**FlipNow**

**Processrapport**

**// insert flip card here**

**Svendeprøveprojekt 2023**

**Daniel Simonsen**

**Titelblad**

**Elev:**

Daniel Simonsen

**Firma:**

SKO

**Projekt:**

FlipNow

**Uddannelse:**

Datateknikker med speciale i programmering

**Projektperiode:**

16/10/2023 – 03/11/2023

**Afleveringsdato:**

xx/xx/2023

**Fremlæggelsesdato:**

xx/xx/2023

# **Læsevejledning**

Der anbefales at læse procesrapporten/produktrapporten, fordi…

# **Forord**

Ord som læseren måske ville have svært ved at forstå

Contents

[**Læsevejledning** 3](#_Toc148532861)

[**Forord** 3](#_Toc148532862)

[**Casebeskrivelse** 5](#_Toc148532863)

[**Problemformulering** 5](#_Toc148532864)

[**Målgruppe** 5](#_Toc148532865)

[**Afgrænsning** 5](#_Toc148532866)

[**Metode- og teknologivalg** 6](#_Toc148532867)

[**Frontend** 6](#_Toc148532868)

[**Backend** 7](#_Toc148532869)

[**Databaser** 7](#_Toc148532870)

[**IDE** 8](#_Toc148532871)

[**Design program** 9](#_Toc148532872)

[**Opgavestyring** 9](#_Toc148532873)

[**Projektplanlægning** 10](#_Toc148532874)

[**Github** 10](#_Toc148532875)

[**Tidsplan** 10](#_Toc148532876)

[**Realiseret tidsplan** 11](#_Toc148532877)

[**Konklusion** 11](#_Toc148532878)

[**Logbog** 12](#_Toc148532879)

[**Bilag** 13](#_Toc148532880)

# **Casebeskrivelse**

## **Problemformulering**

Vendespil er sjovt og nemt at forstå. Men de fleste vendespil online kan kun spilles alene. Hvad nu hvis, man kunne spille vendespil med alle sine venner?

## **Målgruppe**

Mennesker der keder sig i SKO-dagen

## **Afgrænsning**

Det finder vi ud af

# **Metode- og teknologivalg**

## **Frontend**

Til frontend teknologi har jeg valgt at bruge React.js. Jeg har til tidligere projekter brugt React, og har bl.a. eget bibliotek til frameworket. Jeg tænker derfor, at min effektivitet ville være hurtigst, end hvis jeg bl.a. havde valgt ASP.NET eller Vue, som er andre frameworks jeg også har brugt til frontend udvikling, men dog ikke har nær så meget erfaring, som jeg har med React.

Som alternativer kunne jeg bl.a. bruge ASP.NET eller Vue.js. Da min backend allerede bruger ASP.NET, ville det være nemt at lave frontend med samme C# modeller. Dog har C# frontend frameworks problemer med klient integrationer, hvor man alligevel skal arbejde i JavaScript. Jeg har ikke noget imod at arbejde i JavaScript, udover at jeg mangler mine typedefinationer, dog synes jeg, at det giver mere mening at lave en frontend med klientintegration i samme sprog, og ikke dele det op i 2 forskellige dele.

Herunder kommer bl.a. Vue.js, som vi blev introduceret til på hovedforløb 3. Jeg var ret glad for Vue, da vi blev introduceret til det. Dog efter introduktion til Vue, begyndte jeg at lære React, da React er et populært framework indenfor frontend udvikling. Begge frameworks har deres egne styrker – f.eks. er Vue rigtig god til at holde alt inde i sin egen fil, både markup, style og funktionalitet, hvorimod man *kan* gøre det samme i React, men React supporter ikke direkte styling i komponentdokumentet som Vue gør. Vue kan godt lide at dele filen op i mindst 3 sektioner: Markup, funktionalitet og style, hvorimod React blander både markup og funktionalitet i samme scope, hvilket giver mere mening for mig. Derudover bryder jeg mig ikke om store filer, som Vue filer godt kan blive. I React laver jeg typisk en mappe og 3 filer for et komponent, hvilket jeg synes giver bedre struktur.

## **Backend**

Til backend teknologi har jeg valgt at bruge ASP.NET. Jeg har lavet flere API’er i ASP.NET dog også nogle stykker i Node.js. Ligesom i min frontend valg, kunne jeg sagtens have valgt at bruge samme sprog – altså lave en backend i TypeScript. Jeg synes dog personligt, at et ASP.NET API er bedre strukturere, hvilket gør projektet mere overskueligt. Derudover kommer ASP.NET API’er også med ”Swagger”, som er et værktøj, der hjælper udvikleren med at teste API’ets endpoints. Swagger hjælper utroligt meget til testning af API’ets endpoints, og derfor sparer jeg en masse tid.

Vælger jeg at lave mit API i Node.js, ville jeg kunne bruge samme TypeScript modeller i min frontend, dog mister jeg Swagger til testning. Vælger jeg Node.js som backend, skal jeg også finde ud af, hvordan jeg gemmer i en SQL-database. Jeg ved allerede, hvordan jeg gemmer i en MSSQL i ASP.NET via EntityFramework, og har tidligere kæmpet med at gemme i en SQL-database i Node.js uden held.

## **Databaser**

Til database bruger jeg SQL til at gemme min spilhistorik, spillere og deres score til de spil, de har været med i.

Jeg bruger en SQL-database, fordi jeg har relationer mellem mit data. Her kan jeg bl.a. bruge EntityFramework, som er et værktøj, der bruger ”code-first” princippet, hvilket betyder at jeg bare kan oprette mine modeller i kode, og så vil EntityFramework sørge for at oprette en database samt relevante tabeller til mig.

Alternativt kunne jeg vælge at bruge en form for dokument-database som MongoDB, CouchDB eller PouchDB. Som regel, når man gemmer på en historik, ville det være mere effektivt at gemme dataen i en dokument-database, da det fylder og koster mindre, fordi der ikke skal bruges tid på at tjekke kolonnetyper, men man bare kan smide dataen ind i samlingen. Det gør selvfølgelig også, at dataen kan være usikker, da dokument-databaser ikke har faste skemaer, hvilket betyder, at der enten kan mangle data eller komme unødvendigt data ift. hvad der var tiltænkt.

## **IDE**

Som IDE bruger jeg både Visual Studio og Visual Studio Code.­ Jeg bruger Visual Studio til at starte mit API i C#, og Visual Studio Code til at starte mit React frontend.

Visual Studio er bygget til at køre C# programmer, og Visual Studio Code er bedre oplevelse, når det gælder frontend udvikling. Jeg kunne nok sagtens nøje med at bruge én IDE, men ikke uden at gøre det unødvendigt svært for mig selv. I Visual Studio behøver jeg bare at trykke på ”Start” for at starte mit API og Swagger, hvorimod Visual Studio Code kræver noget eksternt konfiguration. Samtidig bruger jeg Visual Studio Code til frontend, da syntaksen i Visual Studio Code er mere læseligt, end hvis jeg havde brugt Visual Studio.

Alternativt kunne jeg bl.a. bruge JetBrain’s ”Rider” IDE, som jeg har hørt meget godt om, når det gælder udvikling i C#. Visual Studio er et meget tungt program, og håndterer ikke computerens resurser lige så godt som Rider. Dog har jeg aldrig brugt IDE’en før, og jeg har ikke tid til at finde ud af, hvordan man sætter et API projekt op.

## **Design program**

Til design af frontenden bruger jeg Figma. Figma er et program som designere bruger til app/hjemmeside design. Jeg ville gerne designe min hjemmeside, før jeg gik i gang med at kode, selvom jeg ikke har meget erfaring med Figma.

Alternativt kunne jeg selvfølgelig have tegnet noget på et stykke papir, eller brugt et paint program som Microsoft Pain eller Paint.NET, som jeg allerede har på min computer. Jeg ville dog hellere bruge et program, som er bygget til UserInterface design. Jeg kunne også have valgt at bruge Adobe XD, som er endnu et populært værktøj indenfor design verden, men jeg valgte at bruge Figma, da Figma’s gratis version har de minimale features jeg har brug for, for at lave et godt design.

Havde jeg brugt et paint program, skulle jeg tegne og farvelægge hele tiden og forestille mig brugerinteraktion, hvorimod Figma også kan lave funktionelle prototyper og komponenter udover at bare tegne et design. Det samme kan Adobe XD, men for at få adgang til XD, skal jeg have en licens, som vi ikke har som datateknikker elever.

## **Opgavestyring**

Til opgavestyring bruger jeg Azure Devops og GitHub. Her er det oplagt at bl.a. bruge Azure Devops, da det er et populært værktøj i udviklerbranchen, men også at Devops kan sættes op til at benytte SCRUM. SCRUM er også meget populært i branchen, og jeg hører tit at virksomheder nævner, at de benytter SCRUM som deres agile metode.

Jeg er efterhånden ved at være vant til at bruge Devops som opgavestyring, men kunne bl.a. også bruge GitHub’s eget system, hvilket også kunne være oplagt. Både Devops og Github kan gemme repositories i skyen, og har opgavestyringssystemer indbygget. Mange af mine egne repositories findes på min GitHub konto, så den eneste grund til at jeg bruger GitHub, er fordi det er her, alle mine andre repositories ligger samlet.

Til opgavestyring bruger jeg Github. Jeg har god erfaring med Github og deres versionsstyring, så det eneste jeg mangler er opgavestyring. Dette kan Github også tilbyde, i form af issues (opgaver), der kan vises i todo list eller roadmap.

Alternativt kunne jeg bruge Azure DevOps, som vi bruger på SKO. DevOps supporter projektstyring i SCRUM og har tilknyttet git repository. Jeg er mere erfaren i at bruge Azure DevOps til opgavestyring, men jeg vil gerne prøve Github’s system af.

# **Projektplanlægning**

## **Github**

Projektet bliver planlagt i Github via deres Project feature.

Her kan jeg bl.a. få 3 forskellige views på mine issues (opgaver), jeg skal have lavet.

## **Tidsplan**

Efter at have lavet mine issues, kunne jeg bare sætte dem ind i projektets ”Roadmap view”.

A screenshot of a computer

Description automatically generated

Den fulde tidsplan ses i bilag.

## **Realiseret tidsplan**

Vandret billede findes som bilag 4

# **Konklusion**

Hvordan synes jeg processen gik?

# **Logbog**

# **Bilag**

Bilag 1: Estimeret tidsplan

A screenshot of a computer

Description automatically generated