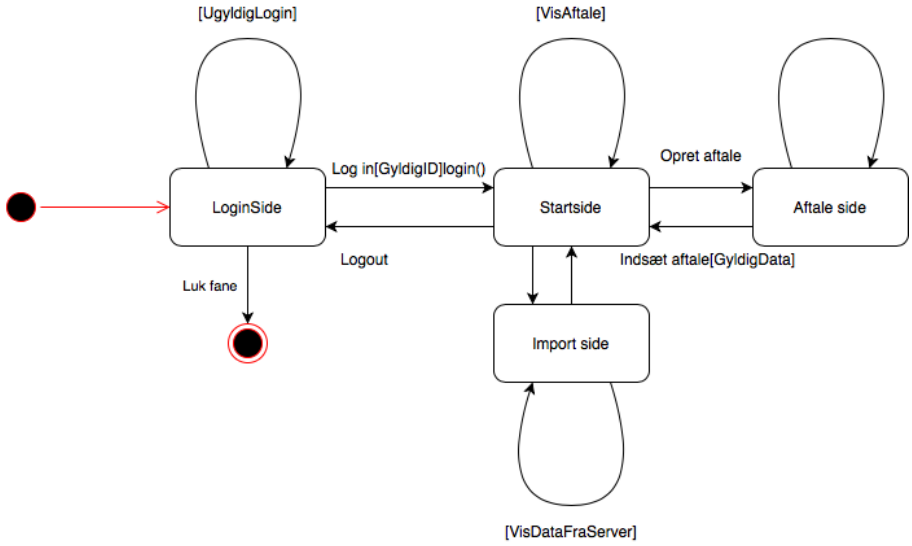

Figure 3: Tilstandsdiagram

4 Design

I dette afsnit vil vi gennemgå vores design af programmet. Dette er alt fra HTML, CSS og JavaScript, til JAVA og Databasen

4.1 Design klasse diagram

Dette klassediagram viser hvordan vi har designet vores JAVA-kode, hvordan vi har opdelt det i klasser, og hvilke metoder og attributter de indeholder.

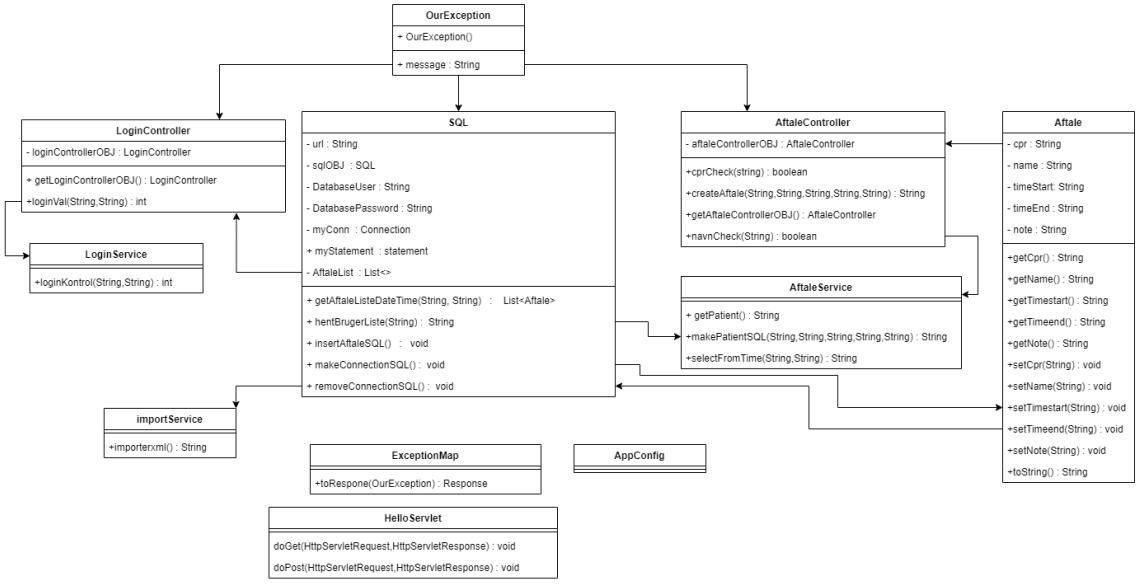

Figure 4: Design Klassediagram

Her kan vi se, hvordan vi har tænkt os at dele vores API op, så vi opfylder SOC.

4.2 PatienterJournal → AftaleKalender

Vi har valgt, at lave vores program om således, at der ikke længere er patienter i programmet, men derimod aftaler, der indeholder patienter i systemet. Vi har lavet dette fordi vi egentlig ikke laver patienter, men derimod aftaler, der skal ligge i en kalender. Senere vil vi skille patienterne fra aftaler, således at patienterne er unikke, og kan have flere aftaler. Denne funktion er dog ikke nødvendig nu, så derfor har vi lavet patienterne til aftaler i stedet.

4.3 Database

Vi vil gerne gemme vores data i en SQL database. Databasen skulle gerne være en separat maskine, så databasen ikke ville gå i stykker, hvis hjemmesiden går i stykker. Vi vil gerne lave en database uden relationer, da vi ikke har et behov for denne funktion endnu. Senere vil dette dog designes. Ud fra vores lille behov, har vi lavet et hurtigt ER diagram, som kort beskriver, hvordan vores tabels skal se ud.

4.3.1 ER diagram

ER diagrammet viser en oversigt over hvilke tables, med tilhørende entities og attributter, der er oprettet i SQL-databasen.

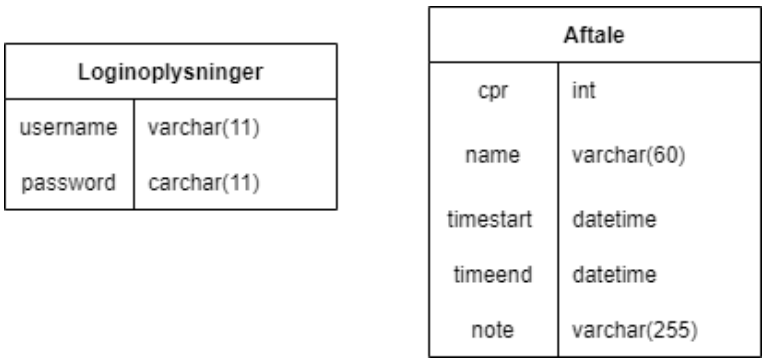

Figure 5: ER diagram

4.4 Frontend

Frontend er vores HTML,CSS og Javascript. Dette bliver designet, således at vi kan fremvise data fra vores backend, gennem en API. Vores frontend er yderligere delt op i de sider, man kan være i på hjemmesiden dvs. login, startside og import. Frontenden fungere således også som vores view del af MVC.

4.4.1 Login

Login siden bliver brugt til at logge ind med. Dette kunne før gøres med hvilke som helst navn og kode, men da vi nu har en database, kan vi login validere.

4.4.2 LoginValidering

loginvalidering fungere således, at vi har brugeroplysninger gemt i en tabel, i vores SQL-database, som vi så kan fetche gennem vores API. Dette gør, at vi nu kan sammenligne hvad

burgeren har indtastet, med hvad vi får fra databasen, og kun hvis de stemmer overens, kan man logge ind, ellers så giver den en fejl.

4.4.3 Startside

Startsiden er den side, man kommer til at være mest på. Det er den side, der bliver brugt til at se, læse og oprette konsultationer. Her kan man ved, klik på en dato, se alle de givne konsultationer, ved brug af tilføj knappen, oprette konsultationer, og ved brug af import knappen, skifte til import siden. Dernæst kan man også logge ud.
Vi skal ved et klik på en dato, kunne lave et fetch, med datoen som parametre, og derefter hente en aftaleliste med kun den tilsvarende dato. Dette gør, at vi så har et dynamisk skema, som kun viser x antal konsultationer, og ikke alle der kan være.

4.4.4 Import

Import siden skal illustrere, at vi kan hente data fra de andre grupper i form af XML, men da de ikke er så langt, har vi valgt at fetche vores egen aftaleliste, som kan tilgås i form af XML, så vi viser alle vores konsultationer, som er i databasen.

4.5 Backend

Backend er alt den bagvedliggende logik i vores program. I backenden vil vi gøre brug af MVC, det vil vi opnå ved at lave "service" klasser der er vores API med tilhørende controller klasser som indeholder selve logikken. Separat fra disse har vi vores SQL klasse som indeholder de metoder som bruger af controller klaserne til at interagere med databasen. Model komponenten er vores Aftale klasse som er den model vi bruger til at lave vores aftale objekter.

4.5.1 Restful-API

Vi vil gøre brug af et restfull API, der skal fungere som et mellemled for vores Frontend og Backend, ved hjælp af f.eks. get og post gør APIen forskellige funktionalitet tilgængelige i interfacet. API er altså den softwaregrænseflade vi gerne vil bruge til at interagere med eksempelhvis vores database. Specifikt skal listen af brugere kunne fetches så vi kan lave en fuldt funktionel loginside, dernæst skal aftaler kunne tilgås og skrives til så der både kan udføres kontrol om en given tid er optaget samt vise aftalerne på siden og oprette nye aftaler.

4.6 Roboustness diagram

Robustness diagrammet er lavet, for at give forståelse for kommunikationen mellem de forskellige objekter i programmet. De giver en forbindelse mellem use case diagrammet , domæeklasserne og MVC-arkitekturen[1], som giver separation of concerns.

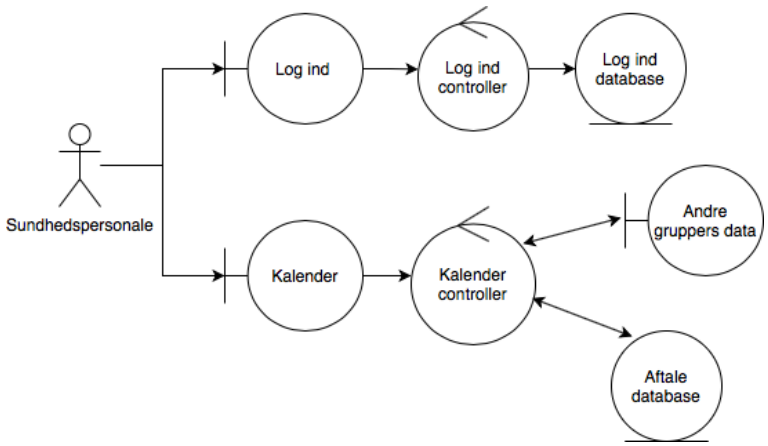

Figure 6: Robustness diagram

5 Implementation

I dette afsnit vil der redegøres for

- implementationen af MVC pattern
- database
- frontend elementer
- RestAPI
- backend

5.1 Frontend

Der er blevet implementeret en række nye elementer i programmets frontend, primært ved at implementere Javascript, og på den måde gøre frontenden mere dynamisk, og skabe funktionalitet. Desuden har vi tilføjet og ændret i det visuelle, altså brugergrænsefladen, dvs. vi har ændret på vores startsides, og tilføjet en importside.
Det generelle design af vores frontend, består af HTML dokumenter med tilhørende CSS og Javacript. Den grundlæggende struktur, er altså lavet i HTML dokumenter, og derefter bruges CSS til at beskrive hvordan HTML elementerne, skal vises. I et tilhørende Javascript dokument laves de funktioner, som elementerne fra HTML kalder. Javascript funktionerne har noget af deres funktionalitet i Javascript koden, men deres vigtigste funktion er, at interagere med backenden gennem API'en. Dette gøres ved hjælp af fetch kald.

5.2 Startside

Vi har implementeret en kalender på startsiden, som vi har konstrueret fra bunden i HTML, i stedet for at bruge de allerede eksisterende biblioteker, til dette. Dette har vi gjort, så vi selv nemt kan tilføje funktioner til knapperne. Kalenderen bliver konstrueret når startsiden indlæses, ved hjælp af makecalender(); funktionen, som opdatere HTMLen. Der er også implementeret et "tider på dagen" felt, som hænger sammen med kalenderen. Når man trykker på en given dato i kalenderen, vil de aftaler tilhørende denne dato hentes fra databasen, og udskrives som en liste i det tomme felt. Til dette felt hører en refresh knap, som ligesom navnet hentyder, genopfrisker den valgte dato, hvis en ny aftale skulle være opstået.

Vi har også implementeret en knap, der laver en ny aftale, denne knap åbner en ¡div¿ ved at ændre dennes status. Dette giver brugeren mulighed for, at oprette en ny aftale. I aftalen bruger vi et datotids objekt til at vælge dato og tidspunkt. Når man har valgt dato og tidspunkt, kører der en funktion, som kontrollere om datoen ligger i weekenden, og runder tidspunktet til nærmeste 15 min interval, og fylder et disabled felt, og to hidden felter ud, der viser hvilken tid man får, og bruges til at gemme tiderne. På submit knappen bliver der lavet et HTTP post, med parametrene til vores API. Vi har for en sikkerheds skyld, implementeret en Javascript funktionen, som skriver den nuværende dato og tid øverst på siden. Til sidst er der lavet en import knap, som skifter siden til importsiden.

5.2.1 Import-siden

Vi har lavet en separat side, som vi kalder import siden, som er lavet til at illustrere, at vi kan hente data fra en server. Der er lavet en drop-ned menu, hvor man vælger hvilken gruppes server, man vil tilgå, gennem deres API. Dernæst hentes og udskrives dataen, fra den pågældende adresse, i det grå felt midt på siden. Senere skal dette kunne hente dataen, og vise den fra de andres servere, men da disse endpoints ikke er konstrueret og informeret, er der kun mulighed for, at se et kald på vores eget.

5.3 Backend

Vi har implementeret en restful-API, et MVC patteren i backenden, en loginvalidering, en exceptionhåndtering og en database med MySQL. Vores backend er implementeret i java, samt Docker til MySQL server.

5.3.1 Restful-API

Vores API skulle have 3 indgangsvinkler:

- Vi skulle have en grænseflade til vores data således, at de andre gruppe kunne modtage vores data i en XMl format, denne grænseflade ligger under /data/liste
- Vi skulle have en grænseflade til vores SQL database, så vi havde mulighed for at oprette aftaler, og hente aftaler for en specifik dag, ligger under /data/liste/listeSQL
- Vi skulle have en grænseflade til vores importfunktion, da vi gerne ville videreudvikle denne funktion til, at kunne vise de andres gruppers data, denne ligger under /data/import.

Vores API er stateless, da vi ikke gemmer information i vores get request, samt at alle request er separate og uforbundet.

5.3.2 MVC

I vores backend, har vi forsøgt at skabe en Model-View-Controller pattern, da det visuelt skaber en god separation of concerns. Vi har lavet controller klasser(som fungere som en controller), som snakker med vores database og aftaleklasse(som fungere som en model), og bliver kaldt af vores service klasser (som fungere som en view til vores backends api).

Aftale klassen indeholder nogle attributter, en toString metode og getter/setter for dets attributter.

5.3.3 Login-validering

Vores login-valideering er blevet implementeret igennem javascript samt MySQL. Det metoden gør er, at vi laver et fetch kalde med det indtastede brugernavn, som en searchquery til vores brugerliste på SQL databasen igennem vores API.
Hvis brugeren eksistere, laves der en kontrol om det indtastede password, er det samme som det på serveren, der passer til brugernavnet.

- Hvis password stemmer overens, returneres der et 1 til javascript, som vidresender brugeren til næste side.
- Hvis password ikke stemmer overens, returneres der et 0 til javascript, som smidder en alert til brugeren, om at username eller password er forkert.

Opsætning af vores login-validering gør, at den stadig er en smule ubrugelig, da man selv ville kunne ændre den til at give en adgang, samt at vi gemmer brugerens password i clear text, og ikke som en hashet-værdi i databasen.

5.3.4 Exception-håndtering

Da vi gerne ville have mulighed, for at lave vores egne exceptions, lavede vi en hurtig exception, kaldet ourException. Denne exception indholder kun en besked, med en getter og setter således, at beskeden kunne ændres. Vi lavede den på denne måde, da vi følte, at der ikke var det store behov for store komplicerede exceptions, og at vores behov for exceptions bare var, at fører en forklaring vider til brugeren. Derfor følte vi, at en simpel, hurtig og fleksibel metode, der kunne ændres til at give den information der skulle f.eks:

```
OurException ex = new OurException();
        ex.setMessage("Brugernavn eller Password mangler");
        throw ex;
```

For at exceptionen skulle kunne laves, implementerede vi en exceptionmapper, der smed en fejl 400, da det primært var syntax fejl, vores exception håndterede.

5.3.5 Database

Vi har oprettet en MySQL database på en ubuntu linux virtuel maskine, der kører på en server på DTU. Vi hentede Docker på maskinen, og med Docker hentede mysql og kørte den i en container. På MySQl serveren implementerede vi vores ER diagram, fra designafsnittet.
Vi brugte MySQL JDBC connector til at skabe/slukke en connection til vores database. Vi bruger en metode til at skabe en connection, og en metode til at afslutte connectionen hver gang, vi skal bruge vores database, således at vi ikke har en generel tilslutning til serveren, der altid er igang.

Vi har lavet en metode, der kan hente dataen fra brugerlisten, en metode der skubber data til patientlisten, en metode der henter alt dataen fra patientlisten og til slut, en metode der henter dataen fra patientlisten for en specifik dag. For at styrke vores database imod SQL-injection, har vi implementeres vores SQL queries, ved brug af preparedstatements. Fordi at parameterværdierne er overført senere i queriet, ved brug af en anden protocol, og derfor er mere modstandparat.

6 Afprøvning

I dette afsnit vil der udføres opgavebaseret usability testing på en testperson.

6.1 Usability testing

En case bliver fremvist til testpersonen Mia, som også lavede usability testing på prototypen. Hun vil igen blive observeret.
Testpersonen skal logge ind, og oprette en ny aftale med valgfrie informationer. Når aftalen er indsendt, skal vedkommende finde sin oprettede aftale i kalenderen, og derefter finde den igen, under oversigten over aftalerne i databasen, under "Import". Til slut, skal testpersonen logge ud.

Brugernavn: navn1
Kodeord: 1234

6.2 Observerede resultater

Mia logger ind, og kommer ind på startsiden. Hun starter med at danne sig et overblik, og trykker herefter på en af dagene i kalenderen. Hun trykker på "Tilføj ny aftale" og begynder at indtaste oplysningerne.
Hun skrev 12 CPR-cifre i første boks, og et navn med henholdsvis fornavn og efternavn. Da der skulle vælges tid, valgte Mia kun at skrive tidspunkt, fordi hun troede, at hun havde valgt en dato, ved at trykke på den markerede dato i kalenderen. Hun skrev en note, og trykkede "Indsend". Her bliver der opgivet en push-meddelelse om, at CPR-nummeret skulle være 10 cifre.
Alle de indtastede oplysninger blev slettet bortset fra tidspunktet, og hun bliver nu nødt til at indtaste alle oplysningerne igen. Hun indtastede et korrekt CPR-nummer og samme navn, og prøvede derfor at indsende igen. Her bliver der opgivet endnu en push-meddelelse, nu om at tidspunktet ikke er korrekt.
Mia bliver nu nødt til at indtaste alle oplysninger igen, og da hun kommer til tid, bliver hun endnu mere forvirret. Hun har svært ved at finde ud af, hvordan hun skal skrive et tidspunkt korrekt, og det gentager sig i alt tre gange. Til sidste finder hun ud af, at man både skal skrive dato og tid ved at klikke på "tætteste tid", hvor der både blev indikeret et tidspunkt og en dato med bogstaver. Her kommer hun dog til at vælge en weekendtid, som bliver indikeret af en push-meddelelse. Til sidst får hun endelig oprettet en tid, men får ikke skrevet en note. Derfor kommer push-meddelsen "Undefined". Hun prøver en sidste gang med alle de rigtige informationer og opretter endelig en konsulation.
Ude på startsiden trykker hun på den pågældende dag, hvor hun har oprettet aftalen. Der står ikke nogen informationer og hun trykker derfor på refresh-knappen. Efter anden gang hun trykkede refresh, kommer aftalen op på displayet til højre.
Hun trykker derefter på "Import"-knappen og kommer ind på siden. Hun finder hurtigt "Dropdown"-knappen og trykker først på de forskellige test før hun ramte "Gruppe 4" og finder listen over alle aftaler i systemet. Her ser hun, at alle de aftaler hun havde oprettet, med fejlmeddelelsen "Undefined", var blevet oprettet i databasen. Hun logger ud.

6.3 Usability testing 2

En ny case bliver fremvist til testpersonen Mia efter, at den mest signifikante fejl er rettet fra forrige test. En kalender er blevet synlig i højre hjørne, så man nemmere burde kunne finde et tidspunkt.
Testpersonen skal nu blot logge ind og oprette en ny aftale med valgfrie informationer.
Brugernavn: navn1
Kodeord: 1234

6.4 Observerede resultater 2

Mia logger ind og kommer ind på startsiden. Hun trykker på "Tilføj ny aftale" og begynder at indtaste oplysningerne. Hun skriver denne gang 10 cifre i CPR-nummer og et navn. Da hun skal vælge tid, trykker Mia på den lille kaldender i højre hjørne og med det samme kommer en kalender op, hvor hun trykker på den dato og tidspunkt for aftalen. Hun valgte først en dag i weekenden og fik en push-meddelelse som forhindrede hende i at vælge dagen. Ingen af oplysningerne blev dog slettet og hun kunne bare vælge en anden dato. Den tætteste dag blev fundet og hun skriver en note til aftalen. Hun trykker "Indsend".

7 Diskussion

I dette afsnit vil der blive diskuteret

- de problematikker der opstod gennem kodnings- og afprøvningsprocessen
- ulige forbedringer, som kunne indføres i fremtiden

7.1 Afprøvning med testperson

Afprøvningen var interessant at udføre, da der var mange elementer, som ikke var taget i betragtning før end, at en udefrakommende person, havde prøvet programmet af.
Testpersonen havde svært ved at oprette en aftale i første test. Her blev det tydeligt, at det var uklart for testpersonen, hvordan man skulle vælge et tidspunkt for aftalen. Desuden var det en besværlig proces, at indsende en aftale, at de fejl programmet fandt, først blev meddelt efter at man havde trykket "Indsend", og derefter skal starte forfra, med at indtaste oplysningerne. Der er altså stadig en mangel på brugervenlighed, som der skulle implementeres i en ny løsning. Det blev løst, ved at tilføje en trykbar brugervenlig kalender, og det blev testet igen, med en ny case. Der sås det, at det var meget nemmere for testpersonen, at finde et tidspunkt for aftalen.

7.2 Knapper på loginsiden

I delaflevering 1 havde vi en 'admin'-knap, som ikke havde en brugbar funktion, på daværende tidspunkt. Den har vi valgt ikke at føre videre til denne kode, da vi ikke følte at den var nødvendig for programmet.
På loginsiden havde vi også en 'reset'-knap, som vi også har valgt ikke at føre med videre. Dette har vi besluttet, da den mere var til forvirring end til gavn. 'Reset'-knappen havde som funktionen at slette teksten i brugernavn- og kodeord-felterne, men der opstod også forvirring her hos folk, der brugte felterne, om den var til at nulstille deres eget kodeord.

7.3 Timeslots

I forbindelse med kodning af vores kalender, startede vi med at lave en statisk kalender. Dette gjorde vi, så vi kunne få en kalender, der ville forsætte forevigt, men det skabte en del udfordringer for os. Dette medførte derfor, at vi ændrede kalenderen om til en dynamisk kalender i stedet. Dette løste vores udfordringer, og resulterede i den kalender, der kan ses i koden nu.

7.4 Fremtidige forbedringspunkter

Til næste aflevering skal sikkerheden forbedres. Der skal indføres kryptering på brugernes informationer, så enhver ikke kan tilgå disse. Loginvalidering skal fungere med brug af token i stedet for 1 eller 0, samt at passwordet til brugeroplysningerne skal gemmes i databasen, med

en 'hashed'- og 'salted'-værdi, i stedet for det oprindelige password.
Derudover, skal importprocessen i næste aflevering udvikles yderligere. Det skal være muligt, at hente andre gruppers data fra deres servere. De skal fremvises ligesom vores egne oprettede aftaler, og lægges ind under de pågældende datoer i kalenderen.

8 Konklusion

Det kan konkluderes, at formålet er opfyldt. Dette er gjort, ved at analysere programmets funktioner grundigt, hvilket har gjort det muligt, at forbedre forståelsen af opgaven, men også at opfylde alle de tilhørende krav. Der er implementeret en backend til programmet, og flere funktioner er blevet tilføjet, ved brug af Javascript. Afprøvningen gav et nyt perspektiv på mulige forbedringer, som kan gøre programmet mere brugervenligt for slutbrugeren.

References

[1] *A Practical Tutorial on Robustness Analysis*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/ (visited on 11/12/2021).