Psyche Assistant

Projektrapport

Produkt- og Procesrapport

Vejleder: Flemming Sørensen, Underviser

Fagekspert: Simone B. S. Larsen, Ergoterapeut

Udvikler: Mads Søndergaard, Elev

TEC Ballerup,

Datatekniker med speciale i programmering

Svendeprøve

Normalsider: 75.829 / 2100 == 36,1 sider

Indholdsfortegnelse

1. Forord:

Dette afsnit introducerer de grundlæggende spørgsmål vedrørende projektets eksistensgrundlag; dets formål, hvem de(n) ansvarlige er, og hvordan rapporten er struktureret.

Denne rapport fungerer som en samlet proces- og produktrapport, og beskriver derfor udviklingen og resultatet af projektet; hvilke overvejelser der er blevet foretaget, udfordringer der har været, hvordan disse er blevet løst, osv., samt en gennemgang af det resulterende produkt, de mål der har været for det, og de mål der er opnået, med mere. Rapporten bygger på en udgave af produktet hvor alle primære (”*need-to-have*”) mål er opnået, samt visse yderligere tilføjelser derudover også er blevet indført – såkaldte sekundære (”*nice-to-have*”) mål.

Produktet kan findes i det offentlige GitHub bibliotek her, både ift. front- og back-end-delene: https://github.com/MadSantiak/H6

1.1. Læsevejledning:

For at gør rapporten mere læsevenlig, er her en kort beskrivelse af den røde tråd læseren kan forvente. Rapporten starter som bekendt med en kort introduktion til projektets eksistensgrundlag. Dette leder over i selve definitionen og beskrivelsen af projektet, således at der er en fundamental forståelse for, hvad projektet forsøger at opnå, hvem der står bag, og hvordan det har været tænkt at målet skulle opnås.
Dette fører over i en beskrivelse af de tekniske valg der har lagt til grunde for udviklingen; hvilke teknologier, værktøjer, osv., der er gjort brug af.
For at bevare den røde tråd, bliver dette efterfulgt af en gennemgang af produktet samt det brugeren ser og interagerer med (”*front-end*”), og det, der faciliterer den brug (”*back-end*”).

Med en forståelse for produktets tilblivelse og det eksistensgrundlag, fortsætter rapporten derfra til en beskrivelse af processen med at udfærdige førstnævnte, og efterleve sidstnævnte. Med andre ord, hvordan produktet er nået til det punkt, det er, da denne rapport blev udfærdiget

Endelig afsluttes rapporten med en kort, overordnet diskussion vedr. projektet som helhed, samt en gennemgang af hvilke muligheder der eksisterer for yderligere udvidelse af produktet, så som mere funktionalitet, forbedret brugervenlighed, eller endda nye mål der relaterer til de oprindelige, og teoretisk kunne gavne af at være samlet i samme produkt. Dette leder naturligt over i en forklaring af de begrænsninger der er i forhold til produktet på nuværende stadie, og deres forklaring.

Endelig vil der være overordnet diskussion af projektet som helhed, efterfulgt af en konklusion, der gerne skulle være i stand til at besvare de problemstillinger, der udgjorde projektets eksistensgrundlag.

Bemærk at visse dele af rapporten, specifikt dele der ikke nødvendigvis er afgørende for rapportens kvalitet, men som læseren vil have gavn af at kunne gennemgå, så som den oprindelige

kravspecifikation, kildekode indeks, brugsscenarier, bruger feedback, daglig logbog, klasse oversigt, osv., vedhæftes som bilag, og henvises til, skulle de blive særligt relevante for et afsnit.

1.2. Formål:

Projektets eksistensgrundlag udspringer fra den nylige COVID-19 epidemi. I særdeleshed fra kølvandet af den. I takt med at flere og flere af de mennesker, der har været inficeret med virussen, nu oplever senfølger, er der sket en stigning af individer med mental træthed (*nogle gange omtalt som "hjernetræthed" i daglig tale*). Typisk er dette en tilstand, der forekommer hos individer, som har været udsat for et voldsomt fysisk, psykisk, eller emotionelt traume, så som biluheld, udifferentieret hjerneskade, voldsommere infektionssygdomme, stress, PTSD, depression, osv. Den resulterende mentale træthed kan ofte være så invaliderende, at patienter der lider under den, ender med at komme på førtidspension, da de ikke har energi til at varetage et arbejde, eller sågar et socialliv.
Desværre er tilstanden en, som det offentlige er langsom til at anerkende – noget som rapporten kommer nærmere ind på i afsnit 2.1. i forbindelse med problemstillingen – og dermed mangler både borgere, og de sundhedsfaglige der behandler dem, værktøjer de kan bruge til at forsøge at administrere og dermed bedre behandle borgerens mentale træthed

Formålet med dette projekt er derfor, at give borgeren der lider af mental træthed et værktøj, der kan assistere dem i deres håndtering af mental træthed, samt gøre dette på en sådan måde, at den er tilgængelig for alle, uanset økonomisk status eller offentlig anerkendelse af tilstanden. Derudover er det et fokus, at det skal være et værktøj, der kan anvendes uden at forværre evt. mental træthed, hvorfor simplicitet og minimalistisk design er et fokus.

Det første skridt var at udarbejde et diagnosticeringsværktøj. Dette har projektets forgænger (der fungerede som et *proof-of-concept*) allerede implementeret. For kort at opsummere dennes funktionalitet; brugere af applikationen kunne tage den såkaldte MFS (*Mental Fatigue Scale*) undersøgelse, som ud fra besvarelsen af 15 spørgsmål, rangerede deres niveau af mental træthed, og gav generelle anbefalinger og symptombeskrivelser, relevante for det scorede niveau. Resultaterne blev derfra gemt lokalt på brugerens enhed via SQL Delight, og tilbød dermed borger (og sundhedspersonale) mulighed for *passivt* at følge udviklingen af borgerens mentale træthed. Forgængerprojektets fundamentale krav var, at funktionaliteten skulle fungere så vidt og bredt som muligt, hvorfor alt data forblev lokalt, spørgsmålene og beskrivelser blev hård-kodet, og der var ingen reel bruger-registrering (for at facilitere offline funktionalitet og undgå privatlivs-bekymringer hos brugere). Alt dette indgår *ikke* som en del af *dette* projekts samlede vurdering, men er fortsat til stede i produktet, omend dele siden er blevet refaktoreret til at passe ind i den nye struktur af kildekoden. Der har dog været et lille fokus på, at holde denne forudeksisterende funktionalitet adskilt, således at dens oprindelige formål med offline funktionalitet, kunne bibeholdes.

Det næste og mere omfattende skridt, og formålet med dette projekt, er at udarbejde et værktøj der giver borgeren, evt. dennes pårørende, samt den sundhedsfaglige, mulighed for aktivt at

administrere – eller hjælpe med at administrere – den mentale træthed, samt opnå en indsigt i borgerens mentale træthed.
Ideen er her, at implementere et værktøj der lader brugere oprette og deltage i grupper, hvori der kan oprettes aktiviteter (*handle ind, støvsuge, fødselsdagsfest, læge besøg, osv.*), som under oprettelse får tildelt en energiomkostning af brugeren. Brugere i gruppen kan derfra handle på de oprettede aktiviteter (*og dermed hjælpe hinanden*), og når de gør dette, kan man efterfølgende udregne brugerens energiforbrug per dato, ud fra udførte aktiviteter, og dermed lave statistik på energiforbruget, så øvrige brugere i gruppen kan følge og forstå andres energiforbrug, og dermed niveau. Ligeledes kan det bruges af den sundhedsfaglige, til at følge dennes borgere, omend som sekundært fokus for projektet.

1.3. Involverede parter:

Simone Larsen er uddannet ergoterapeut, fuldtidsansat på 3. år hos Ballerup kommune, som er en af de meget få kommuner i Danmark, der anerkender mental træthed som en diagnose, der kræver behandling, udover hvad andre diagnoser der så er eller har været relevante for borgeren.
Nogle af Simones mange spidskompetencer er arbejdet med borgere der lider af dysfagi, amputationspatienter, samt netop mental træthed, som hun også selv lider af. Hun nævnes her da udformningen af projektets formål samt produktet, er sket i tæt samarbejde med hende, og hendes viden om det sundhedsfaglige aspekt, har været uvurderlig i forhold til at opnå projektets tiltænkte formål, og hendes løbende feedback og forklaring har været kritisk for at opnå et produkt, der realistisk set kan anvendes af og for borgere, der lider af mental træthed.

Mads Søndergaard er ved at tage en uddannelse som Datatekniker med speciale i Programmering, på 4. år – de første 2 af disse på EUC Syd, og de sidste to på TEC Ballerup. Han er ansat som voksenelev hos VK Data i Tønder, som specialisere sig i udviklingen af moduler til open-source ERP systemet ”Odoo”. Dette betyder også, at hans primære erfaring er med Python og XML, med sekundær erfaring indenfor JavaScript og SQL, som er de sprog der anvendes af Odoo systemet.
Han interesserer sig for full-stack udvikling, dog tendere han mod det funktionelle frem for brugerfladen, da han føler, at det er her, de mest interessante programmeringsudfordringer oftest ligger. I kraft af hans tidligere uddannelse indenfor Kognitiv Semiotik, er han dog også bevidst omkring vigtigheden af en æstetisk indbydende og funktionel UI/UX[1], og gør brug af denne teoretisk viden i forbindelse med udformningen af samme, så vidt mulig.
Mads’ mål med uddannelsen til datateknikker med speciale i programmering er, at omdanne en livslang interesse indenfor IT, og specifikt software, til en mulig levevej, og med tiden forhåbentlig drage nytte af denne erfaring, og forene den med hans interesse i – og eksisterende uddannelse indenfor – kognitiv videnskab og semiotik; specifikt bevidsthedsstudier.

1 User Interface / User Experience, eller brugerflade/brugeroplevelse; hvad brugeren ser og oplever når de anvender et stykke software.

2. Introduktion

Følgende er en beskrivelse af udviklingen af Psyche Assistant; navnet på projektet så vel som produktet. Bemærk at den den primære bruger til tider omtales ”borger” eller ”patient”, imens sekundære (andre) brugere kan være ”pårørende”, ”sundhedsfaglige”; denne variation i betegnelse skyldes den konteksten; er denne set fra et sundhedsmæssigt synspunkt, er det en ”patient”, er det en sundhedsfaglig, er det en ”borger”, og er det som anvender af projektets produkt, er det ”bruger”.

Derudover henvises der til Bilag 1: Kravspecifikation, for en beskrivelse af samme, samt Bilag 6: Projektplan og Log, for en mere detaljeret gennemgang af den daglige udviklingsproces, både planlagte og faktiske.

2.1. Problemstilling:

Til trods for at mental træthed er relativ udbredt, og til trods for at den nu er mere synlig end før, som følgevirkning efter COVID-19, er det mest en udfordring som primær sundhedspersonale (ergoterapeuter, sygeplejere, sundhedsassistenter) støder på. Det er samtidig også en tilstand der oftest overses og misforstås af både patient og sundhedssystemet, da en generel mangel på energi og overskud, nemt misfortolkes som en sekundær effekt, der udspringer af en anden diagnose (*hvis man er stresset eller deprimeret, er man jo naturligt også mere tilbøjelig til at mangle energi og overskud*); men dette skyldes en igangværende spidsbelastning, hvorimod mental træthed referere til en general nedsat tolerance overfor belastning.
Fokus fra sundhedsvæsenet ligger dermed på at behandle den primære diagnose, og forståelsen fra patienterne er oftest, at når den primære diagnose løser sig, følger trætheden med.
Dette åbner for et logisk hul, hvor patientens sundhedsforløb afsluttes ved løsning af den primære diagnose, og systemet bliver efterfølgende ikke delagtiggjort i patientens eventuelle fortsatte problemer med træthed – som ville indikere mental træthed og ikke træthed som følge af spidsbelastning – og dermed evne til at varetage et normalt social- og arbejdsliv, afhængig af sværhedsgraden. Ligeledes står patienten (*og dermed også dennes pårørende*) med en manglende forståelse for den mentale træthed, som patienten fortsat føler, som følge af nedsat tolerance overfor belastning, og dermed let bruger mere energi end de har. Dette fører til frustration over, at de ikke magter de ting de plejede, til trods for at sundhedssystemet har afsluttet deres forløb, og de dermed er (*eller burde være, i systemets øjne*) raskmeldte.

Resultatet er, at sundhedssystemet sjældent har behandlingstilbud til denne specifikke problemstilling, og dermed gives der ikke den fornødne viden eller de nødvendige værktøjer til de sundhedsfaglige, der behandler en borger med mental træthed. Derudover mangler borgeren og dennes pårørende ligeledes forståelse og/eller værktøj, der kan hjælpe dem med at håndtere tilstanden.

Derudover er mental træthed en tilstand, der får gavn af - hvis ikke ligefrem kræver - at der sker en vedholdende, regelmæssig behandling. Det er f.eks. vigtigt, at borgeren så vidt muligt er

opmærksom på deres energiniveau, og proaktivt tager skridt til vedligeholde den, inden det er for sent, og de blive overrumplet af træthed. Denne vedholdende indsat kræver i sig selv en hvis mængde overskud at systematisere på egen hånd. Alternativt kræver det relativ regelmæssig indblanding fra en sundhedsfaglig person, hvilket dog hurtigt ville blive en stor omkostning for sundhedsvæsenet.

Derfor bliver tanken bag projektet her, at lave et stykke værktøj, der kan hjælpe borgere og sundhedsfaglige med at systematisere denne årvågenhed og proaktive tilgang, og forsøge at gøre det på en sådan måde, at risikoen for at forværre brugerens mentale træthed minimeres – for eksempel via fokus på UI/UX præsentationen og simplicitet i brugsscenarier – samt opnå den ønskede funktionalitet, på en sådan måde, at ingen GDPR regler brydes, men så den samtidigt er meningsfuld for bruger(ne).

3. Projektbeskrivelse

Psyche Assistant er en mobil applikation der udvikles via Kotlin Multiplatform (KMP), nogle gange kaldet Compose Multiplatform, hvilket er JetBrains svar på et framework, der kan anvendes til at udvikle både logik og brugerflade, til flere forskellige platforme.[2] Bemærk at i øjeblikket eksisterer applikationen kun til Android platforme, grundet omkostninger og udfordringer der ville have været ved at også udvikle den til iOS. Dog er der forsøgt udviklet efter multiplatform paradigmet så vidt muligt, hvilket vil sige, at så meget af logikken og brugerfladen som muligt, er holdt i platform-agnostisk kode.

3.1. Flowchart

For at få en bedre forståelse for mulige brugsscenarier og anvendelse, henvises der til Bilag 5: Flowchart og Brugervejledning.

4. Teknologier

Følgende er en gennemgang af de teknologier og paradigmer, der er anvendt i udviklingen af produktet til dette projekt.

For en komplet gennemgang af kildekoden, henvises der til Bilag 2: Kildekode indeks.

4.1. Arkitektur

For at give et bedre overblik over produktet, er der her indledningsvis en kort gennemgang af den arkitektur der er anvendt i udformningen af både front- og back-end-delen.

2 https://www.jetbrains.com/kotlin-multiplatform/

4.1.1. Front-end Arkitektur

Overordnet set læner front-end delen af projektet sig op ad en såkaldt MVC (Model-View-Controller) struktur, eftersom den har visninger (i form af ”Composable”), der gør brug af modeller til at generere og vise data i samme, og controller klasser til at sende og modtage data fra back-end-delen.

Der er også elementer af ”Intent” og ”States” i front-end-delen, da brugernes handlinger udløser ændringer i states, så som variabler, som så opdatere visningen til at afspejle ændringerne.

Derudover er der en kombination af programmeringsparadigmer. Der eksisterer både objekt-orienterede principper (modeller, klasser der indeholder funktionalitet, modulær opdeling af samme, osv.), og funktionelle principper, idet selve indholdet i visninger opdateres som en vedvarende proces af handling (intents) og deres indflydelse på tilstanden (state), og denne ændring udløser dernæst en opdatering af visningen, og derudover ses dette funktionelle princip også forekomsten af videresendelse af funktioner fra en visning til den næste.

4.1.2. Back-end Arkitektur

Back-end-delen er en mere traditionel lagdelt arkitektur, med Controller klasser, der eksponerer API end-points, som kan kaldes fra og returnere data til front-end. Dette sker ved at bruge Service-klasser der står for verifikation og behandling af dataintegritet, i leddet mellem Controller og Repository, hvor sidstnævnte står for den faktiske ind- og udlæsning af data fra databasen.

4.2. GitHub versions-kontrol

GitHub er blevet brugt til at versions kontrollere produktet. Grundet antallet af udviklere involveret i projektet, er det dog minialt i hvor stor omfang gren-baseret kontrol har været anvendt under udviklingen. Generelt set er der dog blevet anvendt en ”dev” gren til løbende udvikling, som så blev hentet over i ”main” grenen når en fundamental milepæl blev nået, og funktionel tests ikke fandt nogle graverende fejl.
Der er derudover sket ekstensiv refaktorering af projektets udgangspunkt (vis-a-vis Mental Fatigue Scale undersøgelser), da strukturen under udviklingen blev markant anderledes, for at et teknisk overblik kunne bibeholdes.

Der er blevet anvendt semantisk versionering[3] til at spore produktets stadie samt ændringer, samt en changelog for både front- og back-end, hvor ændringer løbende er blevet skrevet ind ved hver versionsændring.

4.3. Kotlin Multiplatform

Som nævnt indledningsvis er det primære framework for front-end-delen Kotlin Multiplatform (KMP). Et open-source, community-drevet projekt, ledet af JetBrains, som også er udviklerne bag

3 F.eks. ”0.1.0”, for at angive kritiske, større, eller mindre ændringer, respektivt.

selve Kotlin programmeringssproget[4] – som dermed også er det primær anvendte sprog i produktets front-end-del, efterfulgt af Java (til back-end-delen).

KMP er specifikt udviklet til at facilitere udvikling på flere platforme, ved at gøre det muligt, at programmere ved hjælp af platform agnostisk kode, men samtidig at gøre brug af platform-specifik (native) kode, hvor og hvornår det er nødvendigt eller at foretrække.[5] Dermed tilbyder det stor fleksibilitet, i forhold til om man vil udvikle til flere platforme samtidig, muligvis med tab af en vis mængde ydeevne, eller specifikt til én enkelt platform – eller ideelt set, en blanding af begge. Bemærk, at hvis man anvender agnostisk kode til både brugerflade og logik, bruges termen ”Compose Multiplatform” oftest.[6]

Det var først for relativt nyligt – 1. November, 2023 – at KMP kom ud af beta og opnåede en såkaldt ”stable release”, hvilket også afspejler sig i nogle af de udfordringer man kan støde på, når man for eksempel skal finde relevant, opdateret materiale, eller finde eksempler der passer til den nuværende udgave.

KMP blev valgt til projektet grundet dens understøttelse af diverse platformes specifikke native kode, hvorved det er muligt at undgå reduktion i ydeevne, kontra, for eksempel, Flutter, der anvender sin egen renderings motor til at generere brugerfladen, i stedet for platformens egen. Denne ”blanding” af agnostisk og specifik kode i KMP via brugen af expect/actual funktions nøgleord, som gøre det muligt, at definere en agnostisk funktion (”expect”), der *forventer* den platform-specifikke funktions (”actual”) *faktiske* implementering. Essentielt en slags nedarvning, hvor ”expect” udgaven overskrives af ”actual” udgaven, afhængig af den platform softwaren afvikles på.

På samme måde som Flutter, er det også muligt at lave en platform agnostisk brugerflade, så både logik og brugerflade i mange tilfælde kun skal defineres ét (delt) sted, hvorefter det kompileres til de respektive platformes native kode.

En mindre teknisk årsag til valget af Kotlin Multiplatform, er at det er det multiplatform framework, som udvikleren har mest erfaring med, og derudover også, at det som nævnt også var et ønske, at projektet ville have mulighed for at nå ud til en bredere skarre af brugere.
Dog er den erfaring udvikleren har, kun i det omfang, at det var samme framework der blev brugt til at udvikle projektets proof-of-concept.

Endelig er det derudover også udviklerens personlig mål, at udvikle sine kompetencer løbende, og dette projekt anses ikke kun som en mulighed for at udvikle hvad der kan anses for et værdifuldt redskab til slutbrugerne, eller som en mulighed for at gennemføre en svendeprøve, men også som en mulighed for at bygge videre på eksisterende og nye kompetencer, som jo må siges at være essentielt for faget - både før, under, og efter uddannelsen.[7,8]

4 https://kotlinlang.org/docs/faq.html
5 https://kotlinlang.org/docs/multiplatform.html
6 https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-multiplatform-getting-started.html
7 https://innovationmanagement.se/2018/05/25/how-to-keep-up-with-constantly-changing-technology/
8 https://redis.io/blog/future-proof-your-dev-career/

4.4. Spring Boot

Til oprettelse af back-end-delen er der anvendt Spring Boot, hvilket er et framework der reducere boilerplate kode i forbindelse med udviklingen af Java baserede back-end-dele, så som RESTful API, oprettelse af SQL databaser, osv. Dette opnås via en række annoteringer, der som i eksemplet nedenfor, fortæller hvordan et SQL skema/tabel skal se ud, som så bliver oprettet under afvikling:

```java
@Data  MadSantiak +1 *
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")
public class User {
    @Id  no usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(nullable = false, unique = true)  no usages
    private String email;
    private String password;  no usages
    private int energyExpenditure = 0;  no usages

    /**
     * To avoid recursion when calling from front-end, we only use the ID as reference when
     */
    @ManyToOne  no usages
    @JoinColumn(name = "group_id")
    @JsonIdentityReference(alwaysAsId = true)
    private Group group = null;
}
```

Selve opsætningen af database forbindelsen, den specifik SQL dialekt, og opdateringen af skemaer/tabeller fra Spring Boot, er defineret i back-end-delens application.properties fil, hvor den her er sat til at opdatere en H2 database ved ændringer i annoteringerne, om muligt uden at droppe eksisterende data. Bemærk der her er tale om en udviklingsdatabase brugt under udviklingsforløbet – og som senere er migrere væk fra, derfor udkommenteringen – hvorfor legitimationsoplysninger blot er tænkt som pladsholdere:

```
# H2 Configuration:
# spring.datasource.url=jdbc:h2:file:~/data/psycheassistantdb
# spring.datasource.driver-class-name=org.h2.Driver
# spring.datasource.username=TestName
# spring.datasource.password=TestPass
# hibernate.dialect=org.hibernate.dialect.H2Dialect
# spring.h2.console.enabled=true
# spring.h2.console.path=/console/h2
# spring.h2.console.settings.web-allow-others=true
```

Som nævnt simplificerer Spring Boot også oprettelsen af RESTful API, også ved hjælp af annoteringer på klassen, felter, funktioner, osv.:

```
@RestController  no usages  MadSantiak
@RequestMapping("/api/activities")
public class ActivityController {

    @Autowired  8 usages
    private ActivityService activityService;
    @Autowired  no usages
    private JwtUtil jwtUtil;
    @Autowired  3 usages
    private AuthHelper authHelper;


    @GetMapping()  no usages  MadSantiak
    public List<Activity> getAllActivities() { return activityService.findAll(); }

    @GetMapping("/{id}")  no usages  MadSantiak
    public Activity read(@PathVariable int id) { return activityService.findById(id); }
```

Grunden til at valget faldt på Spring Boot, er den relative simple oprettelse og vedligeholdelse, altimens det fortsat er et fleksibelt værktøj, der ikke kræver større refaktorering, for eksempel i forbindelse med at man ville flytte databasen fra en type til en anden, som det har været tilfældet for dette projekt.

4.5. Databaser

Til applikationen har der være anvendt forskellige typer af databaser, for at opfylde forskellige mål. I dette afsnit kigges der på de database typer, der er anvendt i produktet, både under og efter udvikling.
Som nævnt tidligere bygger produktet videre på et proof-of-concept projekt, som anvendte JSON til at lagre hård-kodede værdier (undersøgelses spørgsmål), samt en lokal SQL Delight database, til at lagre data direkte på brugerens enhed, for at efterkomme ønsket om at applikationen kunne bruges så problemfrit som muligt – så f.eks. uden at være online. Disse funktionaliteter eksisterer derfor fortsat i produktets nuværende udformning, men udgør teknisk set ikke en del af dette projekts produkt; de nævnes her for at undgå forvirring om hvad er relevant for produktet.

4.5.1. Eksterne databaser

Der er blevet anvendt forskellige database typer til det nuværende projekts produkt, alt efter hensigten med arbejdet. Under udviklingsarbejdet er der blevet anvendt en H2 database, der er ideel til netop hurtig og regelmæssig ændring og udvikling; den er dog ikke ideel som en reel produktions-database. Derfor er back-end-delen, i forbindelse med afslutningen af udviklings arbejdet, blevet flyttet over på en mere robust PostgreSQL server, der teoretisk ville være bedre udstyret til at behandle større mængde trafik, og dermed afspejle en teoretisk produktions-tilstand.

4.5.1.1 H2 SQL

H2 er en letvægts java baseret SQL database, der kan konfigureres enten som en in-memory eller fil-baseret database, og afviklet i enten embedded eller server tilstand. Den kræver minimal opsætning, og er derfor ideel til udviklingsarbejde.

Forskellen på konfigurationerne er, om data er bestandig mellem afviklinger af serveren, hvor en hukommelse-baseret opsætning vil resulterer i, at når databasen afsluttes, går lagret data tabt, hvorimod en filbaseret udgave vil vedligeholde data mellem afviklinger.

Embedded- og server-tilstandene henviser til databasens tilgængelighed. Specifikt betyder førstnævnte, at databasen kun er tilgængelig så længe applikationen der instantierer og afvikler den kører, hvorimod den i sidstnævnte ville være tilgængelig for flere andre instanser af applikationen – bemærk at der her menes back-end-applikationen, ikke front-end.

Her er den blevet brugt som en fil-baseret embedded database, for at opnå bestandig lagring, også mellem afvikling af back-end-delens applikationen. Uanset opsætning er den dog ikke ideel til større mængder trafik eller mere kompleks dataudveksling, så er derfor ikke lige så godt egnet som produktions database.

4.5.1.2. PostgreSQL

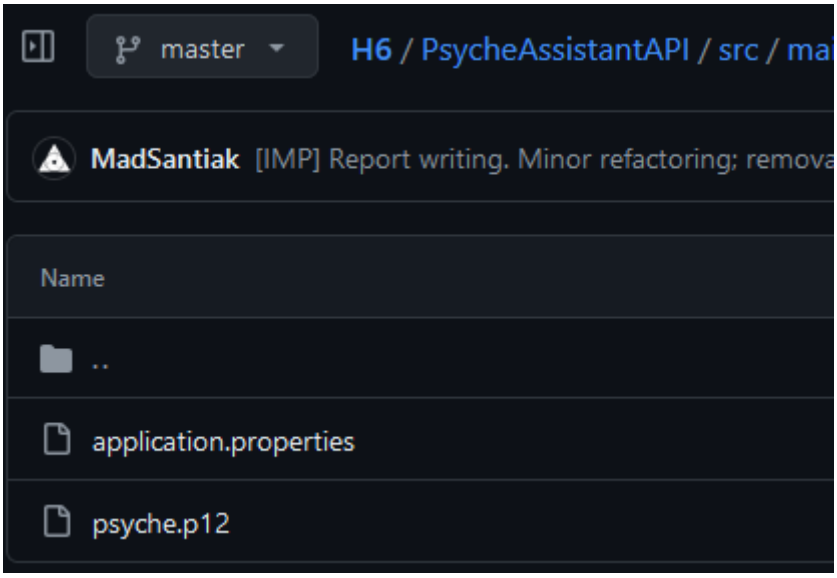
PostgreSQL er en mere robust relations-baseret database, der kan tilbyde mere avancerede funktioner, samt bedre understøttelse af større mængder trafik og kompleks data, skulle det være nødvendigt. Det kan argumenteres, at det strengt taget ikke har være nødvendigt at migrere

databasen for dette projekt, men takket være Spring Boot, har dette være nemt og hurtigt at gennemføre, og blot et spørgsmål om at ændre i back-end-delens konfigurationsdele, som vist nedenfor. Det har selvfølgelig krævet et PostgreSQL serveren blev opsat forinden, men efterfølgende har Spring Boot stået for oprettelsen af de nødvendige tabeller og skemaer. Selv om det ikke var en nødvendighed, var det ønsket at afspejler et produktionsmiljø så vidt muligt, i den endelige udgave af projektet; derfor migreringen til trods.

application.properties fil:

```
# Import credentials from external file (exclude from source control!)
spring.config.import=optional:classpath:credentials.properties

# PostgreSQL Configuration:
spring.datasource.url=jdbc:postgresql://localhost:5432/PsycheAssistantDB
spring.datasource.username=${sql_username}
spring.datasource.password=${sql_password}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode=always
spring.jpa.hibernate.ddl-auto=update

# H2 Configuration:
# spring.datasource.url=jdbc:h2:file:~/data/psycheassistantdb
# spring.datasource.driver-class-name=org.h2.Driver
# spring.datasource.username=TestName
# spring.datasource.password=TestPass
# hibernate.dialect=org.hibernate.dialect.H2Dialect
# spring.h2.console.enabled=true
# spring.h2.console.path=/console/h2
# spring.h2.console.settings.web-allow-others=true
```

Bemærk at legitimationsoplysningerne for PostgreSQL konfigurationen for sikkerheds skyld er flyttet ud i en separat fil (*credentials.properties*), som ikke tages med når koden skubbes op på GitHub, men som importeres i application.properties.



Og derudover kræver migreringen også en lille ændring i back-end-delens afhængigheder:

build.gradle.kts:

```
dependencies {
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-web")
    compileOnly("org.projectlombok:lombok")
    developmentOnly("org.springframework.boot:spring-boot-devtools")
    // runtimeOnly("com.h2database:h2") // H2 Integration
    runtimeOnly("org.postgresql:postgresql") // PostgreSQL Integration
    annotationProcessor("org.projectlombok:lombok")
```

4.5.3. MMKV

Udover back-end databasen, er der i front-end delen også anvendt lokal lagring til at persistere data mellem kørsler af applikationen der. Specifikt er der anvendt Memory-Mapped Key-Value (MMKV) lagring, til at lagre en JSON Web Token – både access- og refresh-token – forkortet JWT, som bruges til at persistere brugerens login-tilstand mellem brug af applikationen. Dette gøres ved at oprette et expect/actual KMP mønster, hvor der på Android kaldes den specifikke implementation af MMKV, som gør brug af biblioteket til at lagre, hente, eller slette førnævnte JWT:

Agnostisk implementering:

```kotlin
/**
 * Auth storage
 * Expect object for platform specific implementation for storing tokens securely on the ph
 * See PsycheAssistant/composeApp/src/androidMain/kotlin/Storage/AuthStorage.android.kt
 * for Actual implementation.
 * @constructor Create empty Auth storage
 */
expect object AuthStorage {  MadSantiak
    fun saveAuthToken(token: String)  MadSantiak
    fun getAuthToken(): String?  MadSantiak
    fun removeAuthToken()  MadSantiak
    fun saveRefreshToken(token: String)  MadSantiak
    fun getRefreshToken(): String?  MadSantiak
    fun removeRefreshToken()  MadSantiak
}
```

Android implementering:

```kotlin
actual fun saveAuthToken(token: String) {  MadSantiak
    MMKV.defaultMMKV().encode(AUTH_TOKEN_KEY, token)
}

actual fun getAuthToken(): String? {  MadSantiak
    return MMKV.defaultMMKV().decodeString(AUTH_TOKEN_KEY)
}

actual fun removeAuthToken() {  MadSantiak
    MMKV.defaultMMKV().remove(AUTH_TOKEN_KEY)
}
```

Dermed kan der fra den platform-agnostiske kode gøre brug af JWT, som sendes via API kald, og i back-end-delen bruges til at verificere brugerens anmodninger, så brugeren ikke behøver logge ind på ny hver gang de starter applikationen, såfremt deres JWT tokens ikke begge er udløbet:

Front-end kald:

```
override suspend fun getUserDetails(authToken: String): User? {  MadSantiak
    val url = ServiceBuilder.url( path: "api/users/me")
    var response: HttpResponse = client.get(url) {
        header(HttpHeaders.Authorization, "Bearer $authToken")
    }

    if (response.status == HttpStatusCode.Unauthorized) {
        // Token might be expired, try to refresh it
        val newToken = refreshToken()
        if (newToken != null) {
            AuthStorage.saveAuthToken(newToken)
            response = client.get(url) {
                header(HttpHeaders.Authorization, "Bearer $newToken")
            }
        } else {
            // Refresh token failed or is not available
            throw Exception("Token refresh failed, user needs to log in again")
        }
    }
```

Back-end API:

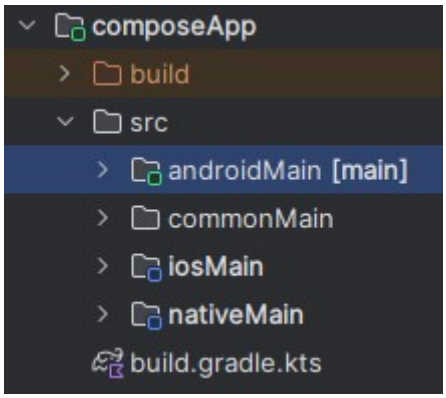
```
@GetMapping("/me")  no usages  MadSantiak
public ResponseEntity<User> getCurrentUser(@RequestHeader("Authorization") String authToken) {
    try {
        String token = authToken.replace( target: "Bearer ",  replacement: "");
        String email = jwtUtil.extractEmail(token);

        User user = userService.findByEmail(email);

        if (user != null) {
            return ResponseEntity.ok(user);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
        }
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null);
    }
}
```

I back-end-delen anvendes JWT Util, som gør brug af JWT biblioteket, til at oprette, finde, slette, eller forny tokens, og derudover anvendes også en overskrivning af JWT Filter, til at verificere at den pågældende token, koblet til brugerens email, ikke er udløbet.

4.6. Struktur

Denne sektion vil kort introducere den fil-struktur der er anvendt i projektet, både for at give et bedre overblik over selve applikationen, men også i lyset af, at dette per definition er unikt for KMP projekter.

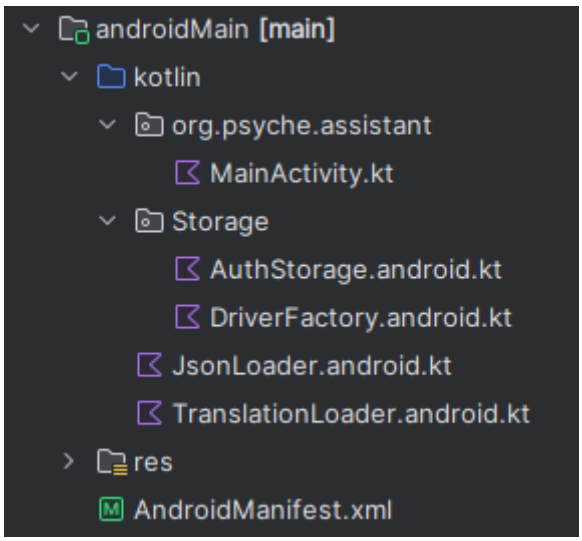
Specifikt kræver KMP en særlig struktur, hvor der gøres brug af specifikke mapper (commonMain, androidMain, iosMain, osv.), til at lade KMP frameworket vide, hvilken kode er til hvilken platform, eller om den er platform agnostisk (common).



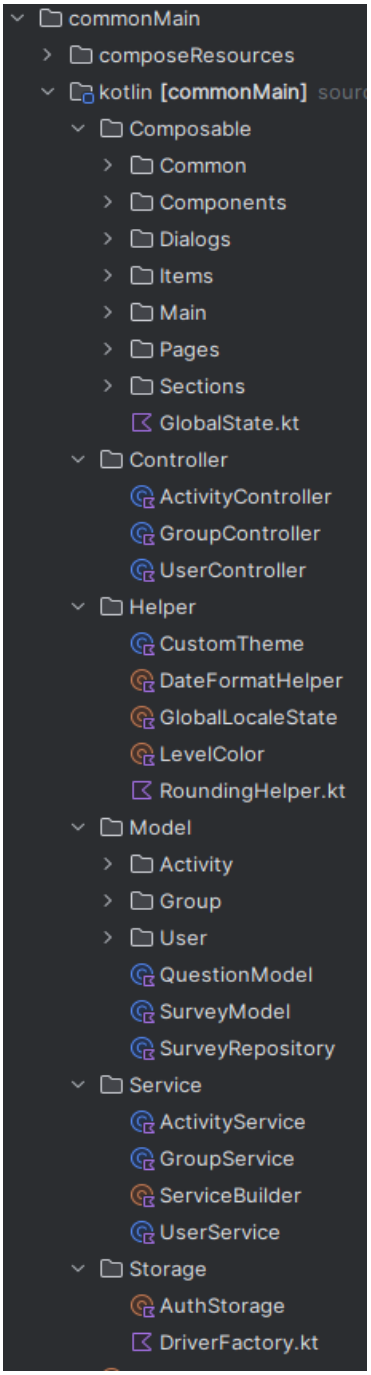
Dette kan dog give visse problemer, så som hvis en klasse skal tilgå en fil med JSON værdier, som skal være tilgængelig på flere platforme, og derfor være centralt placeret i den platform-agnostiske del, ved at den findes i commonMain mappens ”resources” mappe.
Her kan det i værste fald blive nødvendigt, at placere klassens expect/actual i rod-mapperne for hhv. den platform agnostiske- og specifikke del af strukturen, og specifikt instruere projektets android del i, hvor filerne findes, da programmet ellers kan have besvær med at finde frem til ressource filen.
Dette har dog ikke være et problem for det nuværende projekt, muligvis grundet en senere opdatering, men viste sig at være en forhindring på proof-of-concept projektet; af samme grund kan man derfor se ”JsonLoader” og ”TranslationLoader” klasserne i rod mapperne, udenfor en mere semantisk forståelig fil-struktur, så som i en ”Helper” undermappe. Med andre ord har det været et nødvendigt onde at placere klasserne her, og ikke grundet et ønske eller tankeløshed. Bemærk dog også, at de nævnte filer ikke bør medtages i dette projekts vurdering – derfor de eksplicit nævnes her, grundet deres synlighed.

4.6.1. Android struktur

Som det ses her, er det en meget lille del af den samlede kodebase, der reelt ligger i den Android-specifikke sektion. Reelt set er der kun indgangspunktet (MainActivity), der instantierer dette projekts MMKV lagring, samt proof-of-concept projektets SQL Delight integration, via hhv. AuthStorage og DriverFactory klasserne, inden den kalder tilbage til den agnostiske kodes primære indgangspunkt, som derfra afvikler logik og UI/UX agnostisk.



4.6.2. Platform-agnostisk struktur



Som det ses her, er langt størstedelen af kodebasen i den platform-agnostiske del. Af hensyn til læse venligheden er der undladt at vise indholdet i hver eneste mappe – og af samme årsag vil gennemgangen også kun fokusere på formålet med strukturen og mappernes indhold, og ikke selve indholdet af mapperne. Her kan der i stedet henvises til hhv. Bilag 2: Kildekode indeks og Bilag 7: Klasse oversigt.

composeResources indeholder de ressourcer de anvendes i applikationen, og som der ikke ønskes hård-kodet, således at der f.eks. kan angives tekst-strenge, der følger enhedens sprogindstilling, hvis strengen er tilgængelige på det specifikke sprog. Dette angives ved at oprette de passender oversættelser som en række XML værdier, og placere dem i en ”*value-[sprogkode]*” undermappe. Hvis denne ikke findes for sproget, eller den specifikke streng ikke er oversat der, faldes der tilbage til de strenge der er definerert i den grundlæggende *”value”* undermappe.

Composable mappen indeholder de visnings-komponenter der bruges til at vise brugerfladen og interagere med brugeren.
Den centrale del her er undermappen *Main*, der indeholder startpunktet for resten af applikationens brugerflade, og som står for at oprette navigation, tema, med mere. *Pages* er de reelle ”sider” man kan navigere imellem, imens *Sections* er de del-komponenter som sider kan bestå af. *Items* indeholder komponenter der bruges til at vise data (*objekter*) på en hensigtsmæssig måde, så som oprette en visning af en liste af bruger objekter, som igen burger en visning for det enkelte bruger objekt. *Components* er de mere grafiske komponenter anvendt i applikationen, så som visualisering af data i et tegnet søjlediagram, eller interaktiv interaktion med brugeren, så som dato valg (*i stedet for brugeren manuelt skal skrive datoen, hvorved formatet også skulle valideres hver gang*).

Controller mappen indeholder bindeleddet mellem brugerfladen og logikken. Specifikt kalder handlinger i bruger-fladen ned til klasser i controller mappen, som så igen kalder klasser i *Service* mappen, som så står for den endelige interaktion mellem front-end- og back-end-delen. På denne måde ville det være relativt nemt at ændre i kommunikationen mellem front-end og back-end, uden at skulle rette brugerfladen til. Det er dermed mere et abstraktionsniveau, der gør vedligehold af koblingen mellem front- og back-end mere ligetil, kontra en direkte forbindelse mellem handlinger i brugerfladen og klasserne i Service mappen.

Helper mappen er primært mindre funktioner der har være brugt i proof-of-concept projektet, som til dels stadig bruges her. *CustomTheme* er for eksempel opsætningen af brugerfladens farvetema (*lettere omkonfigureret*), imens *LevelColor* og *RoundingHelper* bruges til at hvv. farvegive tekst ud fra en given double værdi, samt afrunde en double til et givent antal decimaler – sidstnævnte da denne implementation ellers ikke var tilgængelig i den platform-agnostiske biblioteker.

Model mappen indeholder definitionerne på de data klasser (objekter) der skal bruges i applikationen, samt deres interfaces (*Repository*), der udgør en deklaration på de funktionaliteter, der skal være tilgængelige i deres respektive kommunikation med back-end-delen (via *Service*), ved at sidstnævnte implementere førstnævnte.

Service mappen er som sagt de klasser, der står for kommunikationen med back-end delen, og udvekslingen af data mellem de to. ServiceBuilder filen her udgør en singelton, der bruges til at centralisere og simplificere forbindelsen til back-end delen. Her findes der også en ConfigProvider expect/actual implementering, for at gøre det muligt, at løsne visse data (IP, nøgler) fra selve koden, og lagre disse i filer, som den platform-specifikke kode kan anvende på enheden.

Storage mappen indeholder de dele vedrørende lokal lagring på hver enkelt platform, som applikationen overordnet set *expect'er* har en *actual* implementering på de respektive platforme den skal virke på.

Øvrige dele udenfor den nævnte struktur, så som QuesitonModel, SurveyModel, og SurveyRepository, er dele det blev udviklet ifm. proof-of-concept projektet, og som stadig anvendes i applikationens "nye" udgave, men som ikke er blevet udviklet på siden, udover evt. refaktorering af f.eks. mappe strukturen.

4.7. Jetpack Compose

Produktet er lavet til at være så platform-agnostisk som muligt. Udover at opnå dette i henhold til den logiske kode, opnås det også ved brugen af Jetpack Compose i front-end-delen, i så stort omfang som muligt.

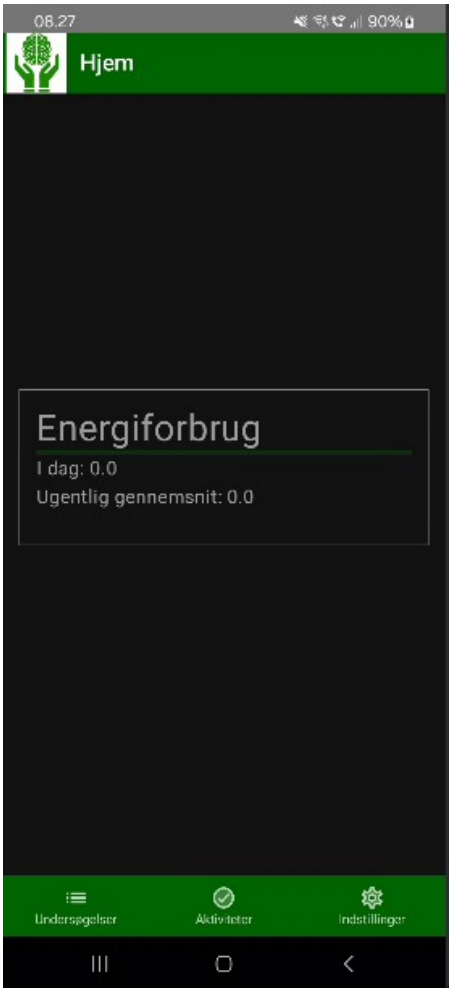
Jetpack Compose gør brug af @Composable funktions-dekoratøren, der gør funktionen i stand til at definere UI komponenter, som så kan håndtere stadier, og reagerer på ændringer på de stadier. Dette opnås ved at gøre brug af funktionaliteter så som remember() og/eller mutableStateOf() funktionerne, som definere variabler som stadier, som der skal observeres og reageres på. Derudover kan man også anvende LaunchedEffect, der aktivt holder øje med- og opdatere en Composable funktion, ud fra ændringer af variabler der ligger udenfor samme, som tager parametre i form af de variabler som, hvis en ændring sker, skal udløse en afvikling af den definerede funktionalitet i LaunchedEffect. Da visningerne dermed er funktioner, betyder det samtidigt, at man kan definere en funktion i én visning, og sende den med som argument til en anden visning, som så igen returnere en værdi til den forrige visning; med andre ord er UI/UX delen opbygget med et funktionelt paradigme – omend der gøres brug af objekt-orienterede bestanddele i selve visningerne.

Disse Composable funktioner blive derfra kompileret til platform native kode, som dermed gør det muligt, at udvikle én central definition på UI/UX, som så kan afvikles på diverse platforme, uden nævneværdigt mere kompleks central kode. Man skal, selvfølgelig, være opmærksom på, at man ikke anvender biblioteker i den agnostiske kode, der kun er tilgængelig på specifikke platforme, hvis man ønsker at bibeholde den agnostiske tilstand, dog er KMP relativ god til at holde øje med dette, da det oftest også kræver en specifik import for at bruge et givent værktøj i en given platforms struktur sektion.

5. Psyche Assistant App

Dette afsnit vil handle om produktet generelt. Specifikt de overvejelser der er foretaget, både i forhold til hvordan applikationen skulle præsenteres for brugeren, hvordan dette design blev opnået, samt den funktionelle baggrund for dette, så som modeller, back-end, og sikkerhedsovervejelser.

5.1. Brugerflade- og oplevelse



Med brugere der lider af mental træthed som den primære målgruppe for applikationen, er en af projektets vigtigste mål, at opnå den ønskede funktionalitet, på en sådan måde, at både brugerflade- og oplevelse ikke risikere at forværre den mentale træthed. Forværrende omstændigheder kan være risikofaktorer så som socialisering, fysisk aktivitet, emotionel turbulens, og vigtigst i denne sammenhæng, audiovisuelle stimuli og større mængder information.

Det har derfor været et fokus, at udvikle applikationen på en sådan måde, at den (1) var simpel i sin præsentation, og dermed undgik overstimulation af brugeren, og (2) at den om muligt blev præsenteret på en sådan måde, at det kunne virke beroligende på brugeren.

Af hensyn til sidstnævnte er applikationen blevet udviklet til at bruge et afdæmpet, mørkt tema, for at eliminere eventuel forværring som følge af overfølsomhed overfor lys. Dette opnås ved at definere en overskrivelse af standard temaet, og i indgangspunktet for applikationen – PsycheAssistantApp.kt – anvende denne i oprettelsen af det staffeli, der definere resten af brugerfladen.
For at hjælpe med at bibeholde brugerens opmærksomhed, er der anvendt en grøn farve som det primære farvevalg, som er valgt for sin overordnede emotionelle positive men neutrale association.[9,10]

9 Oh, J., Lee, H., & Park, H. (2021). Effects on Heart Rate Variability of Stress Level Responses to the Properties of Indoor Environmental Colors: A Preliminary Study. International journal of environmental research and public health, 18(17), 9136. https://doi.org/10.3390/ijerph18179136

10 Güneş, Elif & Olguntürk, Nilgün. (2019). Color-emotion associations in interiors. Color Research & Application. 45. 10.1002/col.22443.

5.1.1. Design

For at opnå et ensformigt design, defineres farverne i deres egen separate klasse:

```
class CustomTheme {  MadSantiak
    companion object {  MadSantiak
        fun psycheColors(  MadSantiak
            primary: Color = Color( color: 0xFF006400),
            onPrimary: Color = Color.White,
            secondary: Color = Color( color: 0xFF444444),
            onSecondary: Color = Color( color: 0xFF00FF00),
            background: Color = Color( color: 0xFF121212),
            onBackground: Color = Color( color: 0xFFB2B2B2),
            surface: Color = Color( color: 0xFF1C1C1C),
            onSurface: Color = Color( color: 0xFF00FF00),
            error: Color = Color( color: 0xFFB00020),
            onError: Color = Color.White,
        ): Colors {
            return lightColors(
                primary = primary,
                onPrimary = onPrimary,
                secondary = secondary,
                onSecondary = onSecondary,
                background = background,
                onBackground = onBackground,
                surface = surface,
                onSurface = onSurface,
                error = error,
                onError = onError,
            )
        }
    }
}
```

Som derfra så sendes med til applikationens indgangspunkt, hvor det definere MaterialTheme farverne:

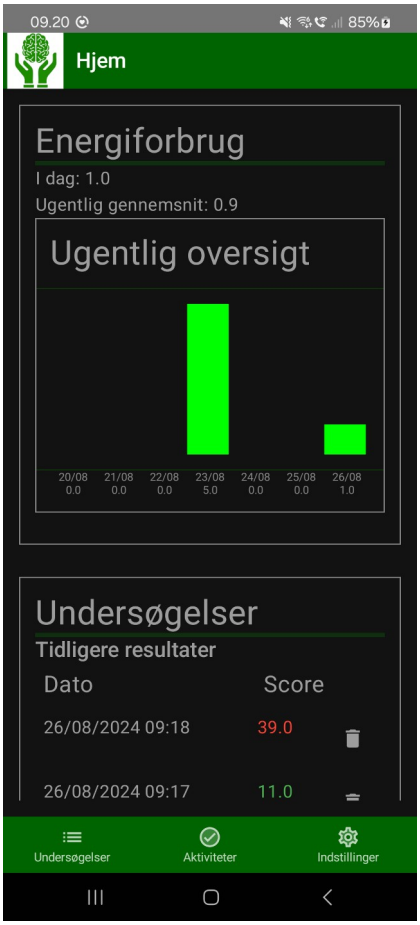
```
fun PsycheAssistantApp() {

    MaterialTheme(
        colors = CustomTheme.psycheColors()
    ) {
```

5.2. Brugerflade funktionaliteten

Når applikationen startes, lander brugeren på ”Hjem” siden. Denne vil, hvis der er fortaget en eller flere undersøgelser, vise en liste over resultaterne – taget fra enhedens lokale lager. *Bemærk at denne del er fra det foregående proof-of-concept projekt*.

Derudover vil der vises en opsummering og visualisering af brugerens energiforbrug, både for (1) den pågældende dag, (2) gennemsnitligt de seneste 7 dage, samt et søjlediagram af det relative forbrug hver dag de seneste 7 dage.

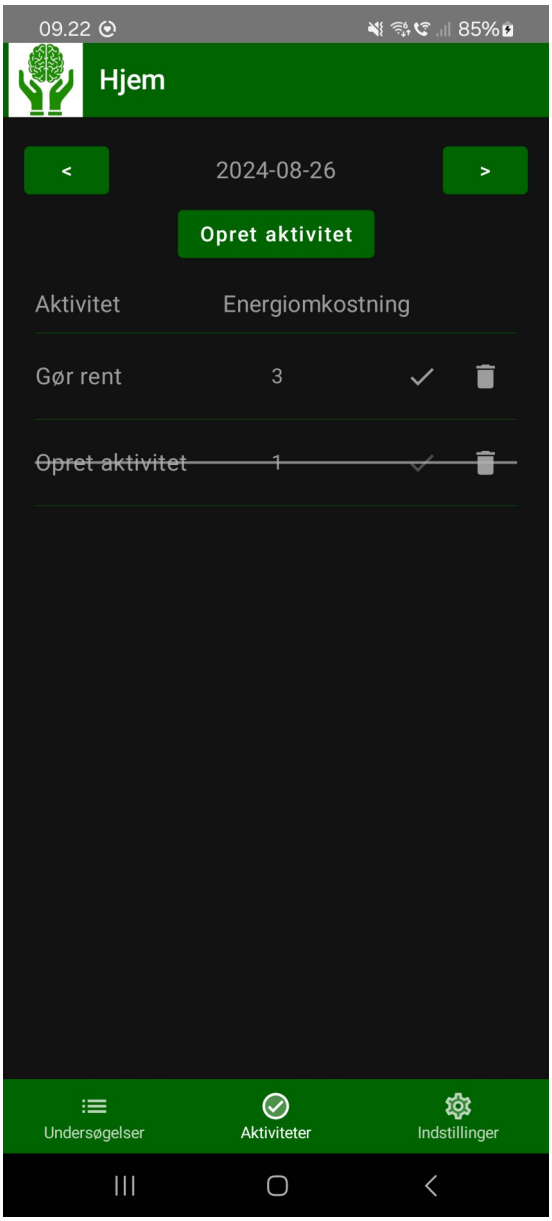
Endelig er der også en simpel navigationsflade, med en navigations-bar i bunden, der indeholder punkter til (1) at tage en MFS undersøgelse (*fra proof-of-concept projektet*), (2) oprette/navigere/handle på aktiviteter, og (3) indstillinger, specifikt i forhold til bruger oprettelse/log ind/ud/slettelse, samt gruppe oprettelse/til- og afmelding/opløsning. Bemærk at punkt 2 og 3 kræver forbindelse til den bagvedliggende API og database, altimens punkt 1 eksisterer offline og lokalt. Derudover er der også en navigations-bar i toppen, til at gå tilbage til forsiden (”Hjem”).

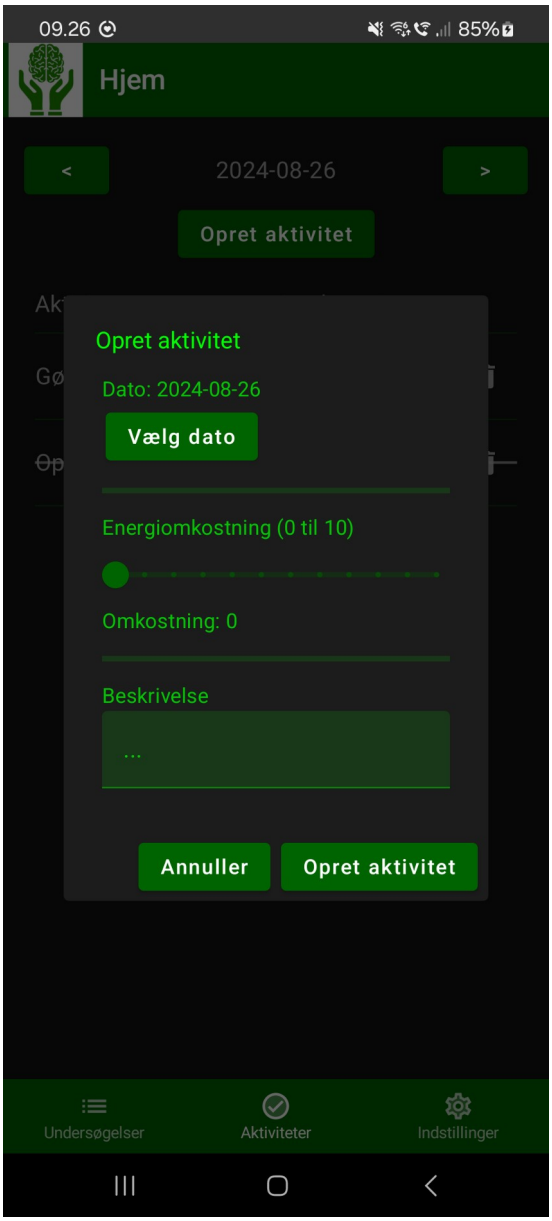


5.2.1. Aktivitet-side

På aktivitetssiden genereres der en liste over de aktiviteter, der gør sig gældende for den pågældende (eller valgte) dato. Når en bruger markerer en af disse som udført, fører det til, at det aktivitets-element gennemstreges, og udfør knappen deaktiveres (udover back-end-delen også ville verificere, om aktiviteten allerede var gennemført, ved forsøg på at gennemføre den igen). De gennemførte aktiviteter bruges derfra til udregning af brugerens daglige/gennemsnitlige energiforbrug, baseret på den dato aktiviteten blev gennemført (*ikke oprettet!*), og af hvem.

Via ”Opret aktvitet” knappen åbnes der en dialog-boks, hvori brugeren kan angive en beskrivelse, energiomkostning, og dato for en ny aktivitet.

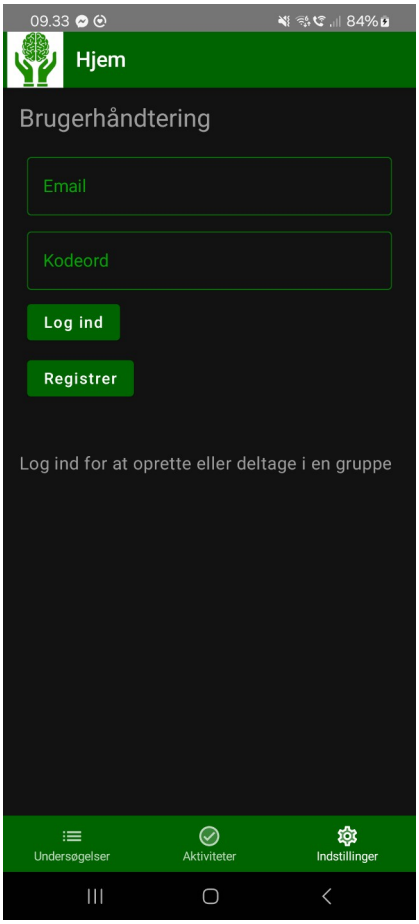




5.2.2. Indstillinger-side

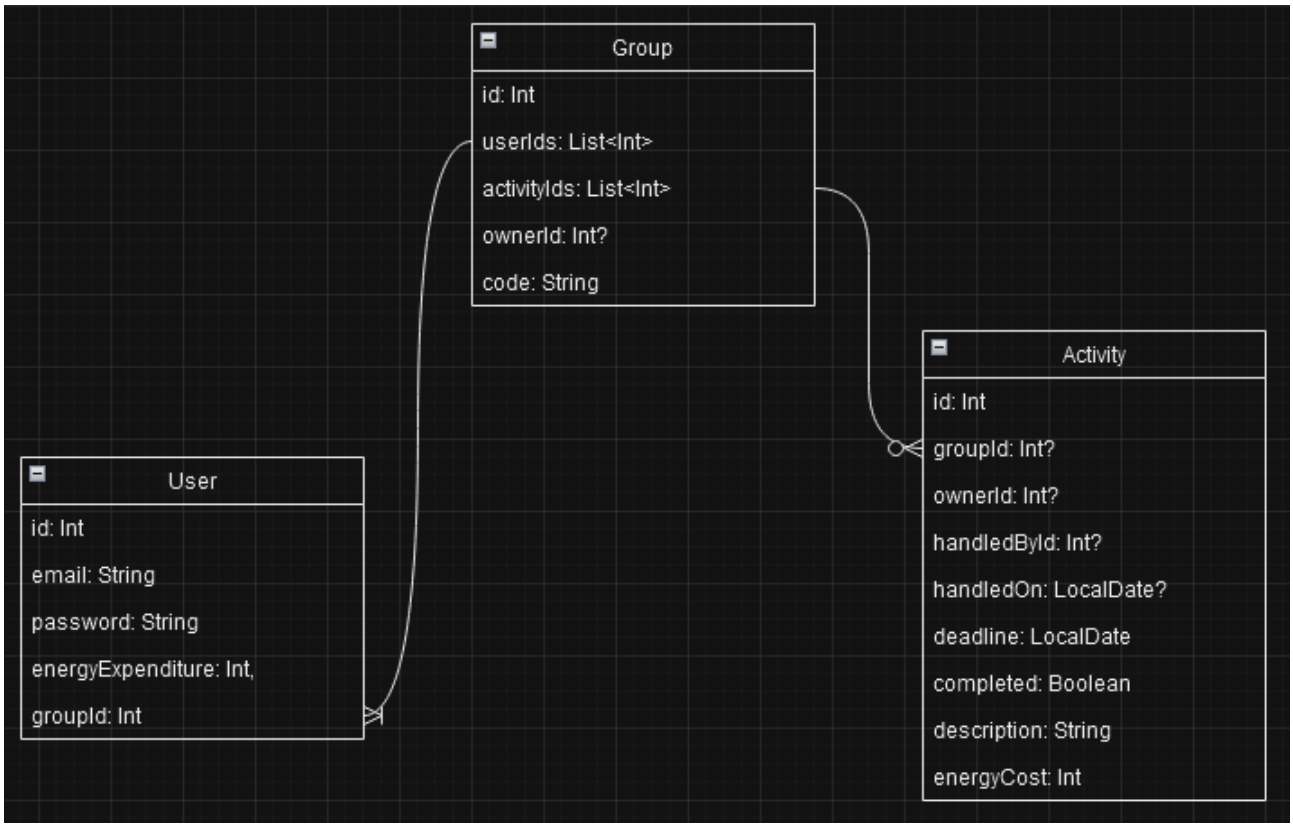
Denne side er den mest informations-intensive side. Dilemmaet her var, at hvis den skulle simplificeres yderligere, ville det kræve en finere opdeling af funktionaliteten, på tværs af flere sider, som så ville forøge den overordnede kompleksitet af brugerfladen. Af samme grund vil man i kildekoden også kunne se, at Bruger- og Gruppehåndtering er findelt i hver sin sektion, og ikke sider. Man kan dog derfor også nemt dele dem ud igen på hver sin side, skulle der være belæg for at dette var mere hensigtsmæssigt.
Da det formodes, at denne side ikke vil bruges nær så ofte som den øvrige funktionalitet, er valget faldet på, at samle konto-centrale administrationsfunktioner på denne side, i stedet for at dele det op i f.eks. Bruger og Gruppe-administrations sider.

Dog er der implementeret en række betingelser, der afgør den specifikke udformning af visningen, afhængig af om man er logget ind eller ej, og om man er en del af en gruppe eller ej, samt om man er ejere af gruppen eller ej, hvor man i sidstnævnte tilfælde har mulighed for at fjerne andre brugere fra gruppen, eller opløse gruppen helt. Derudover vil der i gruppeoversigten også være noteret dagens/sidste uges gns. Energiforbrug for hvert medlem.



5.3. Modeller

Udover den funktionelle tilgang i udformningen af selve visningen, er der også modeller defineret for at kunne repræsentere relevant data, således at data indhentet fra back-end-delen, kan vises og behandles på en objekt-orienteret facon. Som påpeget flere gange, indgår ”Survey” og ”Question” modellerne ikke i dette projekt, men er en del af det forudgående proof-of-concept, så disse udelades i denne gennemgang.



Som det fremgår af Entity-Relation diagrammet ovenfor, indgår 1 eller flere brugere i en relation med 1 gruppe (Many2One), imens 0 eller flere aktiviteter, indgår i en relation med 1 gruppe (ligeledes Many2One).

Bemærk at der refereres til ejer af både gruppe og aktiviteter, samt den håndterende bruger af sidstnævnte, dog sker dette ensrettet, og ikke tovejs, som de øvrige relationer.
Derudover anvendes der i front-end-delen integral-tal til at henvise til identifikations nummeret (id) på de relaterede datasæt, i stedet for at bruge selve modellerne (objektet). Dette er et bevidst valg for at undgå at gøre kommunikationen mellem for- og back-end-delene for kompleks og tung, samt undgå risikoen for rekursion; omend der også er taget foranstaltninger for dette i back-end-delen. I selve back-end-delen er disse relationer imidlertid objekt-baseret, så i stedet for integral-tal anvendes selve objekterne, for at bedre facilitere evt. data manipulationen.

Udover model-definitionerne er der for hver klasse også defineret et Repository, der definere de funktioner der skal være mulige – så altså et interface, samt tilsvarende Controller klasse, der tager

imod kommandoer fra front-end-delen (*dvs. brugeren*), og Service klasse, der omdanner de kommandoer til reel kommunikation med back-end-delen, ud fra de forudbestemte funktioner fra klassens Repository.

5.4. Funktioner og Klasser

Som nævnt gør Compose Multiplatform delvist brug af et funktionelt paradigme i front-end-delen, så hver visning er reelt set en funktion, der kalder diverse grafiske elementer, som så udgør brugerfladen.

Groft sagt kan man derfor sige, at definitionen for disse funktioners formål, er de visninger, der blev gennemgået i afsnit 5.2. Man kan derfor lave en relativ hurtig oversigt over de funktioner, der står for oprettelsen og implementeringen af selve brugerfladen. *Bemærk at visninger fra proof-of-concept projektet er udeladt*:

Funktions navn	**Formål**
PsycheAssistantApp	Funktionen opretter layoutet der kommer til at danne ramme om de øvrige visninger i applikationen
MainPage	Kalder komponenter til at oprette en oversigt over brugerens energiforbrug, samt resultater fra udførte MFS undersøgelser
ActivityPage	Kalder en tabel over aktiviteter for en given dato, samt giver mulighed for at oprette nye aktiviteter.
AccountManagementPage	Kalder sektioner, der giver mulighed for administration af hhv. brugerens egen konto og evt. gruppe.
EnergyOverviewSection	Anvendes til at vise energiforbrug for den pågældende eller specifikke bruger.
GroupManagementSection	En sektion der viser relevant gruppe-information, så som oprette, deltage, forlade eller opløse gruppen, samt tilmeldings koden, og opretter en oversigt over brugere ved at kalde bruger tabellen.
UserManagementSection	Sektion der viser bruger-relaterede administrationsmuligheder, så som opret, log ind, log ud, og slet.
ActivityItemTable	Visning der bruges til at behandle en liste af aktiviteter, ved at forberede en overordnet tabel struktur, og iterere igennem listen, og kalde visningen for det enkelte objekt.

ActivityItem	Visning der behandler enkelte aktivitets objekter, og generere visningen for dem, inde i ovennævnte tabel.
UserItemTabel	Tager imod en liste af brugere, og opretter en tabel lignende struktur, til at kalde visningen for hver enkelt bruger.
UserItem	Danner visningen for et enkelt bruger objekt. Gør brug af en simplificeret EnergyOverviewSection via et argument, der gør at energiforbruget vises stærkt simplificeret, samt et argument for den specifikke bruger, der skal udregnes energiforbrug for.
ActivityCreateDialog	Dialog vindue der bruges, når der på ActivityPage klikkes på knap til at oprette en aktivitet. Tager imod formular input, samt gør brug af en dato-vælger, til at lade brugeren oprette og lagre en aktivitet i den eksterne database.
ConfirmDeleteDialog	Da der flere steder gøres brug af denne sikkerheds advarsel – specifikt ved forsøg på sletning af bruger, og ved forsøg på opløsning af grupper – defineres denne separat, og kaldes fra de pågældende knapper, med de nødvendige strenge/funktioner.
DatePickerComponent	Implementerer et bibliotek til at vise en grafisk dato-vælger, for at simplificere front-end validering af datovalg, og for brugeroplevelsens skyld.
EnergyExpenditureChart	Gør brug af draw funktionalitet til at tegne en ”søjlegraf”, ud fra en brugers samlede energiforbrug på en given dato, 7 dage tilbage i tiden fra dags dato.
ErrorScreen	Forældet visning der blev brugt til at vise fejlbeskeder på deres egen dedikerede side. Denne er sidenhen fravalgt til fordel for at vise fejlen direkte på den relevante side.
LoadingScreen	Visning der bruges til at indikere til brugeren, at applikationen er ved at indlæse – bruges derfor primært i PsycheAssistantApp funktionen.

5.5. Back-end

Back-end delen af projektet består af en Spring Boot-baseret API, der bruger modeller tilsvarende modellerne defineret i front-end-delen, med undtagelsen af, at den som nævnt anvender objektet i relationen, i stedet for et integral tal i form af objektets id. Controller klasser definerer end-points, som front-end-delen kan interagere med, og derigennem manipulere den bagvedliggende data. Dette sker ved at Controller klassen kalder en tilsvarende model-specifik Service klasse i back-end-delen, der står for at validere data integriteten, inden den via tilsvarende Repository klasser interagerer med databasen, ud fra generiske metoder oprettet af JPA (*Java Persistence API*), som opretter de fundamentale interaktioner med SQL databasen, dvs. de typiske CRUD[11] metoder. Bemærk at der dog er defineret mere avancerede, unikke forespørgsler (*queries*) for visse af modellerne, så som Activity, da JPA ikke kan håndtere disse automatisk:

```
@Query("SELECT a FROM Activity a WHERE a.group.id = :groupId AND a.handledOn BETWEEN :endDate AND :startDate")  1 usage
List<Activity> findHandledActivitiesForPeriod(
        @Param("groupId") int groupId,
        @Param("startDate") LocalDate startDate,
        @Param("endDate") LocalDate endDate);

@Modifying  1 usage  MadSantiak
@Transactional
@Query("UPDATE Activity a SET a.handledBy = NULL, a.handledOn = NULL, a.completed = FALSE WHERE a.handledBy = :user")
void updateActivitiesHandledByUser(@Param("user") User user);
```

Den første af de to eksempler ovenover, bruges for eksempel til at finde aktiviteter ud fra disses gruppe, samt den dato de blev håndteret (udført) på, ud fra om denne er mellem de to datoer, som argumenter funktionen kaldes med.
Den anden er en transaktions forespørgsel, der forsøger at opdatere data på aktiviteten. Dette sker i forbindelse med at brugeren slettes, hvorfor den skal være en transaktion; dvs. hvis denne sekundære del fejler, fejler den oprindelige funktion (og altså ”hele transaktionen”) også, hvorved data korrumpering undgås.

Det er også i back-end-delen at autentificerings tokens delen håndteres, via brugen af JWT biblioteket, samt overskrivelser for at denne kan operere med de email-baserede bruger objekter, samt en række hjælpe funktioner til at finde brugere og/eller validere deres token.

5.6. Sikkerhed

Der anvendes en række sikkerhedsforanstaltninger i applikationen. Deres primære formål, er at forsøge at sikre, at brugernes kritiske informationer ikke er umiddelbart tilgængelige.

I front-end-delen anvendes der MMKV til at lagre autentificerings tokens lokalt på selve brugerens enhed, og disse bruges til at verificere brugerens identitet, uden at de skal logge ind hver gang de starter applikationen. Dette udgør dog også en sikkerhedsrisiko, da det betyder, at en tredjepart også ville kunne undgå at skulle logge ind, for at se brugerens informationer (*omend applikationen i øjeblikket ikke gør brug af noget decideret personlig, GDPR-sensitivt data*).

11 Akronym for ”Create”, ”Read”, ”Update”, ”Delete”; en forkortelse for de datamanipulationer man typisk at kunne udføre med objekter i en database.

For at minimere denne risiko, udstedes disse autentificerings tokens med udløbsdato, så hvis de ikke anvendes indenfor en rimelig tidshorisont, frafalder legitimationen, og brugeren skal først autentificere sig ved at logge ind på ny. Ved login udstedes der en primær autentificerings access-token, der udløber indenfor 10 timer, samt en refresh-token, der udløber indenfor 7 dage. På denne måde er der et ekstra lag af sikkerhed, skulle førstnævnte falde i andre hænder, da denne udløber relativt hurtigt, og kræver sidstnævnte, for automatisk at blive fornyet ved udløb.

I back-end-delen anvendes der hashing, specifikt af brugerens kodeord, således at brugerens faktiske kodeord ikke lagres i databasen. Til formålet anvendes BcryptPasswordEncoder biblioteket, som står for både at hashe kodeordene (*med salt*), samt under login, til at verificere, at det angivne kodeord som back-end-delen har modtaget, stemmer overens med det hashede kodeord lagret for brugeren i databasen.

Dette efterlader dog én sårbarhed, nemlig i kommunikationen mellem front- og back-end-delen. For at sikre denne er der anvendt OpenSSL til at generere et self-signed certifikat. Denne bruges til opsætte front- og back-end-delene, således at API'en kræver HTTPS kommunikation via dette certifikat, og applikationen stoler på det udstedte certifikat, som dog kræver en hvis direkte indgriben i den Android-specifikke opsætning for applikationen, for at tvinge den mobile enhed til at stole på den, da den typisk ikke vil stole på et self-signed certifikat, for eksempel hvis man forsøger at installere certifikatet via enhedens egen sikkerheds indstillinger.

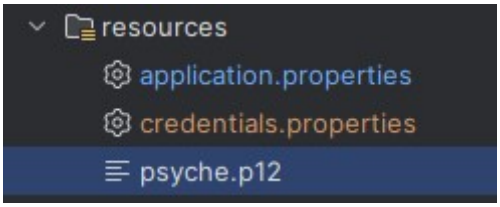
Først oprettes der et certifikat samt en nøgle, for den IP-adresse serveren vil have på netværket:

```
$ openssl req \
-newkey rsa:2048 \
-nodes \
-x509 \
-days 36500 -nodes \
-addext "subjectAltName = IP.1:192.168.1.145" \
-keyout psyche.key \
-out  psyche.crt
```

Dernæst konverteres certifikatet og nøglen til et .p12 bundt, som kan anvendes af Spring Boot API'en i back-end-delen:

```
$ winpty openssl pkcs12 -export \
-in psyche.crt \
-inkey psyche.key \
-out psyche.p12 \
-name psyche \
-CAfile psyche.crt \
-caname root
Enter Export Password:
Verifying - Enter Export Password:
```

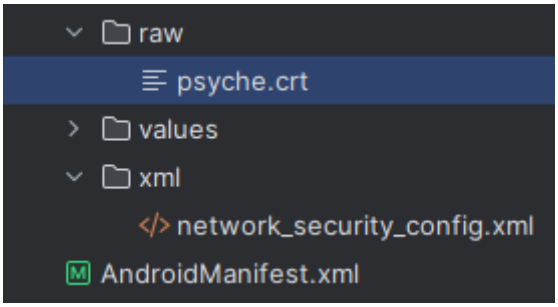
Som dernæst ligges i dennes ressource mappe – bemærk her at *credentials.properties* er en konfigurationsfil der indeholder ting så som nøgler, token levetid, og database legitimationsoplysninger, og som fritages fra GitHub commits, som vist tidligere:



Endelig sættes back-end-delen op til at anvende certifikatet, i *application.properties*. Bemærk at førnævnte credentials.properties som sagt importeres her, for at anvende fornævnte sensitiv information.

```
spring.config.import=optional:classpath:credentials.properties

# HTTPS Configuration:
server.port=8443
server.ssl.key-store=classpath:psyche.p12
server.ssl.key-store-password=${ssl_pass}
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=psyche
```

For at gennemtvinge en app-specifik tillid til certifikatet, ligges denne i front-end-delen, specifikt i den Android-specifikke opsætning, under res/raw mappen:



Hvor den derefter defineres eksplicit i network_security_config.xml:

```xml
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/psyche"/>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

Som så endelig anvendes i android applikationens opsætning, inde i AndroidManifest.xml:

```xml
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="PsycheAssistant"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@android:style/Theme.Material.Light.NoActionBar"
    android:networkSecurityConfig="@xml/network_security_config">
```

Dermed opnås der en mere sikker HTTPS kommunikation mellem API og applikationen, hvorved det ikke er nær så stor en sårbarhed, at kodeordet sendes som ren tekst, eftersom anmodningen nu krypteres.

6. Udviklingsforløb

Som påpeget flere gange i løbet af denne rapport, bygger dette projekt på et forudgående proof-of-concept projekt, hvis hovedmål var at give brugerne mulighed for at selv-administrere anerkendte undersøgelser, til at vurdere sværhedsgraden af deres mentale træthed. Dette forudgående projekt blev også udarbejdet i samarbejde med Simone Larsen, i forhold til f.eks. scoring, visning, og praktisk brugbarhed.
Per definition var proof-of-concept projektets primære mål, at lave et værktøj, som dem der lider af mental træthed, nemt kunne bruge, uden nødvendigvis at have adgang til internet, og være er produkt, der skulle være så lettilgængeligt som muligt, ift. omkostning.

Ønsket har hele tiden været at udvide den fundamentale funktionalitet, til noget der ikke kun kunne agere måleinstrument, men også behandlingsværktøj, af mental træthed.

Ideen til dette projekt er derfor også blevet udarbejdet i samarbejde med Simone Larsen, der ud fra hendes private så vel som professionelle erfaring har stor viden omkring mental træthed, behandlingsmuligheder, faldgrubber, osv..

Specifikt var diskussionen i denne indledende fase, hvilken funktionalitet der teoretisk – og praktisk – kunne fungere som et effektivt værktøj til behandlingen af mental træthed, som defineret i problemformuleringen og formåls sektionen i denne rapport.

Selve udviklingen er blevet udført udelukkende af Mads Søndergaard, og er blevet udført ovenpå det tidligere proof-of-concept projekt, hvor langt størstedelen af det forudgående projekt dog ikke har været relevant for det faktiske udviklingsarbejde i forbindelse med dette projekt, med undtagelse af en funktionel tilføjelse til den eksisterende landingsside og udvidelse af navigations elementerne.
Med andre ord, er der ikke tale om, at dette projekt har draget nytte af et tidligere projekt, tværtimod har der være et større arbejde med at få tilføjet den nye funktionalitet defineret i forbindelse med dette projekt, til det allerede eksisterende økosystem, uden at forstyrre den eksisterende funktionalitet fra det forrige projekt – for eksempel som at bibeholde resultater for undersøgelser lokalt på enheden, og ikke låse det bag nødvendigheden af en online forbindelse. Det blev på et tidspunkt overvejet, om den oprindelige funktionalitet skulle migreres til at være online, som et sekundært mål for dette projekt, men blev fravalgt, da det ville stride imod proof-of-concept projektets fundamentale hovedmål, nævnt ovenfor.

6.1. Metodik

Grundet den funktionelle natur af Compose Multiplatform specifikt, og mobile applikationer generelt, var det ikke umiddelbart hensigtsmæssigt at dele udviklingen kategorisk op i front- og back-end-dele, så hele back-end-delen først blev oprettet med alle modeller og klasser, og dernæst front-end-delen. Det er derimod foregået vertikalt og funktions-drevet. Med det menes, at der her, for eksempel, er fokuseret på at implementere User modellen i front-end- og back-end-delen på samme tid (et *vertikalt ”snit” på tværs af back/front-end*), samt den faktiske implementering af dette (*de funktioner der skal være mulige for netop User modellen*), inden arbejdet er fortsat med næste vertikale snit – den næste model og dens relevante front- og back-end-del, og dens funktionelle implementation i brugerfladen.

Denne fremgangsmåde har haft den fordel, at det har været en naturlig del af udviklingsarbejdet, at løbende udføre funktionelle test af brugerfladen, og dermed funktionaliteten af den bagvedliggende kode. Ulempen har været, at projektets struktur, samt brugerfladens implementation, har skulle gennemgå flere refaktoreringer, for at opnå en tilfredsstillende gennemsigtighed og letlæselighed i koden.
Dog skal det nævnes, at selv denne ulempe kan ses i et mildere lys, da denne løbende refaktorering har sikret, at koden også naturligt har tilskyndet sig et objekt-orienterede ideal, med modulær struktur, og undgået – til trods for brugerfladens funktionelle struktur – at munde ud i for store ”mundfulde” af kode. I stedet er det delt op i sider, sektioner, tabeller, enheder, komponenter, osv.

6.2. Projektstyring

Udover gennemgangen her, henvises der også til Bilag 6: Projektplan og Log, for mere det granulær gennemgang af tidsforbruget og udfordringer.

Da dette har være et solo-projekt, har der ikke været et større behov for oversigtsskabende værktøjer, så som implementering af en kanban baseret plan. Det blev vurderet, at selv om dette ville være visuelt tiltalende i den endelige rapport, ville den tid det ville tage at oprette og vedligeholde, være større end den fordel evt. planlægning og brugbarhed den ville producere.

Tidsstyringen er derfor sket via den oprindelige projektplan, med et overordnet fokus på, at mindst 5 dage skulle sættes til side til rapport skrivning. Det viste sig hurtigt ved projektets start, at grundet blandingen mellem funktionel- og objekt-orienteret paradigmer, var den oprindelige plan ikke umiddelbart realistisk, da denne var mere horisontalt inddelt efter lagene, hvorimod en funktionel tilgang krævede, at der var et større fokus på vertikale snit, så funktionel afprøvning kunne ske for hvert skridt i udviklingen. Denne fundamentale forskel i opbyggelsen af den oprindelige og faktiske projektplan, gør også, at en evt. sammenligning via kanban ville være misvisende, da de reelt skulle stå 90 grader diagonalt på hinanden, for at afspejle den horisontale kontra vertikale tilgang.

Disse vertikale snit har dog affødt en langt mere overskuelig struktur, da det fra start af har været en konstant overvejelse, om kode kunne genbruges andre steder, og at strukturen var overskuelig, for nemt at kunne gå i gang med næste vertikale snit, modsat en overvejelse i forbindelse med gennemgang af kode.

Der har fra start være et fokus på de mulige brugsscenarier, der måtte være for brugeren af applikationen, så som oprettelse, log ind, og log ud, samt oprettelse og behandling af aktiviteter og grupper.
Der har også været tid til at indføre yderligere brugsscenarier, så som slettelse af brugere og grupper, samt data visualisering af behandlede aktiviteter. Eneste nævneværdig tidsrøver, har været enkelte anfald af Hortons migræne hos udvikleren, samt en opdatering af IntelliJ IDEA IDE'en der blev anvendt, som var nødvendig for at debugge asynkrone funktioner, der resulterede i en god dags fejlsøgning og en fejlmelding til JetBrains, da opdateringen løste et relativt kritisk debugging problem, men ødelagde spejlvisningen af den fysiske mobile enhed.

6.2.1. Afprøvning

Den primære afprøvningsmetodik har bestået at løbende funktionel afprøvning af brugerfladen og dens funktionelle kobling til back-end-delen. Valget er faldet på en funktionel frem for programmatisk afprøvningsmetodik, som følge af applikationens tendens mod et funktionelt paradigme. Der var derfor mere fokus på, at brugerens gøren og laden i applikationen, også funktionelt hang sammen og ikke gav anledning til fejl, så som ved forsøg på at udføre en allerede udført aktivitet. Dette gjorde også, at der under udvikling blev lagt vægt på validering af brugerens handlinger og input, og dermed også en høj grad af kontrol over data integriteten, hvorfor en programmatisk test af objekt validitet blev mindre vigtig, og det blev vurderet, at afkastet fra et

sådant afprøvningsparadigme ville være mindre, i forhold til tidsinvesteringen, i forhold til at bruge funktionel afprøvning.

Dette hænger også sammen med, at applikationen blev udviklet i samarbejde med en fagekspert, så for at sikre produktets validitet, var kvalitetssikring i form af funktionel afprøvning fra fageksperten, også et større fokus.

Der henvises til Bilag 3: Brugs- og testscenarier for en oversigt over det funktionelle brugsscenarier der løbende er blevet af- og efterprøvet.

6.3. Feedback

Både før, under og efter denne udviklingsproces, har der løbende været feedback fra fageksperten – Simone Larsen – som har vurderet effektiviteten af den faktiske implementation af f.eks. energiomkostnings systemet, hvilken skala brugeren skulle kunne vælge når der oprettes aktiviteter, samt feedback i forhold til den visuelle præsentation.

Simone har også vendt projektets formål og implementation med kolleger, som har udvist begejstring for produktet som hjælpemiddel og værktøj, og fået feedback på den praktiske nødvendighed og brugbarhed af produktet herigennem, samt vedrørende nogle af de eksisterende ideer til videreudvikling af produktet, som f.eks. afsnit 6.4 længere nede.

Det har også været på tale at inddrage bruger-baseret afprøvning og feedback, for eksempel ved at invitere facebook-grupper for folk med senfølger fra COVID-19, Simones kolleger, el.lign., men dette ville have krævet, at applikationen var tættere på en færdig version, som man med god samvittighed kunne udgive eller tilbyde at installere på potentielle brugerers enheder.
Af samme grund blev denne mulighed fravalgt i sidste ende, til fordel for den mere fokuserede afprøvning foretaget af udvikler og fagekspert.

Der henvises til Bilag 4: Bruger feedback, for en gennemgang af den feedback der har været relevant i forbindelse med kvalitets afprøvningen, og hvad udfaldet af denne feedback har været.

6. Fremtidig udvikling

Til trods for at produktet har opnået de primære mål, er der en række måder hvorpå, at produktet ville kunne både forbedres og udvides. Følgende er ikke en udtømmelig liste, men fremhæver nogle af de mest relevante og nærliggende muligheder, i forhold til eksisterende funktionalitet og formål.

6.1. Gruppe-chat

En oplagt udvidelse ville være at inkludere en gruppe-chat funktionalitet. Dette ville give gruppen – om den så består af én borger med mental træthed og dennes pårørende, eller én ergoterapeut med dennes tildelte borgere – en måde at, at tale sammen med hinanden omkring udfordringer, tanker, og overskud, og på den måde kunne være med til at tilbyde yderligere støtte, til dem der lider af mental træthed, samt endnu større indblik i disses situation, for pårørende og ergoterapeuter.

6.2. Alarmer og Notifikationer

En anden oplagt udvidelse ville var inklusionen af tidsbaserede alarmer. Som det er nu, er aktiviteter isolerede datasæt tilgængelige i applikationen. Tanken her ville være, at give de forskellige aktiviteter et tidspunkt, som ville udløse den mobile enheds alarm, og/eller sende en notifikation. Dette ville gøre det mindre krævende for brugeren at anvende aktiviteterne (*og huske de er der*), men samtidig også forbedre muligheden for samarbejde mellem brugerne i en gruppe; en aktivitet oprettet af bruger 1, kunne udløse en notifikation hos bruger 2, 3, osv., når udførselstidspunktet nærmede sig, således at disse kunne assistere bruger 1 i at få fuldført aktiviteten, eller udføre den på dennes vegne.

Dog ville det nok være en god ide, at også inkludere en måde at frabede sig notifikationer og alarmer, enten fra applikationen som helhed, eller fra aktiviteter oprettet af specifikke brugere (*evt. kun modtage det for dem man selv har oprettet*).

6.3. Migrering af proof-of-concept funktionaliteten

Som nævnt blev dette fravalgt da det ville bevæge sig væk fra det foregående projekts primære mål, i forhold til altid at være tilgængeligt for brugeren.

Dog må man også sætte spørgsmålstegn ved, hvor kritisk en funktionalitet det ville være, at man ikke lige kunne foretage en MFS undersøgelser nu og her, men skulle vente til man nåede ud af en af de få døde signal zoner telenettet i Danmark har, teoretisk set. Derfor er dette stadig en udvidelsesmulighed der er under overvejelser.
Det kræver selvfølgelig at der også er en fordel ved at migrere disse data til en ekstern database. Udover selvfølgelig at være tilgængelig for brugeren på tværs af enheder, ville det også gøre det muligt, at præsentere endnu mere interessant data til brugerne i gruppen. Man kunne for eksempel lave en søjlegraf for alle i bruge oversigten, og så her notere deres seneste MFS score, som en rød linje, der dermed visualiserede for andre (og brugeren selv), når de havde oversteget deres mentale

træthed på en given dag, og dermed risikere at forværre den. Dette ville så kunne bruges til at udregne tendenser, så som effekten på fremtidige MFS målinger, når energiforbruget overstiger den mentale træthed, kontra hvis brugeren holder energiforbruget under denne.

6.4. Andre mental udfordringer

Ovenstående ville også åbne døren for, at selve MFS undersøgelsen ikke ville være hård kodet, men i stedet være en række spørgsmål tilhørende en model, med et definerende mærkat (eller tilhørende en specifik undersøgelse-models datasæt). Det naturlige skridt herfra, ville være at brugere kunne vælge mellem flere forskellige slags undersøgelser, som er beregnet til at følge udviklingen af specifikke tilstande. Dette kunne være depression, søvnkvalitet, osv.

Med andre ord ville man kunne udvide det diagnostiske værktøj, til at inkludere flere forskellige mentale udfordringer, der ligeledes kunne drage nytte af løbende opfølgning og evt. kommunal indsats (*både i den offentlige og sociale forstand*).

6.5. Forum i applikationen

En anden mulighed ville være at inkludere et decideret forum i applikationen, således at brugere – på tværs af grupper – kunne yde støtte og rådgivning, og fungere som en slags fælles terapi. Dette har man set gavner andre i samme type situationer, hvor deres livssituation afføder en vis mængde skam, på den ene eller anden måde, og hvor det, at vide der er andre i samme båd, samt at kunne kommunikere med disse, gør meget for at løsne op for de dogmer, der kan være med til at fastholde dem i lidelsen. Specifikt for dem med mental træthed, er det ikke unormalt, at de selv og deres omgangskreds har svært ved, at acceptere, at de har en psykisk forhindring, der gør det usandsynligt, at det kan varetage en normal dagligdag – socialt og professionelt. Dette fører ofte til, at de kaster sig ud i et forsøg på at varetage deres tidligere energiforbrug til trods, blot for at blive udtrættet yderligere, og med andre ord sat ”tilbage til start”.

Dette specielt set i lyset af tilstandens fortsat relative ukendte status blandt den gængse befolkning og det offentlige.

7. Begrænsninger

Omend produktet har opfyldt projektets primære mål, er der en række mangler, man ville kunne definere som begrænsninger, i den forstand, at det ville have være rigtig rart at have i applikationen allerede – rent praktisk er de dog ikke uopfyldte primære mål.

Der er en tydelig – og bevidsts – begrænsning i den mængde data der lagres om brugeren. Man kunne meget vel have ønsket sig mere data, som så ville kunne være med i de eksisterende statistikker vedr. mental træthed. For eksempel kunne det være relevant, at kende til brugerens alder, behandlingsforløb, osv., i forbindelse med evt. justering af forventet og aktuel energiforbrug.

Dog er dette som sagt en bevidst udeladelse, da det ikke er strengt nødvendigt, eftersom MFS er alder-agnostisk, bl.a..

Man kunne også have ønsket sig, at det var muligt for brugere at nulstille deres kodeord, og funktionaliteten blev også overvejet, men blev til sidst fravalgt, da dette ville kræve en mere tidskrævende implementering af f.eks. email-server, således at back-end-delen ville kunne sende og håndtere nulstillingsanmodninger til brugerens email. Det blev derfor vurderet, at en sådan funktionalitet ville tage for meget af den tildelte udviklingstid, i forhold til hvor meget det ville gavne produktet; det er dog noget der fremadrettet ønskes tilføjet.

Produktet er primært tiltænkt selve individer med mental træthed, som så kan spore deres aktivitet og energiforbrug, og lade deres pårørende tage del- og få indsigt- i deres tilstand. Dog er det også en teoretisk mulighed, at den kan bruges af en sundhedsfaglig, med flere individer med mental træthed som de ”andre brugere” i gruppen, således at én sundhedsfaglig kan følge flere patienter. Dog er dette underudviklet for nuværende, og indeholder en række problematikker, der ville skulle håndteres, før det ville være en reel mulighed. For eksempel ville det være uhensigtsmæssigt i et sådant scenarie, at den sundhedsfagliges borgere, ville kunne se og agere på hinandens aktiviteter. For at opnå en sådan funktionalitet, kunne man evt. angive på gruppe-niveauet, om en given gruppe er ”egocentrisk” – omhandlende den enkelte bruger (*med mental træthed*) – eller ”allocentrisk”, hvor den centrale bruger er den sundhedsfaglige, og fokus er på de øvrige brugere.
I førstnævnte tilfælde ville gruppen fungere som nu, hvorimod den i sidstnævnte tilfælde ville fungere ved, at aktiviteterne var unikke for brugeren der oprettede den, og ikke tilgængelig for de øvrige i gruppen. Dette kunne være ved at angive et boolsk felt på gruppe-modellen, som så indikere, om gruppen er det ene eller andet. Hvis den er allocentrisk, udskrives aktiviteterne tilhørende gruppen kun på brugerens enhed, hvis den bruger der er logget ind, er den samme som ejeren af aktiviteten, og evt. energiforbrug ville kun være synlig for brugeren selv og gruppens ejer.

8. Diskussion

Produktet har opnået de primære mål for projektet; brugere kan deltage i grupper, og via disse oprette aktiviteter, som kan handles på, og som vil resultere i, at den handlende brugers energiforbrug stiger, og kan følges af andre interessenter, så som ergoterapeuter eller familiemedlemmer og bekendte – som rent teknisk også ville være i stand til at aflaste den der lider af mental træthed, ved at kunne se og handle på aktiviteter der skal udføres en given dato.

Derudover er der også blevet tilføjet en række funktionaliteter der rækker udover de primære mål, da de viste sig at være en naturlig forlængelse af den implementerede funktionalitet. For eksempel er der specificeret en ejer af en given gruppe, som kan administrere hvem der er brugere af gruppen, eller nedlægge gruppen helt. Derudover er der også tilføjet muligheden for at slette sin egen bruger fra databasen, med tilhørende advarsel for at sikre, at slettelsen ikke sker ved en fejl.

Der også tilføjet data visualisering på landingssiden, i form af den summerede energiomkostning for brugerens behandlede aktiviteter på dagen, gennemsnittet henover de sidste 7 dage, samt et statistisk overblik i form af et tegnet søjlediagram.

Endelig er der også implementeret en række sikkerhedsforanstaltninger, både for brugeren selv, for brugervenlighedens skyld, og den programmatiske sikkerhed.

Brugerens kodeord lagres for eksempel hashet i databasen, ved hjælp af et krypteringsværktøj, der også sørger for at salte det specifikke kodeord, samt verificere, at et givent kodeord stemmer overens, med det der er gemt i databasen, for den pågældende bruger.

Derudover anvendes der også JWT til at oprette en adgang- og genopfrisknings token til brugeren når denne logger ind, som så bruges til at verificere med serveren når denne åbner applikationen igen, således at der ikke skal logges ind igen med det samme – men også så disse udløber indenfor rimelig tid, så en tabt enhed ikke nødvendigvis medføre direkte adgang til applikationens data.

Da det foregående proof-of-concept projekts funktionalitet er intakt, opfylder produktet nu både det foregående samt det aktuelle projekts primære mål, og derudover også nogle tilføjede ”*nice-to-have*” funktionaliteter, så som data visualisering og udvidede brugsscenarier.

Det har være et fokus at bibeholde simpliciteten, men samtidig give produktet mulighed for den ekstra funktionalitet, der ville komme igennem brugen af ekstern data og kommunal brugerenheder (grupper). Dette er gjort ved at undgå at præsentere for meget information, og dermed overbelaste brugeren, således at det fortsat er nemt at navigere og orientere sig, selv om ens mentale energiniveau er lavt.

Det er forfatterens indtryk, at selvom dette projekt – i forhold til det rent software mæssige – har været et solo-projekt, har det ikke være uden yderst vigtige aspekter af samarbejde, så som det tætte samarbejde på tværs af faglige grænser. Dette har resulteret i et produkt, der både teoretisk og praktisk giver mening. Det har, selvfølgelig, til tider betydet, at arbejdsbyrden har være relativt større.

Endelig kan man sætte spørgsmålstegn ved visdommen af, at gå i krig med et projekt, hvor selve sproget og frameworket ikke er det, som forfatteren har den største erfaring med, og som derudover også er et relativt nye. Men denne ekstra udfordring har værer med til at gøre udviklingen mere spændende, og tilbudt uvurderlig indsigt i processen med at sætte sig ind i og anvende et u-familiært framework og/eller sprog, i en reel situation.
Derudover har et af de primære tanker bag projektet – både det forudgående og aktuelle – været, at produktet skulle kunne nå ud til så mange brugere som muligt, og være så let at tilgå som muligt. Derfor har et multiplatform framework været at foretrække, og blandt disse var Kotlin Multiplatform et, som udvikleren i kraft af det foregående proof-of-concept projekt, nu havde en vis familiaritet med, i forhold til andre lignende frameworks.

9. Konklusion

Psyche Assistant applikationen tilbyder nu brugeren, en måde hvorpå de kan spore deres aktivitetsniveau, og dermed danne sig et overblik over, hvor meget energi de faktisk forbruger i løbet af en dag, samt henover en uge. Dette gør det også muligt for brugeren at overveje, om de tildeler en rimelig energiomkostning til de aktiviteter de opretter, eller om de for under- eller overvurderer, hvor meget energi en given aktivitet faktisk bruger.

Den gør det også muligt for dem, at inddrage interessenter, så som pårørende eller sundhedsfaglige, i grupper, således at de kan følge deres forbrug, samt de aktiviteter de – eller interessenterne – opretter. Der bliver dermed også et værktøj hvorved pårørende kan få mulighed for at opnå større indsigt i – og forståelse for – hvad mental træthed faktisk er, og hvad konsekvensen af et overforbrug af energi har, for en person der lider af mental træthed, og hvordan subjektiv antagelse af energiomkostning for en aktivitet, objektivt kan være vidt forskellig, fra den faktiske omkostning for den, der udfører aktiviteten.

Dermed tilbyder produktet en vedholdende, regelmæssig behandlingsmulighed, der gør borgeren opmærksom på deres energiniveau, og gør det muligt for dem, at proaktivt vedligeholde denne, og få løbende indsigt i, hvor meget energi de faktisk bruger, kontra hvor meget de tror de bruger – i kraft af de nu også kan sammenligne faktisk forbrug med scoringer på MFS undersøgelser.

Derudover er produktet udformet på en sådan måde, at dens regelmæssige brug er simpel og ligetil, med bruger og gruppe administration som det eneste (*relativt*) komplekse brugsmønster. Designet er simpelt, og gør brug af æstetiske principper i farvevalg, for at minimere belastning men samtidig tilskynde opmærksomhed når nødvendigt.

Endelig tages der hensyn til brugernes privatliv, idet der ikke indhentes nogle personlige oplysninger – navn, adresse, alder, CPR, osv. - og det der indhentes, specifikt email, lagres på én central server, der kræver autentificering i API-kaldene, før data kan tilgås. Derudover har brugeren mulighed for, at slette deres brugerkonto helt fra serveren, skulle de ønske det.

10. Bilagsoversigt

Bilag 1: Kravspecifikation (16 sider)

Bilag 2: Kildekode Indeks (100 sider)

Bilag 3: Brugs- og testscenarier (8 sider)

Bilag 4: Bruger feedback (4 sider)

Bilag 5: Flowchart og Brugervejledning (17 sider)

Bilag 6: Projektplan og Log (13 sider)

Bilag 7: Klasse oversigt. (13 sider)

Bemærk: Sideantal er absolutte sider inklusiv forside; ikke normalsider.