Daniel Skriver Hansen

H6PD100120

Procesrapport

Indholdsfortegnelse

[Titelblad 2](#_Toc56081780)

[Forord 3](#_Toc56081781)

[Læsevejledning 3](#_Toc56081782)

[Forklaring af termer brugt senere i rapporten 3](#_Toc56081783)

[Case 4](#_Toc56081784)

[Problemformulering 4](#_Toc56081785)

[Afgrænsning 4](#_Toc56081786)

[Estimeret Tidsplan & Projektstyring 5](#_Toc56081787)

[Versionsstyring 6](#_Toc56081788)

[Projekt Iterationer 7](#_Toc56081789)

[Første iteration 7](#_Toc56081790)

[Anden iteration 7](#_Toc56081791)

[Teknologier & Begrundelse 8](#_Toc56081792)

[Frontend 9](#_Toc56081793)

[Den valgte teknologi 9](#_Toc56081794)

[Alternative teknologier 9](#_Toc56081795)

[Backend 11](#_Toc56081796)

[Den valgte teknologi 11](#_Toc56081797)

[Alternative teknologier 11](#_Toc56081798)

[API 12](#_Toc56081799)

[Den valgte teknologi 12](#_Toc56081800)

[Alternative teknologier 12](#_Toc56081801)

[Konklusion 13](#_Toc56081802)

[**Flutter** 13](#_Toc56081803)

[**Firebase / Firestore** 14](#_Toc56081804)

[**Flask** 14](#_Toc56081805)

|  |
| --- |
| **Elev:**  Daniel Skriver Hansen |
| **Firma:**  Bang & Olufsen A/S |
| **Projekt:**  Bestillings Organiserings System |
| **Uddannelse:**  Datatekniker m. Speciale i Programmering |
| **Projektperiode:**  26/10/2020 – 26/11/2020 |
| **Afleveringsdato:**  18/11/2020 |
| **Fremlæggelsesdato:**  N/A |
| **Vejledere:**  Lars Thise Pedersen &  Lærke Brandhøj Kristensen |
|  |

# Titelblad

Tech College Aalborg,

Struervej 70,

9220 Aalborg Ø

# Forord

Denne procesrapport, som er en ud af to skrevet rapporter, udarbejdet i forbindelse med svendeprøveforløbet 2020, for datatekniker m. speciale i programmering.

Rapporten er skrevet og udarbejdet af Daniel Skriver Hansen.

# Læsevejledning

Projektet beskrevet herefter er opdelt i to dele, en proces- og en produktrapport. Denne læsevejledning dækker over procesrapporten og bør læses før produktrapporten, for ikke at misse nogle af de tanker og overvejelser som er gået ind i dette projekt.

Procesrapporten bør læses kronologisk for den bedste forståelse og læse oplevelse.

## Forklaring af termer brugt senere i rapporten

**CI:** Continuous Integration. Dette begreb går ud på, at man tager små dele af udviklingskode, uploader det til sin version styring, den holder så styr på, om man har skrevet koden rigtigt, ved at køre nogle test på denne kode.

**Memory Allocation:** Når man dykker ned i Memory Allocation, består det af, at når man allokerer noget ”Memory”, så reservere man en del af computerens fysiske hukommelse, nede i den ”RAM” (Random Access Memory)

**Garbage Collection:** Denne ”Garbage Collector” gør det modsatte af ”Memory Allocatoren”, den rydder op i RAM’ene, for at befri noget hukommelse som er optaget, men som ikke bruges længere.

**Frontend:** Dette er det som brugeren vil kunne se på sin smartphone el.lign. Det som brugeren vil interagere med, knapper og lign. Dette bliver også kaldet for ”UI”.

**Backend:** Alt det der sker ”behind the scenes”, altså at vores system snakker sammen og at den har mulighed for at tilgå en database.

**API:** Dette står for Application Programming Interface, hvilket vil betyde, at det er en service som vi selv kan lave og interagere med via hjemmesider el.lign. Dette kan så bruges til både at sende og modtage nogle data, af varierende arter.

# Case

I Danmark, og sågar hele verdenen, kan vi godt lide at være sociale mennesker og hygge sammen over en god middag. Dette sker oftest på restauranter, hvor man tager fint tøj på, og så finder en restaurant som man gerne vil ind i spise på. Når vi i disse tider, pga. corona, gerne skulle undgå så mange forskellige menneskelige interaktioner som muligt.

Under dette vil der blive lagt en case, som læseren skal forsøge at sætte sig ind i:

Du sidder på en restaurant, tjeneren kommer ned og spørger, hvad du gerne vil bestille. Tjeneren går tilbage til skranken for at sende din bestilling ind i systemet. Dette kan have 4 udfald.

* Din ordre kan være bestilt med nøjagtig det som du gerne ville have, og alt er fint.
* Din ordre kan være korrekt, men den tager rigtig lang tid om at blive færdig, så du vælger at gå op til skranken og afbestille din ordre og finde et andet sted at få dit måltid.
* Din ordre kan også være forkert sat ind i systemet, så du får noget som du ikke har bestilt.
* På vej tilbage til skranken kan tjeneren have taget en anden ordre på vejen, og så glemme at skrive din ordre ind i systemet, så du aldrig modtager dit måltid.

3 ud af disse 4 scenarier kan resultere i utilfredse kunder og tabt omsætning.

# Problemformulering

Hvordan får vi en mere struktureret måde, hvorpå man kan afgive sine ordrer på restauranter, minimere risikoen for at tjenere glemmer kunders ordrer og informere køkkenpersonalet om, når en ordre er gået over tid?

# Afgrænsning

Jeg er, i dette projekt, nød til at afgrænse mig til visse dele af det fulde billede, grundet en begrænset mængde tid. Afgrænsningen vil foregå på en sådan måde, at jeg vil beskrive det fulde projekt, men kunne dykke ned i visse dele af det, og få disse dele implementeret og testet.

Projektet er udviklet med et *proof-of-concept* i tankerne. En videre udvikling af dette projekt, med en større tidshorisont, ville kunne give en stor etablering i restaurationsbranchen.

# Estimeret Tidsplan & Projektstyring

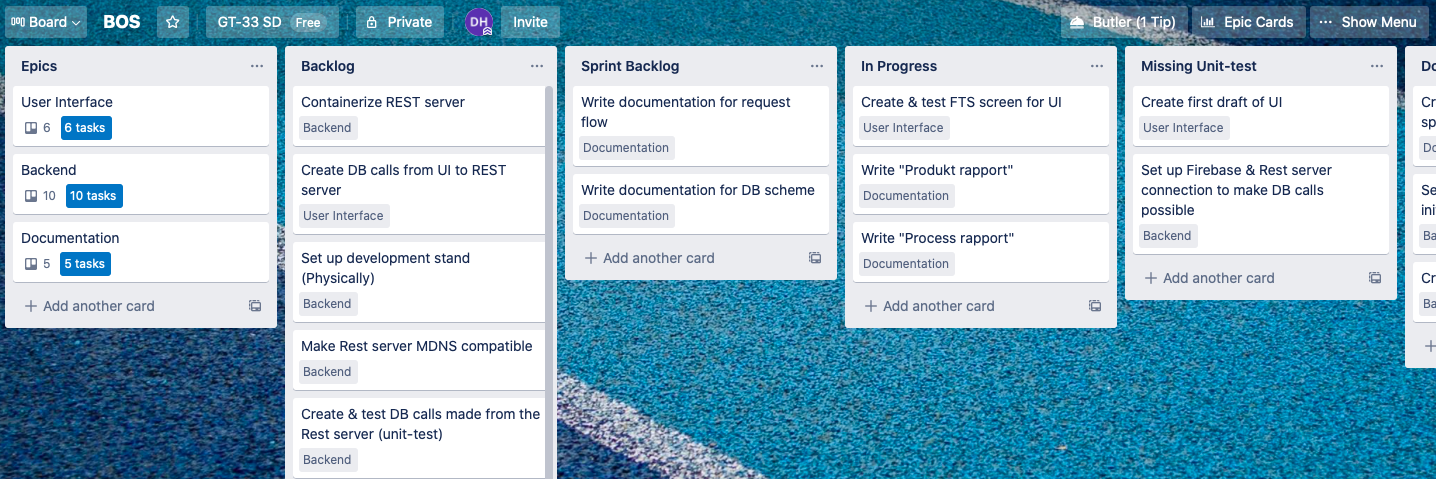
Jeg har udviklet en tidsplan for dette forløb, for at kunne holde overblik over den korte til, som er givet til rådighed. Tidsplanen er med til at holde styr på de forskellige opgaver som jeg har sat mig for hånden. Tidsplanen indeholder dog kun de overordnede emner, som er i dette projekt.

Et billede, der indeholder grøn, monitor, sidder, gade

Automatisk genereret beskrivelseTidsplanen vil også give et overblik over, hvilke opgaver der skal løses først.

I samme omgang har jeg valgt at bruge et værktøj som kan holde styr på mine opgaver, store som små. Jeg har valgt at bruge værktøjet **Trello**.

Jeg har valgt at bruge Epic’s til at kunne styre mine overordnede emner, og ud fra dette lave en masse små opgaver, sådanne at jeg bryder de større opgaver ned i mindre, for at kunne få en bedre gennemsigtighed med, hvad der skal laves, hvad der er blevet lavet og hvad der mangler at blive testet.



## Versionsstyring

Jeg har til min versionsstyring valgt at gøre brug af **Github**. Git er verdens førende inden for ”Source-Code-Management”, hvilket betyder at den holder styr på folks kode. Github giver muligheden for at have et decentraliseret sted, hvor alt kode kan ligge, så man altid få fat i det, så længe man har en internetforbindelse. Der er mulighed for at gå tilbage i kode historien og finde ud af, hvad der blev lavet på de bestemte tidspunkter. Selve Github har nogle meget smarte funktioner, som består i, at man kan lave lokale branches, altså tage den sourcekode som ligger oppe i skyen og lave lokale ændringer, uden at det påvirker den kode som ligger på serveren. Man kan derefter committe til sin branch, sådan at branchen kommer op i skyen, med de ændringer som er blevet lavet, men uden at ændre på det som andre udviklere arbejder på. Man vil derefter kunne lave en Pull-request og få andre udviklere til at kigge koden igennem, sådan at man har flere øjne på samme problem / løsning. Hvis det godkendes, så kan man få det ind i kodebasen.

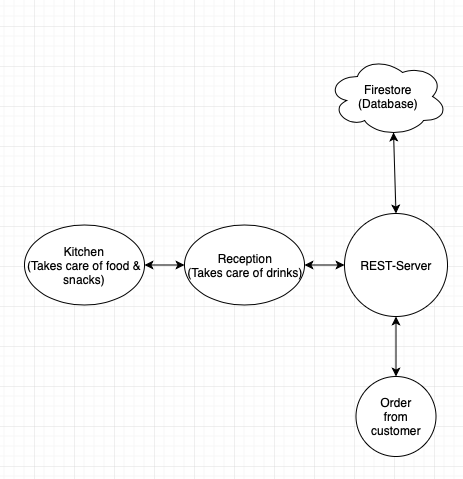
Github’ globale markedsandel ligger på 88.63%1 ifølge Slintel, hvilket gør dem til det største ”Source-Code-Management” system i verdenen.

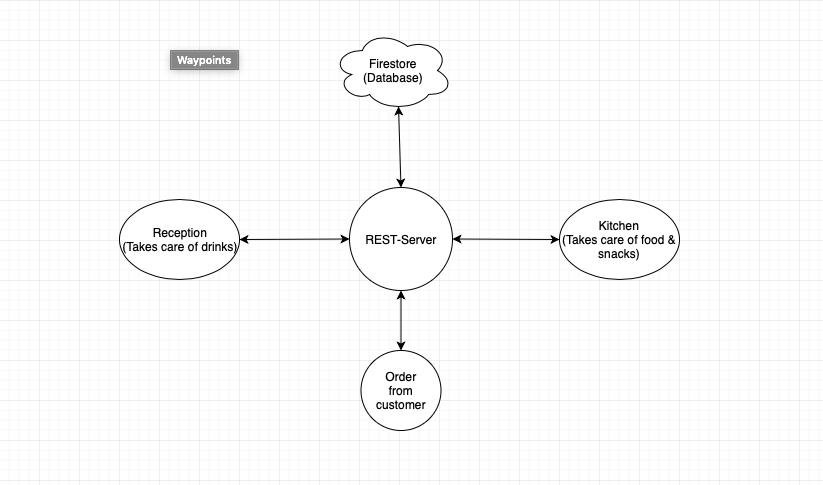
1 <https://www.slintel.com/tech/source-code-management/github-market-share>

# Projekt Iterationer

I dette afsnit vil du kunne læse omkring, hvordan projektet har udviklet sig under vejs, hvilke overvejelser der er blevet lavet ang. Workflow igennem koden, hvordan afgrænsningen er blevet ændret, flere gange og hvordan det ender med at blive.

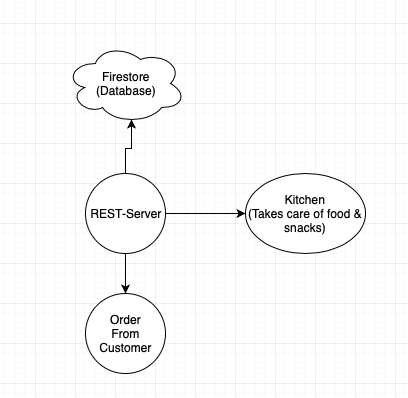
## Første iteration

Det startede med at jeg ville have lavet hele det store projekt, hvor jeg havde 2 idéer omkring, hvordan workflowet kunne være, dette er illustreret i billederne herunder:



## Anden iteration

Efterhånden som vi når længere hen i forløbet og bliver mere presset på tid, ender det med at jeg må lave en yderligere afgrænsning, hvilket ender med at workflowet kommer til at se sådan ud:



# Teknologier & Begrundelse

I denne sektion vil jeg komme ind over, hvilke teknologier jeg har brugt, begrundelserne bag disse, pros & cons og nogle alternative teknologier, som kunne være blevet brugt i stedet for. Dette er gjort for, at læseren kan få en bedre forståelse for de valg som er truffet i de forberedende tider samt hvilke ændringer der er sket under forløbet. Afsnittet vil blive sluttet af med en konklusion vedrørende de valgte teknologier kontra de alternative.

Jeg har valgt at inddele denne sektion i 3 undersektioner: Frontend, Backend & Api.

Når du læser igennem denne sektion, vil det være inddelt i ”bullet-points”. Måden som disse skal læses på, er som følgende:

* Navn på teknologi
  + Kort beskrivelse / baggrund for teknologien
  + Pros
  + Cons

## Frontend

### Den valgte teknologi

* **Flutter**
  + Flutter er Google’s UI Toolkit, som bruges til at bygge ”Native-Looking”, cross-platform applikationer. Google begyndte på at udvikle dette tool-kit, oven på deres eget udviklede programmeringssprog, Dart, tilbage i 2015 og var i en Beta version indtil sent 20181, hvor det blev udgivet. Flutter har gennem hele udviklingsforløbet været Open-Source, så man har kunne følge med i udviklingen og komme med sine egne forslag og hjælpe med at forbedre koden.  
    Blandt ~20.000 udviklere som blev spurgt om, hvilket sprog de bruger, til cross-platform er flutter steget med 9% fra 2019 til 20202.  
    1<https://www.futuremind.com/blog/pros-cons-flutter-mobile-development>  
    2<https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/>
  + **Pros**
    - Hurtig udviklingstid med dens J-I-T compiling og mulighed for Hot-Reload.
    - Release compiling sker A-O-T, så der kan skabes native-like apps.
    - Super godt til MVP (Minimum Viable Product)
    - Det er udviklet og vedligeholdt af en tech-gigant. (Google)
  + **Cons**
    - Apps kan fylde meget, da den indeholder koden til flere slags styresystemer.
    - Det er nyt, så alle features er ikke på plads endnu.
    - De store CI-systemer har ikke fået integreret Flutter endnu, dog er der et lille projekt (Codemagic.io) som kan lave CI til Flutter.

### Alternative teknologier

* **Swift**
  + Udviklingen af Swift startede tilbage 2010, men først i 2014 til Apples World Wide Developer Conference (WWDC), blive den første app, skrevet i Swift, vist frem. Ifølge TIOBE index’et for november 2020, så bliver swift kun brugt af 1.35% af de udviklere som er blevet spurgt, hvilket sprog de bruger mest1.  
    1 https://www.tiobe.com/tiobe-index/
  + **Pros**
    - Da swift er lavet til en native platform (IOS) har det nogle features som kan hjælpe med at reducere Memory leaks & usage.
    - Appen som bliver skrevet, vil meget nemmere kunne integreres med det Hardware som er på devicet.
    - Let skalerbart.
  + **Cons**
    - Det er ikke begynder venligt.
    - Der er dårlig bagud kompatibilitet.
    - Svært at integrerer i andre former end Apple produkter.
* **Xamarin**
  + Xamarin startede som et projekt, tilbage i 2000, som et forsøg på at lave .NET om, sådan at det kunne bruges på Linux platformen, det var dengang kendt som Mono1. Det blev senere hen opkøbt af Microsoft, men ikke før navnet blev ændret til Xamarin. Det er nu blevet så veludviklet, at det virker til et cross-platform udviklingsmiljø, hvor man kan skrive en kodebase til både IOS, Android og Windows (.NET)2  
    1 <https://www.oreilly.com/library/view/mobile-devops/9781788296243/72473bdb-ac6f-457f-9f10-63d252f63c9f.xhtml>2 <https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin>
  + **Pros**
    - Det har indbygget Memory Allocation & Garbage Collection.
    - Det er skrevet med sproget C#, hvilket er et veludviklet programmeringssprog.
  + **Cons**
    - Xamarin understøtter kun dets eget bibliotek af komponenter, hvis man vil have andre former for komponenter, er man nødsaget til selv af lave dem.
    - Xamarin apps fylder som regel mere end en native app, grundet at den skal have hele appen – både IOS & Android – men også alle de biblioteker som er taget med ind.

## Backend

### Den valgte teknologi

* **Google Cloud Firestore / Firebase**
  + Firebase / Firestore startede som en uafhængig virksomhed tilbage i 2011, i 2014 blev det opkøbt af Google og kan nu bruges af alle som opretter en konto. Firebase er det overordnede navn for hele servicen, som kan tilbyde både realtime database, normal storage, hosting og deres Firestore service, som jeg benytter mig af.
  + **Pros**
    - Det er gratis for begyndere.
    - Både Firestore og Realtime Database er cloud hosted og NoSQL
    - Det er vel dokumenteret & har en stor ”Community”
  + **Cons**
    - Selve Firebase er Android centreret, hvilket vil sige at det har en bedre integration med Android produkter i forhold til IOS produkter.
    - Der er ikke de samme Query muligheder som der vil være i en SQL-database, da det hele er lagret som JSON.

### Alternative teknologier

* **MsSQL**
  + Det startede med den først SQL server version, tilbage i 1989, som var en 16-bit server til OS/2 styresystemet. Dette blev gjort for at kunne portere Sybase SQL Server til OS/2 systemet. Udvikling fortsætter stadig den dag i dag, hvor hele kodebasen er blevet skrevet om fra C til C++. I 2017 fik de dog lavet en stor milepæl1 for hele udviklings Communitiet, som var at kunne køre en SQL-Server i en docker container, så det var nemt at holde styr på sine servere, samt at oprette nye.  
    1 <https://en.wikipedia.org/wiki/Microsoft_SQL_Server>
  + **Pros**
    - Det har indbygget ”Data recovery support”
    - Det virker out of the box, hvilket vil sige, at man ikke behøver at installere flere ting end selve MsSQL
  + **Cons**
    - Hvis man skal bruge en enterprise version a MsSQL koster det en væsentlig del penge.
    - Det er begrænset af, hvilke Hardware komponenter du har i din server, da det afhænger meget af hastighed.

## API

### Den valgte teknologi

* **Flask**
  + Flask bliver beskrevet som et ”Micro Web Framework”, hvilket betyder at man ikke er nødsaget specifikke værktøjer og / eller biblioteker. Alt hvad man skal bruge til Flask kommer med den biblioteket man installere via pip (pythons package manager). Dette framework baserer sig på et underliggende bibliotek, som hedder Werkzeug, hvilket er Flask’ utility bibliotek, som gør at man kan få forbindelse via http standarden. Flask kan bruges på mange forskellige måder, den primære funktion er at fungere som en web-server. Barebones kan man lave et REST-API, den måde jeg har valgt at bruge det på, men man kan også lave fulde hjemmesider ved at få API’et til at loade HTML (hjemmesidekode) ind. Hvis man f.eks skal bruge en database, skal man selv lave funktionaliteten til at kunne hente / give data til databasen.
  + **Pros**
    - Skrevet i Python, hvilket er et let begyndersprog, så de fleste programmører vil nemt kunne komme ind i en verden af Flask.
    - Det er meget fleksibelt, baseret på, at det kan håndtere de fleste opgaver som man vil have det til, hvis man kan skrive det i python, kan man også få det til at virke med Flask.
    - Der er gode muligheder for testing af Flask, via Unit-test & Integration-Test.
  + **Cons**
    - Det er ”Single Threaded”, hvilket betyder at den tager alle forespørgsler i kronologisk orden, som de kommer ind.
    - I takt med den overstående, så er det ikke så specielt godt til større applikationer.

### Alternative teknologier

* **Django**
  + ”Django is a high-level python Web framework that encourages rapid development and clean, pragmatic design. Buildt by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvet the wheel. It’s free and open source.”1  
    1 <https://www.djangoproject.com>
  + **Pros**
    - Har et nemt naviger bart admin UI, hvilket giver mulighed for at gøre visse ting nemmere igennem en brugerflade, så alt ikke skal laves i kode.
    - Det har et buildt-in bootstrap (Hjemmeside styling) tool.
    - Django er bedre til at håndtere ”Multi-Application” scenarier, som betyder at man kan dele sin kodebase op i flere forskellige applikationer, og så ville Django stadig kunne finde ud af det.
  + **Cons**
    - Det er et svært framework at sætte sig ind i, da der er så mange forskellige muligheder at få sit program til at virke på.
    - Det kan ikke håndtere flere forespørgsler på samme tid.
    - Det er ikke så godt til mindre projekter, da det tager lang tid at få en struktur op at køre.

## Konklusion

Under denne sektion, vil jeg beskrive mine valg af teknologier yderligere, hvorfor jeg valgte dem og hvorfor jeg valgte de andre fra.

Jeg har primært fokuseret på at finde nogle teknologier, som ikke krævede de helt store miljø opsætninger. Flutter downloades bare fra hjemmesiden og smides ind i din computers PATH, som gør at man kan køre flutter uanset, hvor man står henne i sin terminal. På Firebase opretter man bare en bruger via sin google konto, opretter sin database, henter en pakke til Python (Firebase\_admin) og så er man kørende med det. Eftersom at jeg udvikler på en Macbook Air i øjeblikket, skulle jeg ikke tænke over at installere Python, da det allerede er en del af styresystemet, så skulle jeg kun lige hente pakken Flask, via Pip, og så var jeg også kørende med mit REST-api.

### **Flutter**

Da jeg gennemsøgte teknologier, som skulle bruges til min frontend, gik valget på Flutter frameworket. Dette er begrundet med at jeg skulle bruge noget uden den store opsætning, det skulle være hurtigt at komme i gang med, veldokumenteret og have en store nok community til at kunne finde svar, hvis jeg eventuelt kom ind i problem, som jeg ikke selv kunne løse. Xamarin kunne også være en kandidat, men eftersom at det er skrevet i C# og kræver en compiler, ville dette ikke være hurtigt nok. Med Flutters Hot-Reload mekanisme gier dette en super hurtig måde, hvorpå at man kan se og teste sine kodeændringer.

### **Firebase / Firestore**

Når det kom til valg af database, tænkte jeg at lægge ansvaret lidt væk fra mig. Jeg valgte Firestore af den grund, at jeg ikke selv skal sørge for at den er oppe at køre, det har jeg et stort firma som står bag. Jeg skal selvfølgelig tjekke om databasen er oppe og køre, via min kode, men det er ikke min opgave at den kører inde ved dem. Dette giver mig samtidig mulighed for at bruge andres API / python bibliotek, som man forventer er gennemtestet og veludviklet.

### **Flask**

Hele essensen af mit projekt omhandler at komme igennem en REST-server. Eftersom at mit projekt ikke er verdens største skulle jeg have noget der ikke fylder alverdens, det skulle være fleksibelt, eftersom at der skal være adgang til en database og så skulle det være i et veludviklet og veltestet sprog. Brugen af Flask var præcis det jeg skulle bruge. Det eneste minus ved flask, i forhold til min applikation, er at det kun har en tråd at køre med. Jeg kunne også godt have valgt Django, men størrelsen på mit projekt gik direkte i mod at skulle bruge det. Brugen af python til mit REST-Api, gør det nemt at bruge andres pakker til at få lavet min server, det vil også gøre det nemt at teste den f.eks via unit-test.

# Konklusion

Projektet er endt ud med at bliver mere afgrænset end ønsket. Bagud set, er det tydeligt, at jeg var for optimistisk i forhold til udviklingstid, indlæringskurver på nye teknologier samt problemløsningstid.

Undervejs har der været flere udfordringer ift. min manglende viden omkring opdatering af dvs. frameworks. Sidst jeg arbejdede med Flutter, f.eks., var der ikke så mange features som der er nu, der er også blevet lavet yderligere udvikling på nogle af de originale widgets. Nogle af disse opdateringer har givet flere problemer end forventet.

Trods dette kom der et fint produkt ud af udviklingen. Produktet er dog ikke fuldent endnu, dette vil ske inden rapporten afleveres.

For yderligere indblik i det endelige resultat, læs produkt rapporten.