|  |  |
| --- | --- |
| Roameo  Your personal travel planner | Procesrapport  Aarhus Tech  Jonas Wittstrøm  Svendeprøve – Datatekniker programmering |

Indhold

[Casebeskrivelse 2](#_Toc193397779)

[Problemformulering 2](#_Toc193397780)

[Metode 2](#_Toc193397781)

[Anvendte fagområder 2](#_Toc193397782)

[Anvendte teknologier 2](#_Toc193397783)

[Begrundelse af valg 3](#_Toc193397784)

[Estimeret tidsplan 4](#_Toc193397785)

[Estimeret flowchart 4](#_Toc193397786)

[Mockups 5](#_Toc193397787)

[Udviklingsprocessen 5](#_Toc193397788)

[.NET (MAUI og Web API) 6](#_Toc193397789)

[AWS 6](#_Toc193397790)

[Google Cloud APIs 6](#_Toc193397791)

[Brugerhåndtering 7](#_Toc193397792)

[Realiseret tidsplan 8](#_Toc193397793)

[Perspektivering 10](#_Toc193397794)

[Konklusion 10](#_Toc193397795)

[Kilder og referencer 10](#_Toc193397796)

[YouTube 10](#_Toc193397797)

[AI 10](#_Toc193397798)

[Grafik 10](#_Toc193397799)

[Bilag 11](#_Toc193397800)

[Bilag 1 – Mockups 11](#_Toc193397801)

# Casebeskrivelse

Formålet med denne app, er at hjælpe rejsende med at planlægge deres ture på en struktureret, overskuelig og visuel måde. Den kombinerer en dag-til-dag rejseplanlægger med integration af offentlig transport og brugerens personlige præferencer for mad og aktiviteter

# Problemformulering

Hvordan kan jeg udvikle en brugervenlig rejseplanlægningsapp, der kombinerer dag-til-dag planlægning, kortintegration, offentlig transport og personlige anbefalinger?

# Metode

## Anvendte fagområder

Jeg skriver her i punktform de fagområder jeg berører under mit projekt.

* Client-side programmering
* Server-side programmering
* Objektorienteret programmering (OOP)
* App-programmering
* Versionering
* Cloud-teknologi

Jeg føler at disse spænder bredt, og giver god mulighed for at vise hvad jeg kan inde for mange forskellige områder.

## Anvendte teknologier

Jeg skriver her i punktform de teknologier, som er brugt i mit projekt. Efterfølgende uddyber jeg hvorfor jeg har taget nogle af disse valg frem for andre, samt deres fordele og ulemper.

* Visual Studio Community 2022
* Github
  + Versionering
* ASP .NET
  + MAUI
  + Core Web API
* Amazon Web Services (AWS)
  + Lambda
  + API Gateway
  + DynamoDB (NoSQL)
  + AWS NoSQL Workbench
* Google Maps + APIs
  + Navigation
  + Geocoding
  + Places
* PlantUML
  + Som plugin i Obsidian (Notat-software)
  + Flowchart

### Begrundelse af valg

Valget af Visual Studio (VS22) var nemt, da det for det første er den IDE jeg har arbejdet med mest – både privat og på min læreplads. Udover dette, er det specielt bygget for .NET applikationer og tilbyder skabeloner for de fleste projekttyper, så man hurtigt kan komme i gang.   
Jeg kunne godt have lavet et tilsvarende projekt i Visual Studio Code (VSCode), som er mere letvægt (kræver færre ressourcer af din maskine) og mere fleksibelt på mange punkter.   
Hvis jeg lavede mit projekt i eksempelvis React Native frem for .NET, havde det givet meget bedre mening med VSCode.

Som nævnt ovenfor, valgte jeg .NET MAUI, da jeg i forvejen er mest komfortabel med .NET frameworket, som er det jeg har arbejdet mest med gennem hele uddannelsen – både i skole og på min læreplads. Denne løsning gør det også muligt at skrive apps til flere platforme på samme tid med en fælles kodebase, uden at jeg behøver at lære at skrive til Android eller Apple specifikt.   
Havde dette ikke været tilfældet, kunne en anden god løsning have været React Native, som er baseret på JavaScript, og har et stort fællesskab og et bredt udvalg af plugins og biblioteker. Dog kræver det også at man har noget dybere kendskab til de platforme man udvikler til, end hvad jeg har.

Grunden til at jeg valgte at gøre brug af AWS til mit projekt frem for Azure, som måske havde været det mere naturlige valg, var fordi jeg på hovedforløb 5 havde forsøgt netop dette, men fandt det meget svært at sætte op. Det endte derfor med at jeg ikke havde et produkt at vise efter flere ugers arbejde, fordi jeg sad meget fast.   
Azure ville dog give god mening, da det allerede er en integreret del af Microsoft, og på papiret skulle være meget nemt at sætte op til et projekt som mit.  
Desuden fandt jeg også opsætningsprocessen i AWS meget let, især deres UI var intuitivt at finde rundt i. Jeg følte derfor jeg kunne bruge mere tid og kræfter på udviklingen af mit projekt.

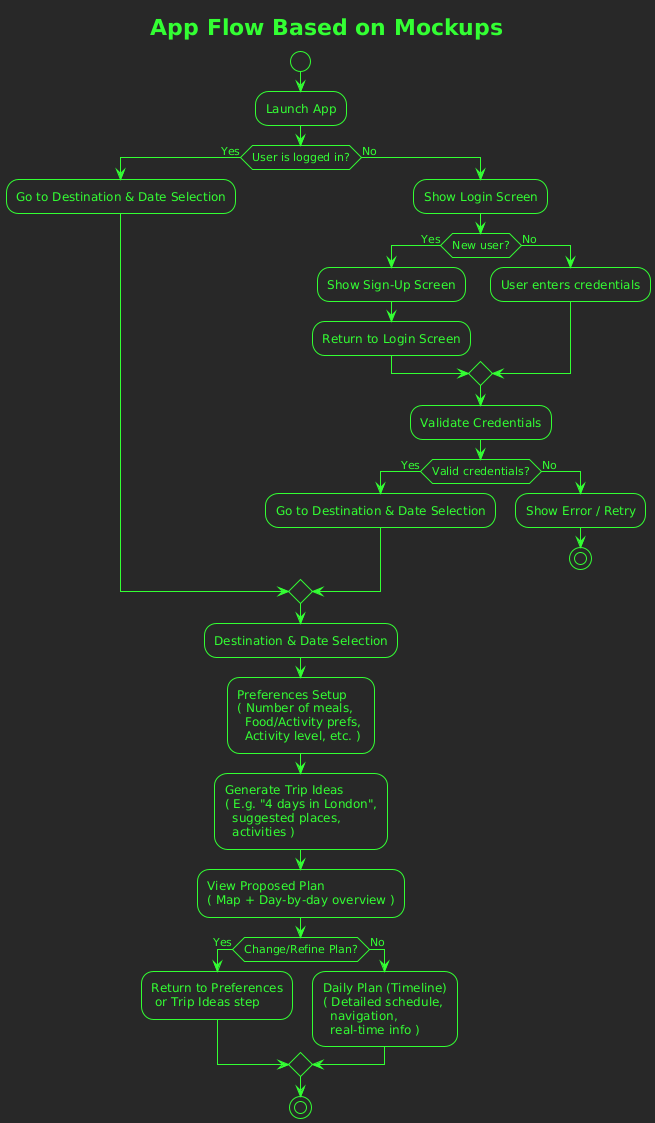
## Estimeret tidsplan



Ovenfor ses min estimerede tidsplan for projektet.

## Estimeret flowchart

Her vises et flowchart over, hvordan jeg havde tænkt mig at flowet skulle være, før jeg startede med at udvikle projektet.



## Mockups

I dette afsnit vil jeg gennemgå de mockups jeg fik lavet før jeg startede udviklingen. Disse ligger som bilag 1.

Mine mockups dækker ret godt over de funktioner, som også endte med at komme med i appen i sidste ende. Har bygget min app meget med udgangspunkt i disse.   
På tegningerne kan man se en login-side, hvor der også er gjort tanker om brugeroprettelse. Der ses et udkast til DestinationPickerPage, som er appens startside, hele vejen til siden for den detaljerede dagsplan med navigation, som er sidste side i min app.

# Udviklingsprocessen

I dette afsnit vil jeg skrive lidt om, hvad der gik godt og mindre godt under udviklingsprocessen. Jeg vil også sætte nogle ord på, hvordan jeg fik løst de problemer der opstod undervejs. Jeg vil komme forbi alle de mest centrale elementer i min app.

## .NET (MAUI og Web API)

Grunden til at jeg valgte MAUI, var en kombination af to ting: 1. Jeg ville udfordre mig selv med at lave en mobil app, efter mit fejlslagne projekt på hovedforløb 5. 2. Jeg er allerede godt bekendt med .NET, som nævnt tidligere.

Selve opsætningen gik nemt – Jeg valgte MAUI skabelonen i VS22 og på meget kort tid havde jeg nogle grundlæggende filer og mapper på plads.

Det første jeg gjorde, var at tilføje et Web API til mit projekt og begynde at bygge mine datamodeller og controllere.

Gennem hele processen, var dét at arbejde med .NET klart det der kørte mest flydende og med meget få problemer undervejs. De få der kom, indebar ofte at jeg manglede en NuGet pakke eller lign. Dette løses ret nemt gennem NuGet Packet Manageren i VS22.

Da jeg ikke har arbejdet meget med APIer var jeg lidt i tvivl om, hvordan jeg skulle bygge mine datamodeller og controllere. Dette fik jeg hjælp til ved google søgninger og AI.

Generelt kørte denne del af processen uden de helt store problemer.

## AWS

Til at hoste min app online, havde jeg tænkt at bruge AWS. De har et ret generøst Free Tier, så man hurtigt kan få sin app i luften uden omkostninger. I hvert fald ikke for et projekt så lille som mit.

Til selve hosting af appen brugte jeg Lambda, som en serverløs tjeneste, som kører kode uden behov for serveradministration. Denne aktiveres automatisk ved specifikke hændelser eller HTTP-kald.   
Da AWS ikke er en integreret del af Microsofts ”øko-system” som Azure, krævede det lidt mere opsætning. Til at få styr på denne del, fandt jeg en YouTube video, der på trods af at være indtalt af AI, gav en god guide til opsætningen.

Til at håndtere API kald brugte jeg API Gateway, som er en service, som gør det muligt at administrere og sikre mine APIer online og forbinde klienter med mine backend services.   
Opsætningen af denne foregik for det meste online, efter jeg havde bygget mine controllere i mit .NET projekt. Her skulle jeg opbygge min API til at spejle de endpoints og metoder jeg havde i mit projekt.

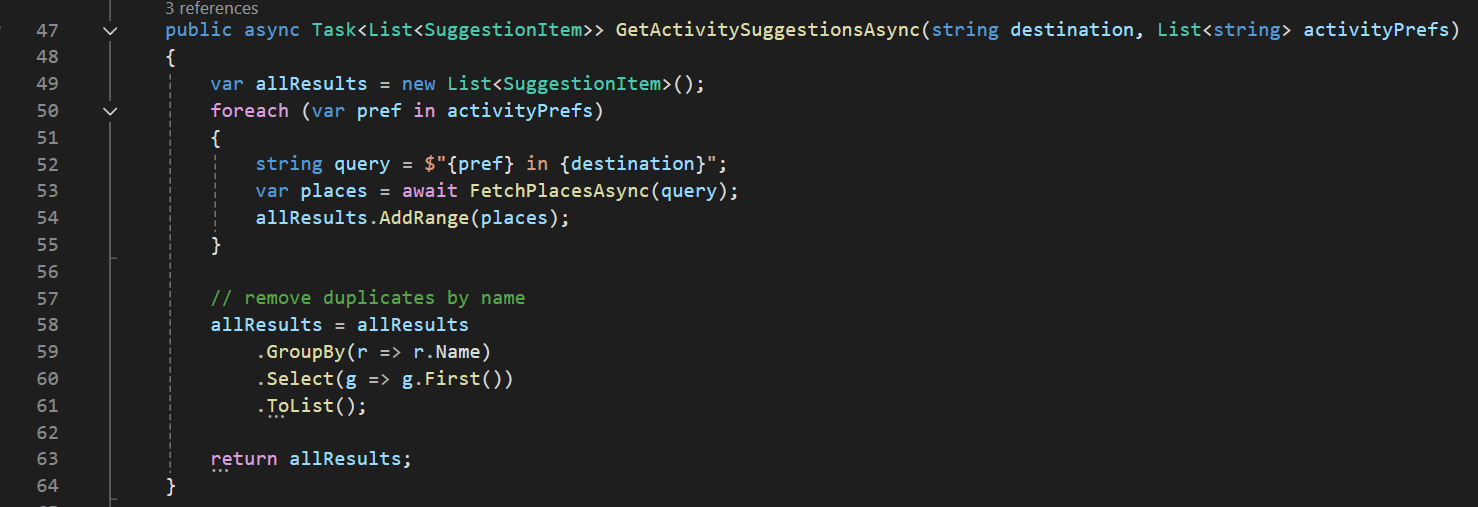
Som min database valgte jeg at gå med DynamoDB, som er en NoSQL database. NoSQL er en en type database, som gemmer data i fleksible strukturer, som f.eks. dokumenter eller nøgleværdier. Dette giver større fleksibilitet end traditionelle tabeller.  
Opsætningen af denne foregik gennem NoSQL Workbench, som er et stykke software AWS har lavet til netop dette formål.   
Da NoSQL er nyt for mig, skulle jeg også lige bruge lidt tid på at sætte mig ind i dette, men alt i alt gav det ikke de store problemer.

## Google Cloud APIs

Jeg vidste at jeg skulle bruge navigation og informationer om forskellige virksomheder til min app. Derfor føltes Google som det oplagte valg til netop dette.

Ligesom AWS har Google Cloud også et ret generøst Free Tier, der tillader brugen af de fleste af deres APIer i et ret stort omfang uden omkostninger.

For at komme i gang, skulle jeg bruge en API Key, som jeg fik kort efter jeg skrev mig op til deres services. Denne kunne jeg så bruge i mit projekt til at lave forespørgsler (queries) til den API jeg nu skulle hente informationer fra.   
I min app brugte jeg Geocoding API til at hente oplysninger om Byer/områder når jeg oprettede rejser fra min DestinationPickerPage.  
Senere brugte jeg den til at hente basale oplysninger om spisesteder og aktiviteter fra deres Places API.

Forespørgslerne til disse API’er er ret simple:  


I kodestykket her, viser jeg min GetActivitySuggestionsAsync-metode. Her henter jeg data omkring hvilke aktiviteter brugeren kan tage til inden for deres præferencer. På linje 52 vises et eksempel på hvordan en query til Googles API kan se ud. Der bliver bare sendt en forespørgsel omkring hvilke aktiviteter der kan findes i det valgte område.  
Derefter sorteres den, og dubletter fjernes, så brugeren ikke ser det samme resultat flere gange.

Generelt syntes jeg at brugen af Googles APIer var meget ligetil, og stødte ikke rigtig på nogle problemer.

## Brugerhåndtering

Jeg ønskede at have brugerhåndtering med i mit projekt, da det åbner op for en masse features, som for mig er er essentielle ved mange apps. Features som deling af rejser, tilføjelse af venner/rejsemakkere og måske opbyggelse af fælles planer sammen med andre.

Dette er et punkt jeg altid har kæmpet og haft meget svært ved. Det har fejlet for mig i tidligere projekter gennem hovedforløbne og også i en mindre opgave på min læreplads. Det skulle heller ikke være anderledes denne gang.

Jeg følte, at jeg gennem min research og grundige opsætning af mit projekt hos AWS havde et solidt fundament til at håndtere simpel brugerhåndtering. Men når det kom til det, viste det sig ikke at være tilfældet.  
Jeg kunne gennem fejlfinding se at jeg fik hentet nogle af de oplysninger jeg skulle bruge, men at mine kald i sidste ende slog fejl. Jeg brugte (alt for) lang tid på at gennemsøge nettet og spørge AI om hjælp, men kom ingen vegne. Efter jeg havde brugt det meste af en uge på at prøve at få det løst, måtte jeg til sidst give op og fortsætte uden. Ellers ville jeg slet ikke have et produkt at vise i sidste ende.

Al denne forsømte tid, har også resulteret i at resten af udviklingen af mit projekt er blevet lidt ”rushed”, og at jeg har måtte bruge meget af min weekend på at komme i mål med projektet.

Man kan derfor sige at jeg løste mit problem, ved ikke at løse det. Jeg måtte lægge mine prioriteter om til fordel for at have en funktionel app.

## Realiseret tidsplan

Nu vil jeg dykke ned i min realiserede tidsplan og hvordan den afviger fra den estimerede.

**Uge 1 (24-28 februar)**

|  |  |
| --- | --- |
| **Dag** | **Aktivitet** |
| Mandag | - Første udkast til problemformulering  - Logbog  - Første udkast til estimeret tidsplan  - Casebeskrivelse  - Udkast til MVP |
| Tirsdag | - Logbog  - Problemformulering (godkendt)  - Estimeret tidsplan (godkendt)  - Planlægning af anvendte teknologier  - Planlægning af anvendte fagområder  - Mockups i samarbejde med min hustru. |
| Onsdag | - Logbog  - Research af NoSQL  - Testprojekt med AWS Lambda, DynamoDB og API Gateway |
| Torsdag | - Mockups i samarbejde med min hustru.  - Datamodeller i NoSQL Workbench  - Opsætning af projekt hos AWS. |
| Fredag | - Datamodeller i NoSQL Workbench  - Flow chart  - Logbog  - Procesrapport |

Første uge var for det meste sat af til planlægning og opsætning. Her fik jeg lavet min problemformulering, case, diagrammer og mockups. Jeg havde tid til at lave noget research på nogle af de nye teknologier jeg ville prøve kræfter med som AWS og NoSQL Workbench. Jeg lavede også et helt lille testprojekt, hvor jeg øvede opsætningen af Lambda, DynamoDB og API Gateway.

I forhold til den estimerede tidsplan passer den nogenlunde. Jeg fik ikke startet på mit hovedprojekt, som jeg ellers havde troet.

**Uge 2 (3-7 marts)**

|  |  |
| --- | --- |
| **Dag** | **Aktivitet** |
| Mandag | - Opsætning af DynamoDB, Lambda og API Gateway hos AWS  - Opsætning af Google Maps API  - Opbyg API backend for users  - Logbog |
| Tirsdag | - Besøg fra Dansk Metal  - Models and Controllers  - Logbog |
| Onsdag | - Logbog  - Test af backend (AWS og Google Maps)  - Frontend |
| Torsdag | - Logbog  - Frontend |
| Fredag | - Logbog  - Frontend (brugerhåndtering)  - Procesrapport |

Anden uge starter fint, med at jeg får opsat mine services hos AWS og får min API key hos Google. Jeg begynder at bygge min API til brugerhåndtering.  
Jeg får testet mine services og alt ser indtil videre fint ud.   
Jeg får begyndt arbejdet på min frontend.

I forhold til den estimerede tidsplan, burde jeg have været færdig med opsætningen af mine services. Med andre ord; det burde bare køre nu. Derudover, følger jeg stadig tidsplanen.

**Uge 3 (10-14 marts)**

|  |  |
| --- | --- |
| **Dag** | **Aktivitet** |
| Mandag | - Frontend (Valg af destination)  - Logbog  - Processrapport |
| Tirsdag | - Logbog  - Frontend (Opsætning af præferencer, Destination)  - Google Maps API  - Procesrapport |
| Onsdag | - Forsøgte at løse problem brugerhåndtering. Fejlede.  - Logbog |
| Torsdag | - Logbog  - User Preferences Page |
| Fredag | - Logbog  - Trip Suggestions Page |
| Lørdag | - Logbog  - Edit trip page |
| Søndag | - Logbog  - Detaljeret overblik over den enkelte dag + navigation. |

Denne uge, begynder planen at skride. Jeg sidder skiftevis og arbejder med front- og backend. Sidder og kæmper med brugerhåndtering. Og når det bliver for frustrerende, sidder jeg lidt med noget frontend.   
Onsdag aften sætter jeg brugerhåndtering til side, for at fokusere på resten af min app. På dette tidspunkt var planen at vende tilbage til det igen på et senere tidspunkt.  
Torsdag til søndag sidder jeg hovedsageligt med frontend. Mit fokus nu er at have et funktionelt produkt at vise til svendeprøven.   
Grundet udfordringerne med brugerhåndteringen, ender jeg med at sidde og arbejde på projektet hele weekenden også.

I forhold til den estimerede tidsplan, var planen sat sådan op, at jeg ikke skulle arbejde i weekenden. Jeg skulle også have været færdig med min demo.   
Jeg skulle også have begyndt på min produktrapport.

**Uge 4 (17-21 marts)**

|  |  |
| --- | --- |
| **Dag** | **Aktivitet** |
| Mandag | - Logbog  - Bugfixing og cleanup. |
| Tirsdag | - Logbog  - Produktrapport  - Procesrapport |
| Onsdag | - Logbog  - Procesrapport  - Produktrapport |
| Torsdag | - Logbog  - Produktrapport  - Procesrapport |
| Fredag | - Logbog  - Procesrapport  - Sidste polering af projekt inden aflevering |

Hele mandag brugte jeg på at fikse alle de bugs jeg havde fundet undervejs gennem de løbende tests af min app. Der er stadig nogle stykker, men fik ram på de fleste.

Resten af ugen gik med at skrive rapporter. Jeg havde samlet ind og skrevet noter til min procesrapport undervejs i udviklingen. Produktrapporten skulle jeg skrive fra bunden.

Som en sidenote, forstår jeg slet ikke hvordan jeg skulle have haft tid til at skrive rapport under udviklingen af mit produkt. Forstår heller ikke hvorfor der skal skrives to rapporter, når man sagtens bare kunne lave en samlet.

# Perspektivering

# Konklusion

# Kilder og referencer

## YouTube

[Amazon DynamoDB in .NET - The Getting Started Guide](https://www.youtube.com/watch?v=BJYDWMN8taI) – Mukesh Murugan

[Deploy .Net Core Web API (.Net 8) to AWS Lambda Function from VS2022 and Config with The API Gateway](https://www.youtube.com/watch?v=9a-wpPg0QyI) – Lucky Happy

## AI

[ChatGPT](https://chatgpt.com/)

GitHub Copilot

## Grafik

Michelle Wittstrøm Petersen – Mockups, app ikon og forsidebillede.

# Bilag

## Bilag 1 – Mockups

