PROCESRAPPORT

‘Mange Bække Små’

Resumé:

Procesrapporten beskriver udviklingsforløbet samt valgte teknologier af mobilapplikationen ‘Mange Bække Små’.

Udarbejdet af
Gruppe 6 - Elvin E. A. Jensen
Hold: 1205hf22026p

Titelblad

Projekttitel:	Mange Bække Små
Projektperiode:	21/02-2022 - 29/03-2022
Hovedvejleder:	John Storm Ellehammer
Skole:	TEC, Ballerup
Hold:	1205hf22026p
Projektgruppe:	6
Antal sider:	25 (19,5 normalsider)
Antal bilag:	1

1 Læsevejledning

Denne rapport beskriver selve udarbejdelsen og planlægningen af mobilapplikationen 'Mange Bække Små', samt teknologiske valg truffet ifm dette og begrundelse for disse.

Al front- og back-end kildekode er at finde på GitHub i følgende repositories.

- MBS mobilapplikation:

https://github.com/Nivle84/MBS

- MBS REST API:

https://github.com/Nivle84/MBS_API

2 Indledning

Det er et faktum at vi som bevidste væsner besidder et såkaldt "negativity bias"[1]. Der findes adskillige videnskabelige studier[2] som dokumenterer dette fænomen, og hvordan det har en negativ indflydelse på vores selvopfattelse.

"Therefore, when you think rationally and realistically you end the vicious circle of negative thinking. For this reason, you shouldn't allow your fanciful mind to think what it wants and how it wants. In fact, you must train your mind to think according to the relevant data, to reality."[3]

Som beskrevet i citatet ovenover, kan den negative cirkulære tankegang brydes ved at fokusere på relevant data bundet i virkeligheden. Det er dette jeg vil forsøge at sætte fokus på vha MBS, således at brugeren bliver mindet på sine succeser og ikke lader sig rive med ved at fokusere på nederlagene, som "vejer mere" i vores til tider skrøbelige psyke.

3 Problemformulering

Vi lever i en travl verden hvor den ene hverdag hurtigt tager den anden. Vi glemmer at sætte pris på vores succeser, og i stedet husker på vores nederlag som hurtigt bliver et fokus, hvilket skaber en uhensigtsmæssig, ond cirkel af tankemæssig negativitet og følelsesmæssig ubalance.
Men vi oplever alle små succeser i hverdagen, og som vi ved gør *mange bække små en stor å*. Jeg ønsker derfor at lave en mobil app som hjælper brugeren til at indse og blive mindet på deres små succeser i hverdagen, for at vende en evt. negativ cirkulær tankegang i en positiv retning.

Gennem projektet håber jeg at jeg at kunne besvare følgende spørgsmål:

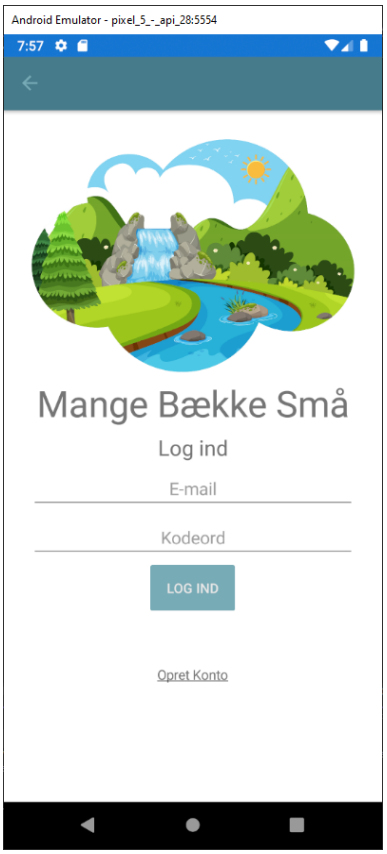
[1] https://greatergood.berkeley.edu/article/item/how_to_overcome_your_brains_fixation_on_bad_things
[2] https://www.nature.com/articles/s41598-019-50821-w
[3] https://exploringyourmind.com/how-to-break-the-chain-of-negative-thinking/

- Kan min løsning gennem brugerens egne dokumenterede data effektivt visualisere en tendens for brugerens succeser/nederlag?
- Kan UI/UX være med til at bidrage til en positiv oplevelse?
- Hvordan kan brugeren motiveres til at benytte app'en hver dag?

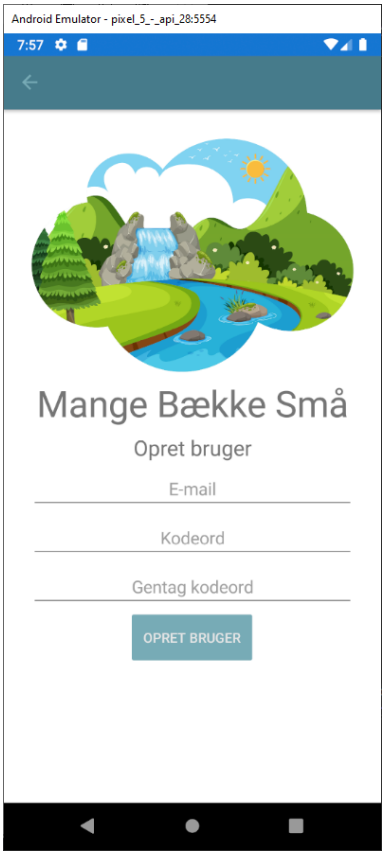
4 Vejledning til programmet

Som beskrevet i Produktrapportens afsnit 3.2.6 og 3.2.7.1, er MBS designet ud fra simple designprincipper som straks kan genkendes og bruges af den gængse bruger.
Denne vejledning er udelukkende for læseren af denne rapport.

Ved start af programmet ses login skærmen (figur 1).
Ved klik på "Opret Konto" vises figur 2.



Figur 1.



Figur 2.

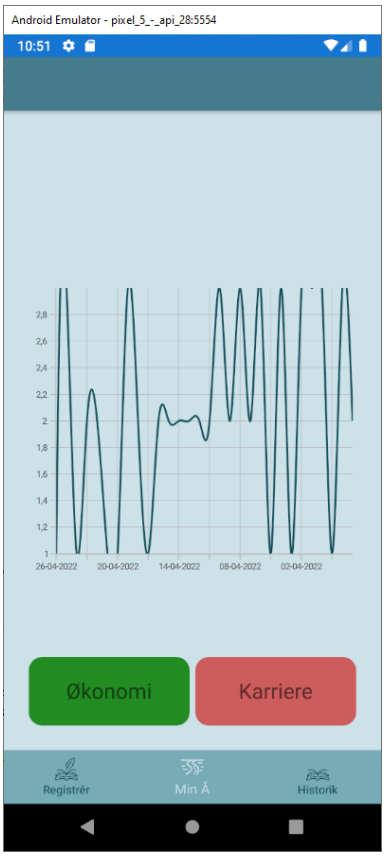
Ved succesfuldt login ses "hovedsiden" hvor brugeren kan registrere en dag (figur 3).
Når brugeren har valgt både en "smiley" (figur 4) samt en tilhørende "påvirkning" (figur 5), får brugeren mulighed for at skrive en note til dagen (figur 6). Ved fejlklik af enten smiley eller påvirkning kan brugeren vende tilbage og rette sit valg ved tryk på "TILBAGE" knappen.

Figur 3.

Figur 4.

Figur 5.

Figur 6.

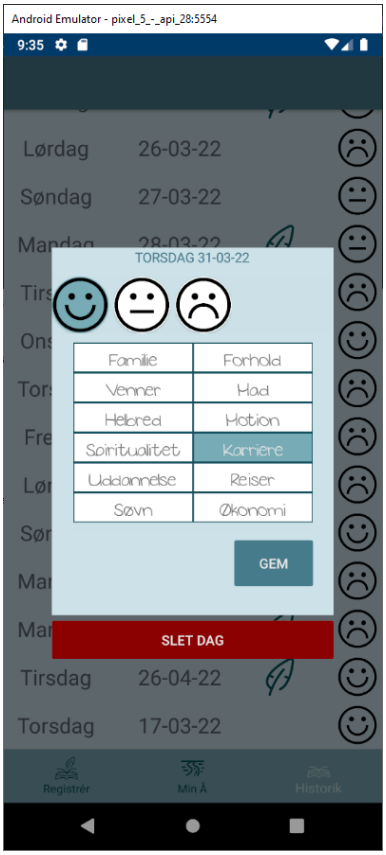


Figur 7.

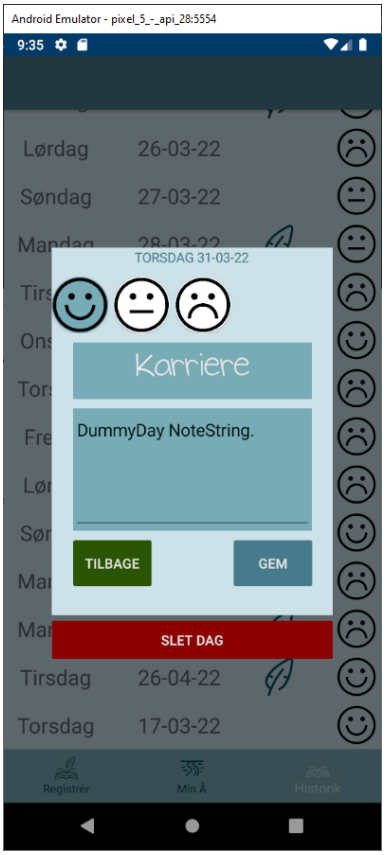
Ved at klikke på "GEM" sendes brugeren videre til skærmen 'Min Å' (figur 7), som også kan tilgås via knappen i bunden af app'en. På denne side ses en graf over brugerens valg af smiley ved de 30 seneste registreringer (god smiley i toppen, skidt smiley i bunden).
Under grafen ses de to valgte påvirkninger ved disse 30 registreringer, som er hyppigst associeret med hhv. "gode" og "skidte" registrerede dage.

På historiksiden (figur 8) ses en liste af alle brugerens registrerede dage.
Ved at klikke på en af disse åbnes et popupvindue (figur 9) med den pågældende dags informationer. Her kan der (meget lig hovedsiden), vælges smiley og påvirkning samt skrives en note (figur 10), som så ændrer den registrerede dag.

Figur 8.



Figur 9.



Figur 10.

5 Projektplanlægning

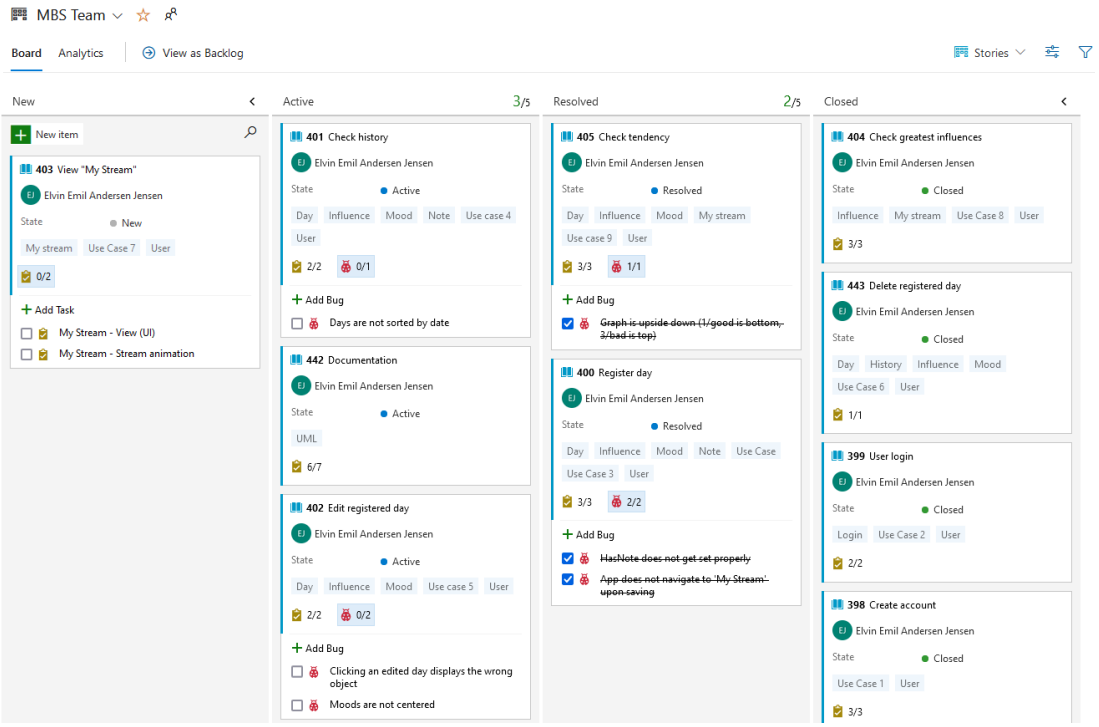
Med mine erfaringer fra at arbejde hjemmefra ifm. Covid-19 pandemien frisk i hukommelsen, valgte jeg fra projektets start af at bruge TEC som min arbejdsplads, skønt der var lagt op til at arbejde hjemmefra under forløbet.
Det er valgt på baggrund af at jeg har svært ved at adskille arbejds- og fritid når jeg befinder mig i mit eget hjem, samt at jeg som enkeltmandsgruppe ikke har haft faste makkere at sparre og brainstorme med. Derfor har jeg sporadisk brugt mine kollegaer i TECs praktikcenter, samt instruktører og SPS-vejledere, som sparringspartnere samt inspirationskilder.

Derudover har jeg haft to ugentlige aftaler med vejlederen, John Storm Ellehammer, på TEC i et forsøg på at holde mig på rette kurs, samt sørge for at jeg holder en støt udvikling i projektet, og ikke får stirret mig blind på enkelte dele/problemstillinger som kunne ende med at stjæle for meget tid.

5.1 Projektstyring

Grundet størrelsen af gruppen og de få kommunikationskanaler forbundet hermed, har den helt store planlægning virket som "overkill". Jeg har derfor holdt mig meget op af KISS-princippet, som nævnt i produktrapportens afsnit 3.2.7, og taget tingene som de dukkede op.
Jeg startede med en grundidé som jeg skabte use-cases (se bilaget "MBS - Bilag - Use-Cases") og det tilhørende diagram (se figur 1 i produktrapportens afsnit 3.2.3) ud fra. Disse har været grundstenen for planlægningen af hele forløbet.

Use cases blev efterfølgende til wireframes, og sidenhen delt op i overordnede "epics" med hver deres "user stories" (tilsvarende de faktiske use cases), med deres respektive, individuelle "tasks". Til hver enkelt af disse tre kategorier tilknyttede jeg "tags" på DevOps Board'et (figur 11), således at jeg frit kan søge efter f.eks. "use case 9", og så få vist alle elementer af projektet i sin helhed som på nogen måde involverer funktioner der er relevant for at kunne succesfuldt udføre use case 9.



Figur 11.

Min tilgang i starten af selve udviklingsforløbet af applikationen var meget eksperimentel, da jeg endnu ikke var helt indforstået med Xamarin som framework. Men min naturlige

arbejdsgang og process mundede basalt set ud i vandfaldsmodellen, med mindre opgaver som skulle løses før den næste opgave kunne berøres og bearbejdes.
Grundet mit tidligere udførte grundarbejde med inddeling af epics, user stories og tasks, kom dette ganske naturligt.

6 Metodevalg og teknologi

6.1 Værktøjer og udviklingshjælpemidler

6.1.1 Visual Studio 2022

Visual Studio har været min foretrukne IDE, især grundet sine mange funktioner og udvidelsesmuligheder via NuGet Package Manager. Det er den IDE jeg føler mig absolut mest hjemme i, og som en (stadig lettere) nyudklækket programmør nyder jeg godt af den kraftfulde hjælp der er at hente fra IntelliSense. VS er også hjemme for Xamarin som jeg ganske tidligt besluttede mig for at bruge.
Visual Studio har også en indbygget Android Emulator som, selv sagt, har været et flittigt brugt værktøj under udviklingsforløbet.

6.1.2 SQL Server Management Studio

Siden jeg begyndte at arbejde med databaser (SQL Server) har SSMS været en integral del af min værktøjskasse. Den har jeg flittigt brugt til at verificere og indsætte data i MBS's tilhørende database, samt oprette brugeren og dens tilhørende login som API'et forbinder med.

6.1.3 GitHub.com (versionsstyring)

Under produktets udvikling har jeg benyttet mig af et Git versionsstyringssystem.
Som et soloprojekt får jeg ikke benyttet dette kraftige værktøj til fulde, men ikke desto mindre har det været en uundværlig del af projektet. Jeg har særligt benyttet mig af sikkerheden i at kunne oprette forskellige forgreninger ("branches") af projektet, når jeg har skulle lave nogle testfunktioner som potentielt kunne ødelægge/forstyrre allerede implementerede, funktionelle dele af projektet.
Den indbyggede funktionalitet i Visual Studio gør det nemt, sikkert og overskueligt.

6.1.4 Azure DevOps Boards (Kanban-board)

Azure DevOps har jeg benyttet mig af i mindre grad, men det har dog været med til at give et hurtigt overblik over hvilke dele af projektet der har været afsluttet, og hvilke der skulle arbejdes på som det næste.
Det har især været nyttigt til at holde styr på i hvilke funktioner/dele der har været bugs, samt trin til at "replicate" disse.

6.1.5 PostMan

PostMan har været et nyttigt værktøj ift. at fremme hurtigere iterationer af API'et, samt fejlfinding af både dette og omkring den emulerede Android enhed.
Som beskrevet i afsnit 8 var jeg plaget af udfordringer vedr. SSL og konfigurationen af Visual Studios IIS Express server ift. at forbinde fra den emulerede telefon, og her kunne jeg via PostMan se om serveren rent faktisk var oppe at køre med dens gemte collections af HttpRequest metoder. Var serveren nede kunne jeg også få detaljerede oplysninger herigennem og derved spore mig ind på fejlen.

6.1.6 Draw.io

Dette har været mit foretrukne værktøj til at tegne diverse diagrammer. Det har hjulpet med at give et større overblik over bl.a. database design (entity relations), use-cases og funktioners interageren med hinanden.

6.1.7 Microsoft Excel

Jeg benyttede mig af en skabelon[4] til at skabe mit Gantt diagram, som jeg har lænet mig op af ift at overholde min relativt stramme tidsplan på projektet.

6.1.8 Designmønster (MVVM)

Grundet mange erfaringer med at lave WPF programmer ud fra MVVM (Model-View-ViewModel) designmønstret, er det dette jeg har valgt at følge (desuden læner Xamarin sig også meget op af dette). Dets uafhængighed mellem brugergrænseflade (View) og logik (ViewModel) giver i teorien god mening for mig (skønt der findes mange meninger/fortolkninger af korrekt implementering af MVVM), og gør at der i fremtiden kan videreudvikles på de individuelle dele uden at de allerede etablerede dele lider overlast og kompromiteres.

Jeg har lavet mange "krumspring" for at overholde MVVM designmønstret sådan som jeg forstår det, med en klar inddeling af kode som omhandler back-end "business logic", og front-end "UI logic". Alle referencer view og viewmodel imellem, sker udelukkende med referencer fra et view til en viewmodel, og aldrig den anden vej.
Det tætteste jeg er kommet på at "synde" i MVVM-regi, har været i code-behind af HistoryDayPopup hvor jeg udfører lidt business logic via en instans af HistoryViewModel. Som beskrevet i kommentarerne i koden synes jeg godt at kunne forsvare det da det trods alt omhandler forandringer i view'et (ændringer i DaysSource som udgør datasource for visualiseringen af registrerede dage på historiksiden).

Min implementering af MVVM er blevet pænere og mere optimeret desto nærmere slutningen af projektet jeg er nået, da jeg har lært meget undervejs.

[4] https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

F.eks. kan view'et MyStreamGraphView fungere selvstændigt som sit eget view med den tilhørende MyStreamGraphViewModel, og derved "unloades" fra hukommelsen hvis et andet element f.eks. skulle vises i MyStreamPage på dets plads i grid'et.
Dette står i kontrast til f.eks. de to CollectionViews og Editor elementerne på DayView, som bliver skiftet ud via deres respektive IsVisible property, og derved eksisterer i hukommelsen ligegyldig om det er synligt for brugeren eller ej.

6.2 Frameworks og libraries

I min tid som datateknikerelev på TEC har vi primært beskæftiget os med Microsoft produkter, hvorfor det er dem jeg er blevet mest bekendt med. Projektet bærer præg af dette og benytter sig næsten udelukkende af bærende frameworks og teknologier herfra.

6.2.1 MBS applikationen

Target Framework er .NET Standard 2.0.
Hertil er der installeret følgende libraries:

6.2.1.1 BCrypt.Net-Next

Dette library er baseret på "Blowfish encryption algorithm". Det anses for at være en god blanding af sikkerhed og hastighed. Den har som standard en funktion der først sammenkobler det input vi ønsker at "hashe" med en unik "salt" værdi, som så sidenhen hashes. Dette sikrer at to endeligt hashede værdier skabt af to identiske input strings, stadig vil være individuelle og derved øger sikkerheden ift. at modstå cyberangreb, f.eks. såkaldte "rainbow table attacks".

Koden til følgende eksempel er at finde i mit 'ExperimentsTester' projekt i 'SplashScreenTest02' solution'en på MBS GitHub repository som er linket til i indledningen.

```
Input password to verify.
kodeord
Generated salt         = $2a$11$458eIg.r0p8FonYRDn0/ne
HashPassword(PW, salt) = $2a$11$458eIg.r0p8FonYRDn0/nejA4oxBl3w7UXPu4ZjgjX8w1/n6ZbZNq
HashPassword(PW)       = $2a$11$ZKYzFRc4/9NFsMdkxmqyC.am5wxiY.eNXiIdwTUBm35LHfAf9BJNG
Stored hash-salt PW    = $2a$11$vh/zMzo0AIQa4shGfuQxHeEDCp0tu5MEd/fJ/93p2X5/xd5wtG5.W
Password is verified: False

Input password to verify.
testpass
Generated salt         = $2a$11$3z2Cdk1q0PlC6AOJcnrSUO
HashPassword(PW, salt) = $2a$11$3z2Cdk1q0PlC6AOJcnrSUOyavlZCEQyLSLUViT2CSKDWdN5EAS3.q
HashPassword(PW)       = $2a$11$AQaP9br8TEJrm31icAw5EO4CR/xFAwpI/lCeqftqKi4TkZa.iwN.m
Stored hash-salt PW    = $2a$11$vh/zMzo0AIQa4shGfuQxHeEDCp0tu5MEd/fJ/93p2X5/xd5wtG5.W
Password is verified: True
```

Figur 12.

Det korrekte kodeord i dette tilfælde er "testpass", som i anden paragraf i konsolvinduet (figur 12) bliver verificeret. Dette kodeord er her "pseudo-opbevaret" som en komplet hashet værdi i en string variabel, som kan ses på linjen ud fra "Stored hash-salt PW".

I første linje med lighedstegn af hver paragraf, ser vi først en udskrevet salt-værdi skabt af BCrypts GenerateSalt() metode, hver gang et input indtastes af brugeren.
På næste linje benyttes en overload af BCrypts hashing metode, HashPassword(string inputKey, string salt), som benytter det brugerdefinerede input samt den generede salt-værdi til at skabe en unik hash ud fra disse to værdier.
På næste linje benyttes samme metode, men blot med det brugerdefinerede input (BCrypt skaber automatisk en salt-værdi ud fra selv samme GenerateSalt() metode).
Det væsentlige at lægge mærke til her, er at skønt inputtet er 100% ens, er den outputtede hash-værdi komplet anderledes, da deres tilknyttede salt er anderledes.

Hvorfor bruge hashing frem for kryptering af brugernes følsomme data?
Meget kort fortalt foregår hashing én vej, mens kryptering foregår to veje. Hashet data kan ikke "dehashes", kun verificeres via sammenligning af to hashede værdier, mens krypteret data kan dekrypteres. Dekryptering kræver imidlertid en nøgle, hvilket ville skabe flere sikkerhedsmæssige problemer og bekymringer, eftersom denne skal opbevares og derved kan havne i hænderne på folk med tvivlsomme intentioner.

I MBS bruges hashing til sikker opbevaring og verificering af brugerens følsomme data.

6.2.1.2 Com.AirBnb.Xamarin.Forms.Lottie

Dette tillader brugen/afspilningen af Adobe After Effects animationer, gemt som json-værdier. Det bruges i MBS til at vise animationen i loadingskærmen, men vil senere også være brugbart ift. andre animationer.

6.2.1.3 Syncfusion.Xamarin.SfChart

Dette library gør at data kan præsenteres som diverse grafer, med mange forskellige valgmuligheder, f.eks. graftype, farver og "legends".

I MBS bruges det til at vise grafen på siden 'Min Å', bestemt ud fra de registrerede dages associerede "smileys", og er derved med til at give brugeren et overordnet billede af deres egenopfattede velbefindende.
I MyStreamGraphViewModel skaber jeg en ObservableCollection<ChartDataPoint> (ChartDataPoint værende en datatype fra samme library) ud fra samlingen af de relevante GraphDay objekter, hvor property'en Date bruges som datapoint til X-aksen og MoodID som Y-aksen. Denne collection bliver der så refereret til via databinding i <chart:SfChart />-tagget på MyStreamGraphView.

6.2.1.4 Rg.Plugins.Popup

Dette library lader os definere et allerede eksisterende view som et popupvindue.
I MBS bruges det til at redigere en dag når der klikkes på en dag på historiksiden. Det viste popupvindue er en ny instans af DayView, som også udgør view'et på "hovedsiden".

Når der trykkes på en dag i historiksiden, fyres en Command af som har den pågældende dag med som sit parameter. Jeg skaber et nyt HistoryDayPopup objekt med det førnævnte dag-objekt, samt det aktive HistoryPageViewModel objekt som parametre således at DaysSource ObservableCollection'en på HistoryPageViewModel kan redigeres gennem popupvinduet. HistoryDayPopup objektet benyttes så som parameter i library'ets PopupNavigation.PushAsync() metode.
En ny DayViewVM instans construct'es så med førnævnte dag som parameter, som bruges til at construct'e en ny DayView instans med denne ViewModel som parameter.
DayView instansen databind'es til som content i et ContentView-tag i HistoryDayPopup.

6.2.1.5 NETStandard.Library

Dette er en del af Target Framework'et. Det tillader kommunikation, og derved udvikling, over de forskellige former for .NET på flere platforme via delte API'er.

6.2.1.6 Newtonsoft.Json

Dette gør det muligt at parse og konvertere mellem en JSON string og .NET objekter.
JSON (JavaScript Object Notation) er et format bestående af "attribute-value" par samt arrays, typisk brugt til at sende og modtage data.
Her ses f.eks. et array sendt fra API'et, bestående af to JSON objekter baseret på MBSs GraphDay klasse, med tre attributes og deres tilhørende værdier.

```
[
    {
        "moodID": 1,
        "influenceID": 6,
        "date": "2022-04-26T08:47:04.6458404"
    },
    {
       (endnu et objekt...)
    }
]
```

Newtonsoft.Json kan så konvertere disse værdier til et .NET objekt som mobilapplikationen så kan "forstå" og bearbejde.

6.2.1.7 Xamarin.Essentials

Dette giver adgang til endnu flere delte API'er på tværs af platforme (Android, iOS, UWP). Specifikt for projektet her bruges det til at gemme værdier mellem sessioner gennem Preferences.Set() og Preferences.Get() funktionerne.

6.2.1.8 Xamarin.Forms

Forms er et UI framework, en samling af UI API'er og elementer, som tillader at skabe uniforme UI over de forskellige understøttede platforme, samt enheder med forskellige skærmstørrelser.

Dets Shell er især rigtig nyttigt og smart, da det giver adgang til fundamentale features som er alment brugte, og i visse tilfælde nødvendige, for moderne mobilapplikationer. Det giver bl.a. adgang til en delt navigationsoplevelse. Gennem et URI-baseret (Uniform Resource Identifier) scheme kan der navigeres til hvilken som helst side i applikationen.
I MBS er dette shell altid tilstedeværende gennem de tre knapper i bunden.

6.2.1.9 Xamarin.CommunityToolkit

Xamarin.CommunityToolkit fra Microsoft er en samling af genbrugelige elementer til udvikling af Xamarin projekter.
I MBS bruges det til validering af brugerens e-mail og kodeord, ved både login og oprettelse af en ny bruger.
Til validering af e-mail brugte jeg "EmailValidationBehavior"-tagget på Entry-elementet i LoginPage og CreateUserPage, som så tjekker om inputtet stemmer overens med formatet for e-mails (at det inderholder et snabel-a (@) og et punktum (.)).
Ligeledes brugte jeg "MultiValidationBehavior"-tagget på samtlige Entry-elementer for kodeord, som via "TextValidationBehavior"-tagget tjekker for om inputtet er minimum 8-tegn, samt via "CharactersValidationBehavior"-tagget for om der er minimum 1 stort bogstav, og 1 tal.
Er inputtet ikke dømt "valid" sættes en style fra ResourceDictionary i App.xaml på elementet som gør teksten rød, og ligeledes grønt hvis det dømmes som "valid".

6.2.2 MBS REST API

Target Framework er .NET 5.0.
Hertil er der installeret følgende libraries:

6.2.2.1 Microsoft.EntityFrameworkCore m.m.

EF Core er en ORM (Object Relational Mapper) som lader os "kortlægge" projektets .NET modeller, til databasens struktur/opbygning, og omvendt. Det har nogle fordele, især at undgå at skulle sammensætte og sende SQL statements frem og tilbage, samt automatisk forebyggelse mod SQL injections (en slags cyberangreb på databasen).
Metoden brugt under udviklingen af MBS har været "code-first approach", hvilket vil sige at .NET modellerne (klasserne Day, Influence, Mood osv.) dannede grundlag for opbygningen af tabellerne i databasen, samt deres relationer til hinanden, gennem "database migrations" skabt af vha. det støttende library 'Microsoft.EntityFrameworkCore.Design'. En valgfri pakke værktøjer, 'Microsoft.EntityFrameworkCore.Tools', lader os udføre disse migrations via kommandoer i package manager console'en. Endeligt bruges også en "database provider", Microsoft.EntityFrameworkCore.SqlServer, som gør kommunikationen mellem EF Core og SQL Serveren mulig, som parser og oversætter de modtagne værdier til T-SQL statements som tjenesten på serveren forstår.
Alt dette sikrer kompatibilitet mellem de to forskellige "verdener", altså at objekterne kan "kortlægges" som beskrevet ovenover.

7 Opmærksomhedspunkter fra produktrapport

7.1 Afvigelser

7.1.1 Aktører

7.1.1.1 Bruger

Det er desværre ikke muligt for brugeren i den nuværende iteration at hverken logge ud, ændre sit kodeord eller slette sin konto.
Som berørt i produktrapportens afsnit 3.2.4.2 ser jeg gerne en fremtidig udvidelse af siden 'Min Å' hvor brugeren har mulighed for at udføre disse handlinger.

7.1.1.2 "Den Indre Å"

"Den Indre Å" er helt fraværende fra den nuværende iteration af MBS.
Dette umuliggør Use Cases 10, 11 og 12.

7.1.2 Use Cases

7.1.2.1 Use Case 3

Undtagelsen for use case 3 er desværre ikke implementeret i den nuværende iteration. Der vises ingen meddelelse ved at forlade siden med en delvis registrering, og dataen bibeholdes når der navigeres væk fra siden.

7.1.2.2 Use Case 6

Trin 3 i beskrivelsen af use case 6 (bekræftelse af sletning af registreret dag) er desværre ikke implementeret i den nuværende iteration. Trykkes der på "SLET DAG" slettes dagen ud yderligere meddelelse herom.

7.1.2.3 Use Case 7

Undtagelsen for use case 7 er desværre ikke implementeret i den nuværende iteration. Der vises ingen meddelelse ved for få registereringer til at vise siden 'Min Å'.
I skrivende stund crasher programmet sågar hvis der ikke er tilstrækkelig data at vise.
Jeg håber at have løst dette problem inden eksamen.

7.1.2.4 Use Case 8

Det samme problem for use case 7 gør sig gældende her.

7.1.3 Krav til udviklingsforløbet

Gyldighed af e-mail og kodeord tjekkes ikke via "regular expressions".
Jeg benyttede mig i stedet af Xamarin.CommunityToolkit som har nogle funktioner specifikt til validering af indtastede input.

Til e-mail brugte jeg “EmailValidationBehavior”, og til kodeord “TextValidationBehavior” til at tjekke for minimumslængde af inputtet, samt “CharacterValidationBehavior” som tjekker for at der er minimum ét stort bogstav samt ét tal.

7.2 Kvalitetsfaktorer

Jeg vil her gennemgå hver kvalitetsfaktor som udpenslet i produktrapportens afsnit 3.5, som delkonklusion for om målet for kvaliteten af applikationen er nået.

7.2.1 ✔️ Pålidelighed

Mens applikationen kører i et udviklingsmiljø er det svært at konkludere noget endeligt på dette punkt.
API’et er udviklet til en, for nu, tilfredsstillende iteration med den påkrævede funktionalitet.
MBS applikationen er funktionel og stabil.
Der er ingen ukendte data-, retur- eller exceptiontyper som skulle kunne opstår og standse de to programmer. Derfor vil det være min vurdering at der intet er i vejen for at pålideligheden skulle kunne være ganske høj, og derved overholder faktoren, ved en faktisk deployment af produktet.

7.2.2 ✔️ Vedligeholdelsesvenlighed

Når først næste version af MBS er fuldført, med udbedring af de resterende bugs, samt de få opgaver vedr. “refactoring” og oprydning af visse dele af kodebasen, så vil der være meget lidt vedvarende vedligeholdelse af programmet, hvilket jeg vurderer møder kriteriet for faktoren.

7.2.3 ✔️ Udvidelsesvenlighed

Eftersom at dette punkt ikke havde høj prioritet, vil jeg mene at faktoren bestemt i produktrapporten er overholdt.
Dette vurderes ud fra bl.a. Xamarins AppShell funktionalitet hvor det visuelle aspekt nemt kan udvides med flere sider at navigere til med informationer, samt databasens opbygning af tabeller som vil kunne udvides med flere kolonner.
Kodens kompleksitet er ikke uoverkommelig høj, så der bør være plads til at kunne indsætte flere ønskede funktioner i fremtiden.

7.2.4 ✔️ Brugervenlighed

Det er min egen fornemmelse, samt på baggrund af enkelte personers udsagn som har set applikationen, at jeg vil vurdere denne faktor til at være overholdt.
Med tre primære sider, i alt seks individuelle skærmbilleder (hvor af de to går igen), 10 overordnede ting der kan trykkes på (hvor af fire går igen), samt blot tre tryk for at udfylde og gemme en dag, vil jeg mene at denne faktor lever op til sin høje vurdering i produkrapporten.

7.2.5 ✔️ Genbrugbarhed

Denne er for mig lidt svær at bedømme, ud fra faktoren på 25-30% genbrugt kode fastsat i produktrapporten.
Som nævnt i afsnit 7.1.4 ovenfor, går to af applikationens seks skærmbilleder igen (DayView), samt dennes tilhørende fire ting der kan trykkes på ud af en totalt på 10 overordnede ting.
Som en hurtig, løs beregning kan vi derved udlede at det genbrugte view udgør ca. 34% og de fire genbrugte knapper udgør 40% af applikationen. Ved at fratrække nogle arbitrære procenter for kode som er skrevet specifikt til genbrug af view'et (på historiksiden), må vi lande på omkring de 30% genbrugt kode.
Selvom dette på ingen måde er præcist, og blot et "guesstimate", vil jeg i henhold til faktorens lave prioritet kalde den overholdt.

7.2.6 ❌ Effektivitet

Via. Visual Studios Output vindue, kan vi se hvor lang tid det tager at render et vindue.
Indlæsning af "hovedsiden":

```
[Choreographer] Skipped 64 frames!  The application may be doing too much work on its main thread.
[ashscreentest0] Accessing hidden field Landroid/os/Trace;->TRACE_TAG_APP:J (light greylist, reflection)
[ashscreentest0] Accessing hidden method Landroid/os/Trace;->isTagEnabled(J)Z (light greylist, reflection)
[ashscreentest0] Accessing hidden method Landroid/os/Trace;->asyncTraceBegin(JLjava/lang/String;I)V (light
[ashscreentest0] Accessing hidden method Landroid/os/Trace;->asyncTraceEnd(JLjava/lang/String;I)V (light gr
[ashscreentest0] Accessing hidden method Landroid/os/Trace;->traceCounter(JLjava/lang/String;I)V (light gre
[ashscreentest0] Explicit concurrent copying GC freed 1105(96KB) AllocSpace objects, 0(0B) LOS objects, 49%
[Choreographer] Skipped 57 frames!  The application may be doing too much work on its main thread.
[EGL_emulation] eglMakeCurrent: 0x763eaca6d7a0: ver 2 0 (tinfo 0x763eacb785c0)
[OpenGLRenderer] Davey! duration=1073ms; Flags=0, IntendedVsync=609690777100941, Vsync=609691727100903, Old
[monodroid-assembly] open_from_bundles: failed to load assembly System.Collections.Immutable.dll
[monodroid-assembly] open_from_bundles: failed to load assembly System.Collections.Immutable.dll
[monodroid-assembly] open_from_bundles: failed to load assembly System.Runtime.CompilerServices.Unsafe.dll
[ashscreentest0] Explicit concurrent copying GC freed 2479(230KB) AllocSpace objects, 0(0B) LOS objects, 49
```

Figur 13.

Vi ser her at vinduet tager *lige* over et helt sekund at render, og overholder derved ikke kvalitetsfaktoren. Som beskrevet i produktrapportens afsnit 3.2.5.1 vil dette være et punkt at forbedre i en fremtidig version, hvor adskillige ting kommer til at ske i PreLaunch klassen frem for ifm. indlæsning af hovedsiden (f.eks. at ApiHelper og DataFiller klassen bliver statiske og derved ikke skal instantieres ved construction af DayViewVM).
Dette vil formentlig skabe marginalt længere ventetid på loadskærmen, som dog er udelukket for kravet om 1 sekund for visning af et skærmbillede.

Indlæsning af 'Min Å':

```
[Choreographer] Skipped 86 frames!  The application may be doing too much work on its main thread.
Loaded assembly: /data/data/com.companyname.splashscreentest02/files/.__override__/System.Drawing.Common.dll [External]
[OpenGLRenderer] Davey! duration=2148ms; Flags=0, IntendedVsync=610070291815965, Vsync=610071725149241, OldestInputEven
[Choreographer] Skipped 46 frames!  The application may be doing too much work on its main thread.
[OpenGLRenderer] Davey! duration=806ms; Flags=0, IntendedVsync=610071742255830, Vsync=610072508922466, OldestInputEvent
```

Figur 14.

Jeg vil tro at disse to værdier er for de to respektive view's (MyStreamGraphView, MyStreamInfluencesView) som udgør denne side. Det ville give mening at MyStreamGraphView er den første, og mest langsomme, da grafens datapoints først kan skabes når dataen bliver hentet fra API'et, og sidenhen omdannet til ChartDataPoint objekter.

Som beskrevet i produktrapportens afsnit 3.2.5.1 vil dette være et punkt at forbedre på i en fremtidig version, hvilket burde give ekstra optimering og hurtigere indlæsningstider.
Indlæsning af 'Historik':
Her dukker den samme OpenGLRenderer meddelelse ikke op, men efter min mavefornemmelse at bedømme er historiksiden den som indlæses hurtigst.

Som konklusion på kvalitetsfaktoren om effektivitet, er faktoren ikke overholdt.

7.2.7 ➗ Integritet

Denne kan jeg ikke konkludere noget på endnu, mens produktet kun findes i et udviklingsmiljø. Men RAID 1 opsætning er en valgmulighed hos langt størstedelen af, hvis ikke alle, serverudbydere, så overholdelsen af denne faktor vil kun være bestemt ud fra den endelige økonomiske overvejelse ifm. hosting af databasen.

8 Udfordringer og overkommelsen af disse

8.1 Software

8.1.1 Xamarin

Som beskrevet i afsnit 5.1 var jeg ingenlunde intimt bekendt med Xamarin da jeg påbegyndte projektet. Xamarin var valgt på baggrund af mine erfaringer med, og kendskab til, WPF (Windows Presentation Foundation), da jeg af flere kilder kunne forstå at disse to mindede rigtig meget om hinanden. Det gør de også, men der er så sandelig også store forskelle.
Min første store udfordring var således simpel navigation af siderne via Xamarin.Forms' Shell. Jeg skabte indtil flere test projekter med for at komme nærmere på at forstå navigationen, som ved første øjekast virker vildledende simpel og ligetil, men alligevel kan snyde hvis man ikke har sit "navigations hierarki" (se eksempelvis linje 40+ i AppShell.xaml for MBSs hierarki) i mente når man skriver adressen på siden der skal navigeres til.

Jeg kæmpede ligeledes over nogle dage (se projektlog d. 02/03-22 og fremad) med at få skabt en ordentlig implementering af MVVM mønstret ifm. at lave MyStreamPage, hvor jeg i de respektive siders XAML namespace deklaration, deklarerede min overordnede MyStreamViewModel hvori de "mindre" viewmodels (MyStreamGraphViewModel og MyStreamInfluencesViewModel) er instantieret og initialiseret som properties.
Dette resulterede i adskillige instanser af MyStreamViewModel, med MyStreamGraphViewModel instantieret af en standard constructor uden parameter. Da jeg skulle bruge en ObservableCollection givet med som parameter ved initialisering af viewmodel klassen, gav dette en vedvarende 'System.Reflection.TargetInvocationException' som var en stopklods for udviklingen.

Vha. flittig og strategisk benyttelse af Debug.WriteLine() samt en simpel integer variabel til at tælle hvor mange gange klassens constructor var blevet kaldt, fandt jeg endelig frem til min fejl (de mange instanser), og fik lavet en bedre implementering.
Jeg deklarerede de to "subviews" (MyStreamGraphView og MyStreamInfluencesView) i MyStreamPage.xaml namespace, samt MyStreamViewModel, som resourcer. Via. de to views 'x:key' property kunne jeg så angive dem som content for de to ContentView elementer på MyStreamPage.
Eftersom MyStreamViewModel også var deklareret som resource, kunne jeg i code-behind angive de to views bindingcontext til at være deres respektive viewmodels, instantieret som properties i MyStreamViewModel.

8.1.2 LINQ (Language Integrated Query)

Jeg har benyttet ganske simple LINQ statements (metode syntaks) flere steder i mit projekt. Men desværre blev udfordringen for stor der hvor jeg syntes at have mest brug for det. MyStreamInfluenceViewModel skulle have indholdt en ObservableCollection<Influence> med de største påvirkninger, associeret med hhv. positive og negative dage, hvilket jeg havde tænkt var en simpel opgave at finde frem til via LINQ.
I sidste ende valgte jeg en mere "lavpraktisk" løsning hvor jeg benyttede mig af to Dictionary<int, int> hvor 'key' repræsenterer InfluenceID, og 'value' værende antal forekomster af det pågældende ID i mit datasæt af GraphDay objekter.
Via. LINQ inddelte jeg min ObservableCollection<GraphDay> i tre IGrouping grupper, med deres key-værdi værende det tilsvarende MoodID.
Disse løbes så igennem en "foreach" løkke, hvori endnu en foreach løkke løber gennem de enkelte objekter i grupperne. Her tælles forekomsterne for hvert ID i hhv. en dictionary til "gode" og "skidte" dage. Sidst benyttes LINQs Aggregate metode, til at sammenligne value fra de to dictionaries og udvælge den største (hyppigst forekommende) værdi, som via et lambda udtryk finder det tilsvarende Influence objekt der returneres.

8.1.3 Scaling af visuelle elementer

Som noget af det første jeg lavede på projektet havde jeg vanskeligheder med at få DayView til at virke ordentligt.
På "hovedsiden" *ser* alt fint ud, men faktisk er de tre smiley'er som standard orienteret mod venstre, hvilket medfører at på en anden enhed med en anden skærmstørrelse/opløsning, vil disse ikke være centreret i view'et.
Dette ses når DayView bruges som content på HistoryDayPopup, hvilket står i kontrast til de to kolonner med påvirkninger som scale'es korrekt ligegyldig skærmstørrelse/opløsning, pga. at de vises i et Grid-element.
Grunden til dette er at finde i at jeg bruger et Frame xaml-element pga. dets CornerRadius property, som gør at jeg kan style'e mit MoodImage når det er blevet trykket på/valgt som en farvet cirkel frem for en firkant som akavet dækker hele det valgte billede.
Jeg forsøgte mig med forskellige løsningsforslag, men stødte på kompatabilitetsproblemer mellem de forskellige elementer med min implementering af GestureRecognizers (måden jeg kalder MoodClickedCommand) på de pågældende "parent" elementer.

Jeg tror løsningen er at finde i at konvertere koden til at bruge endnu et grid, hvorpå jeg skulle kunne implementere den nuværende DataTemplate uden større udfordringer.
Grunden til at jeg ikke gjorde dette ved udviklingen af denne del af MBS, var fordi jeg ganske simpelt havde brugt for lang tid på det og måtte komme videre med de øvrige dele.

8.2 Hardware

8.2.1 Projektet på flere klienter

Jeg måtte hente mit projekt fra GitHub og fejlfinde indstillinger, på forskellige computere et overraskende antal gange i løbet af projektet, hvilket har taget en ikke-ubetydelig mængde tid.
Som beskrevet i afsnit 5 er produktudviklingen næsten udelukkende foregået på TEC, hvor jeg har haft en fast arbejdsklient med masser af kræfter.
Den umiddelbare plan var at bruge min personlige bærbar til at fremlægge projektet til eksamen, men den viste sig desværre at have for mange år på bagen, og slet ikke nok RAM (4GB) til at køre både SQL Server, to instanser af Visual Studio til hhv. MBSAPITest01 og SplashScreenTest02, samt Android Emulatoren.
Jeg fik lånt en bærbar på TEC til forløbet, men det samme problem gjorde sig gældende (4GB RAM).
Den næste bærbar jeg lånte havde 16GB RAM og kunne derfor godt løfte opgaven, men med en HDD og ikke en SSD, tager det stadig lang tid at starte selve computeren og programmer m.m.
Og sidst men ikke mindst, har jeg naturligvis også haft alle produktets dele installeret på min egen, personlige klient derhjemme.

8.3 Graverende fejl i skrivende stund

8.3.1 Ved manglende kontakt til API

Starter programmet uden at der findes en tidligere bruger som har været logget ind, og der derfor ikke er gemt hverken UserID, eller Moods og Influences via.
Xamarin.Essentials.Preferences, hænger programmet og giver i sidste ende en ‘System.AggregateException’ grundet tasks i PreLaunch klassen som bliver canceled, hvilket ikke giver en fejlmeddelelse i programmet.
Findes de tidligere nævnte data i Xamarin.Essentials.Preferences, hænger programmet og crasher sidenhen når der forsøges at tilgå data gennem siderne ‘Historik’ eller ‘Min Å’.
I fremtiden ønsker jeg at overkomme dette ifm. renskrivningen af Prelaunch, ApiHelper og DataFiller klasserne som beskrevet i produktrapportens afsnit 3.2.5.1.
Det bør være et overkommelig problem ved at få programmet til at navigere til en fejlside i stedet for MainPage skulle dette opstå.

8.3.2 ‘Min Å’ ved færre end syv registrerede dage

Forsøger brugeren at navigere til siden ‘Min Å’ med færre end syv tidligere registrerede dage, sker en ‘System.Reflection.TargetInvocationException’ under udførsel af metoden FillGraphDays() i DataFiller klassen.
Til test formål kan dette omgås ved at generere dummydata, hvilket kan gøres via CreateDummyData(int UserID, DateTime startDate) metoden i Program.cs-filen i ExperimentsTester-projektet.
NB: Ved brug af ExperimentsTester-projektet, husk da at ændre adressen som HttpClient forbinder til i ApiHelper klassen (udkommenter linje 17 og brug linje 18 i stedet), således at der forbindes til localhost adressen, og ikke 10.0.2.2 (som er adressen Android Emulator bruger til at forbinde til host-maskinen).
I fremtiden ønsker jeg at overkomme dette problem ved indlæsning af et alternativt view for ‘Min Å’, ved kommunikationsfejl til API eller for få registrerede dage.

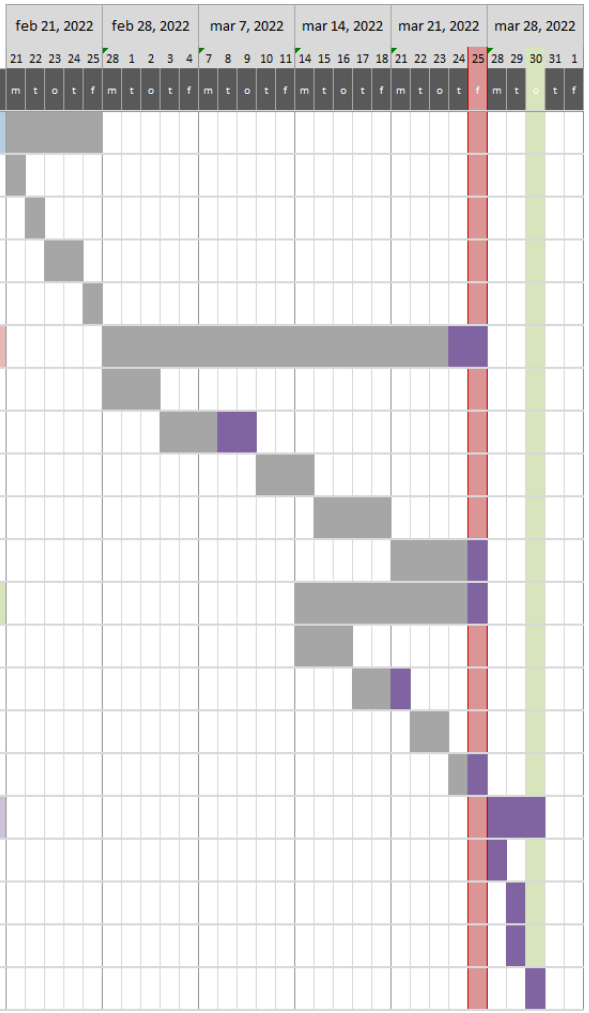
9 Projektlog

Projektloggen er at finde som bilag kaldet “MBS - Bilag - Projektlog”.
Bemærk at det er en log over *hele* udviklingsforløbet af MBS.
Loggen over svendeprøveforløbet begynder d. 21/02-22 på loggens side 9.

10 Realiseret tidsplan

Elvin Jensen — Display Week: 1

OPGAVE	FREMSKRIDT	START	SLUT	ARB. DAGE
Planlægning og diagrammer	100%	21-02-22	25-02-22	5
Casebeskrivelse	100%	21-02-22	21-02-22	1
Gantt Diagram	100%	22-02-22	22-02-22	1
Produkt "flow" (planlægning)	100%	23-02-22	24-02-22	2
Planlægning af opgavestyring (Azure DevOps)	100%	25-02-22	25-02-22	1
Produktudvikling	93%	28-02-22	25-03-22	19
Data indsamling og analyse (modeller, API)	100%	28-02-22	02-03-22	3
Graf tendens (view og viewmodel)	85%	03-03-22	09-03-22	5
Største påvirkninger (view og viewmodel)	100%	10-03-22	14-03-22	3
Min Å (samlet view og viewmodel)	93%	15-03-22	20-03-22	4
Debugging og opsamling af "det løse"	85%	21-03-22	25-03-22	4
Rapporter	98%	14-03-22	25-03-22	9
Projektplanlægning	100%	14-03-22	16-03-22	3
Metodevalg og teknologi	95%	17-03-22	21-03-22	3
Opmærksomhedspunkter fra produktrapport	100%	22-03-22	23-03-22	2
Konklusion	95%	24-03-22	25-03-22	1
Præsentationsforberedelser og eksamen	0%	28-03-22	30-03-22	3
Udvalg af opmærksomhedspunkter fra rapporter	0%	28-03-22	28-03-22	1
Udarbejdelse af præsentation	0%	29-03-22	29-03-22	1
Action cards til præsentation	0%	29-03-22	29-03-22	1
Eksamen	0%	30-03-22	30-03-22	1



Figur 15.

Sammenlignet med min oprindelige tidsestimering, som er at finde i produktrapportens afsnit 3.7, har jeg haft tilføjet et par enkelte punkter samt udvidet produktudviklingsperioden væsentligt.
Jeg har holdt mit estimat ganske pænt, men dog hoppet en del rundt i de forskellige dele af programmet undervejs og derved ikke overholdt de fastsatte dage per punkt under Produktudvikling.
Der mangler nogle procenter fra det endelige fremskridt i både kategorien Produktudvikling og Rapporter.
Punkt to i Produktudviklings (Graf tendens (view og viewmodel)) er grundet at jeg mangler lidt styling af grafen før jeg er helt tilfreds med den.
Debugging nåede jeg ganske simpelt ikke at blive helt færdig med.
Punkt to og fire i Rapporter stammer fra min hang til perfektionisme, og at noget altid kan blive bedre. Jeg kan imidlertid ikke sætte fingeren på noget konkret jeg skulle mangle.

11 Konklusion

Som en vis mand engang sagde, så overlever selv de bedst lagte planer sjældent mødet med virkeligheden. Især nu hvor projektet er afsluttet vil jeg imidlertid ikke vove at påstå at det var et godt eksempel på "bedst lagte planer", for i bagklogskabens klare lys ser jeg flere mangler ift. en mere detaljeret planlægning.
Derved ikke sagt at der ikke var mange ting som jeg har gjort rigtigt. Bl.a. brugen af Kanban-board har været en god hjælp til at holde et overblik med mine mange tags, skønt jeg nu ville ønske at jeg havde været mere konsekvent i min brug, særligt at løbende have krydset opgaver af, således at jeg kunne generere grafer og diagrammer ud fra denne data.

- Kan min løsning gennem brugerens egne dokumenterede data effektivt visualisere en tendens for brugerens succeser/nederlag?

Især dette punkt viste mig at min idé desværre var ret forfejlet helt fra dens begyndelse. Som jeg netop skrev i indledningen til denne rapport, så har vi som bevidste væsner tendens til at lade os rive med at vores negative oplevelser, og lade disse blive til fokuspunkt. Og som fjeren der blev til ti høns, kan én negativ oplevelse hurtigt komme til at overtage hele mindet om den pågældende dag. Så når en bruger (f.eks. om aftenen) vil registere sin dag, så er den behagelige, opmuntrende oplevelse om formiddagen allerede blevet erstattet af oplevelsen om den sure pædagog da barnet skulle hentes fra børnehaven.
Så min fejl var nok tofoldig:

1. At lade brugeren generalisere og registrere en hel dag, fremfor adskillige (positive) oplevelser over en dag.
2. Overhovedet at lade brugeren registrere negativt ladede oplevelser, da en graf med mere eller mindre udelukkende negative dage ikke fostrer håb om flere positive dage.

Men her vil 'Den Indre Å' kunne gøre en kæmpe forskel med sine påmindelser der fokuserer på de positive aspekter af registreringer, fremfor den nuværende iteration hvor brugerens egen opfattelse af sin egen data, faktisk kan være en potentiel hindring for det positive fokus.

- Kan UI/UX være med til at bidrage til en positiv oplevelse?

Da produktet aldrig kom ud til uvildige testere er dette umiddelbart svært at konkludere noget på. Men jeg synes selv at have opnået en brugergrænseflade som er behagelig at kigge på, med bløde, dæmpede farver og kanter, som henleder tankerne til vand og natur og derved en rolig sindstilstand.
Jeg har forsøgt at holde mig op af moderne "material design" principper, med dets simplicitet og få, gengående farver som gerne skulle danne et indtryk af et sammenhængende produkt. Gennem meget få interaktive, intuitive elementer gives der meget lidt anledning til frustration og undren, og enhver implementeret handling (use-case) kan fuldføres med ganske få tryk.

Som beskrevet i produktrapportens afsnit 3.2.5, har jeg en idé om en animeret å, som via en animation bliver større desto flere positive dage der er registreret. Med tanker om moderne overgange mellem skærme, ser jeg en bobbel af vand som løber ned over skærmen og "forsvinder" ind i navigationsknappen 'Min Å'. Når 'Min Å' siden åbnes fortsætter boblen sin færd op af skærmen til den store å som bliver større.

- Hvordan kan brugeren motiveres til at benytte app'en hver dag?

I den nuværende iteration føler jeg desværre ikke at der er særlig stor motivation for brugeren til at gøre brug af app'en. Alt afhænger af brugerens egen lyst til, og evne til at minde sig selv på, at benytte app'en.
Skønt jeg anser grafen som potentielt havende en mild motiverende faktor, er denne afhængig af at brugerne faktisk registrerer positive dage/oplevelser. Og som jeg også konkluderede i svaret på det første spørgsmål i dette afsnit, har dette muligvis været en fejl helt fra idéens begyndelse.
Som beskrevet i Produktrapportens afsnit '3.2.5 Programmets fremtid', har jeg planer om et intelligent notifikationssystem som ikke blot skal minde brugeren på sine positive oplevelser, men også lade disse være årsag til at brugeren har lyst til at registrere flere dage, samt naturligvis minde brugeren på at få lavet registreringer.