Aarhus Universitet

FORÅR 2021

Plug-n-Play - Analyse

Gruppe 2021F12

Navn	Studienummer
Trí Nguyen	201610974
Niclas Spas	20106235

Vejledt af
Torben Gregersen

1. juni 2021

Indhold

0.1 Indledning

Dette dokument vil omhandle analytisk tilgang til mulige teknologier som vil blive brugt til udviklingen af projektet. Der vil diskuteres og redegøres for hver teknologis styrker og svagheder, hvorved der så vil konkluderes i en valgt teknologi for det diskuterede område. Dette vil blive gennemgået i forskellige niveauer af projektet, fra den øverste grænseflade til databasen som ligger nederst i kommunikationskæden. Der vil til sidst også drages diskussion til nogle lignende projekter, hvor forskelle og ligheder kan forklares.

Analysen af teknologierne bruges til at finde værktøjer eller løsninger, som der kan gøres brug af til projektet. De teknologier der bliver diskuteret heri er:

- Frontend teknologier - Single Page vs. Classic MVC?
- Frontend teknologier - Hvilket sprog?
- Frontend teknologier - Custom CSS mod resource pakke som Material Design eller Bootstrap?

- Backend teknologier - Hvilket Framework?
- Test teknologier - Hvilket Framework?
- Database teknologier - Hvem skal opbevare data?
- GDPR - Hvilke data skal/må opbevares?
- Spil-logik - Hvem har ansvaret?
- Websockets - Vanilla websocket eller websocket pakke?
- Udviklingsværktøjer - Hvilke IDE'er?
- Projektstruktur - Hvordan skal det opdeles?
- Lignende projekter - Er der værdi i at udvikle dette projekt?

Kapitel 1

Teknologier og projekter

1.1 Frontend teknologier

Til udvikling af frontend er der flere områder som skal undersøges. Før der kan tages nogen beslutning om hvilke teknologier som skal anvendes er det nødvendigt at overveje hvilken tilgang der skal tages til frontenden. Dette kan enten være en klassisk serverside rederet arkitektur baseret på Model View Controller mønsteret (MVC)[1], eller vælge at lave en klar separation mellem backend og frontend ved at lave en single page applikation (SPA)[2] ved at benytte et frontend framework.

1.1.1 Single Page vs. Klassisk MVC

Klassisk MVC

Serverside MVC tilbyde visse fordele såsom direkte adgang til database laget fremfor at skulle hente data gennem et web api. Frontend og backend er samlet i et projekt og dermed ikke nødvendigt med mere en et udviklingsmiljø. Serverside rendering er godt egnet til statisk indhold, og kan med tilføjelsen af javascript og en html præprocessor, som f.eks Handlebars eller Pug, også håndtere dynamisk indhold.

Single Page Applikation

Med brugen af et frontend framework er det muligt at separere frontend og backend og derved opnå en løs kobling mellem de to. Frontend frameworks, som Angular og React, tilbyder derudover en komponent baseret arkitektur samt andre features som lifecycle

hooks og HTML conditionals som gør det nemmere at udvikle en mere dynamisk brugeroplevelse. Performancemæssigt afvikles en SPA på klientsiden hvilket gør at der serverside ikke bliver brugt ressourcer på at generere HTML sider.

Delkonklusion

Valget her falder på en SPA.Valget tages ud fra et ønske om en løs kobling mellem frontend og backend. Dette vil gøre det nemmere at udvikle frontend og backend paralelt med hinanden. Derudover er det en fordel, når det kommer til potentiel skalering af systemet, at backenden ikke skal bruge resourcer på at servere html til clienten.

1.1.2 Valg af framework

De tre største frameworks til frontend udvikling er Angular, React og Vue. Af disse kigges nærmere på Angular og React, da holdet har erfaring med disse fra tidligere.

Angular

Er et komplet frontend framework udviklet af Google. Angular udstiller og indkapsler en masse forskellige features hvilket gør det nemt og hurtigt at udvikle i. Disse features inkludere bla service som HttpClient. Dette er en service som simplificere fetch kald og returnere svar i form af observables i stedet for promises. Alle de features som Angular kommer med gør dog også at en tom Angular applikation er meget større end en tilsvarende app i f.eks React. Dette skyldes netop at Angular er et komplet framework og man ikke blot kan tilføje de funktionaliteter som man benytter.

React

React er i sig selv ikke et framework, men blot et Javascript bibliotek. Dette betyder at React i sig selv ikke har den samme bredde af funktionalitet som Angular udbyder. Dette kan dog hjælpes på vej af andre Javascript biblioteker, som er udviklet til at spille sammen med React. Den mindre funktionalitet er dog ikke kun negativt. React applikationer er meget mindre en tilsvarende Angular applikation. Denne mindre størrelse kan være med til at forbedre indlæsnings tiden for applikationen.

Delkonklusion

Begge frameworks er gode mugligheder de har deres styrker og svagheder. Det er valgt undervikling at fokusere hovedsageligt på at få mest muligt funktionalitet med i prototypen. Derfor er det nødvendigt at inddrage tidligere erfaring med de to frameworks som en overvejelse. På trods af at der er kendskab med begge framework er der meget mere erfaring med Angular på projektholdet hvilket gør valgtet falder på Angular som framework.

1.1.3 Udviklingssprog

Med valget om at benytte Angular frameworket til at udvikle en SPA er der kun to muligheder for udviklingssprog. Disse muligheder er JavaScript[**?**] og TypeScript[**?**].

JavaScript (JS)

Javascript er det mest udbredte programmeringssprog til web-applikationer. JavaScript er et letvægtigt programmeringsprog. JavaScript er et programmerings sprog uden typer. Dette er en af ulemperne ved JS. Kombinationen af at JS bliver fortolket og ikke kompileret sammen med dets mangel på typer gør at kode kompleksitet meget hurtig bliver meget høj hvis man skal lave et større projekt i JS. Dette er fordi det er meget begrænset hvad man kan få af hjælp fra dit IDE under udviklingsprocessen, og man vil derfor først opleve fejl når koden bliver kørt.

TypeScript (TS)

TypeScript er en udvidelse af JS oprindeligt udviklet af Microsoft men som i dag er open source. TypeScript er i sin simpleste form blot JS men gjort type stærkt. Dette er gjort tydeligt da TS kompileres til JS. At TS er type stærkt har dog mange fordele da det betyder en stor forbedring i hjælpeværktøjer som intellisense, og muligheden for at typetjekke koden under kompilering før den køres hvilket hjælper meget med debugging.

Delkonklusion

Der vælges at benytte TypeScript som programmeringssprogs ikke blot for fordi det betyder adgang bedre udviklingsværktøjer, men ogsp fordi Angular er som er valgt som frontend teknologi er udviklet til at benyttes med TS.

1.1.4 Custom CSS eller Resource Package

Styling er en vigtig del af det at bygge en brugergrænseflade. Laver man selv stylingen af sine komponenter har man fuld kontrol over udseendet mens med en ressource pakke ofrer man en del af denne kontrol for en besparing i tid.

Custom CSS

Med custom CSS menes her for at man selv udvikler alt det CSS som anvendes i applikationen. Dette vil betyde at man har fuld kontrol over appens udseende. Dette kan dog være ekstremt tidskrævet især hvis der ikke er stor erfaring med brugen af css.

Resource package

Brugen af en ressource pakke betyder at man binde sig til et design som er bestemt af udviklerne af den pågældende ressource pakke. Dette er dog nødvendigvis ikke et stor problem hvis designet passer til applikation. dernæst kan farver somregel sættes separat. Ressource pakker som for eksempel Angular Material tilbyder mere end blot styling og design. De tilbyder også en lang række strukturelle komponenter som kan anvendes til at sætte grænsefladen op.

Delkonklusion

Valget her falder på at benytte en ressource pakkke. Specifikt vælges der at benytte Angular Material[**?**] samt Angular flex[**?**] layout som er ressource pakker udvikling af Google specifikt til brug med Angular frameworket. Både fordi selve designet er passende til applikationen men også i høj grad på grund af udvalget af strukturelle komponenter som tilbødes.

1.2 Backend teknologier

Når der skal tænkes på forskellige mulige backend-teknologier er der forskellige at vælge mellem, hver med deres fordele og ulemper, som kan være mere eller mindre relevante for det overordnede projekt. Når der skal vælges teknologier for backend skal der vurderes på kode-sproget der bliver brugt, og hvilken frihed/restriktion det sprog har til brugen i backend.

1.2.1 C# - ASP.Net Core

ASP.Net Core[3] er et fleksibelt og stærkt backend framework. Styrkerne for brug af dette framework ligger i dets optimering af kode gennem kompilering. Dertil er det også et af de frameworks vi er blevet undervist i at bruge på studiet, så dertil vil der være grunderfaring med brug af dette framework. Gruppen har også tidligere haft erfaring i at udvikle backend i ASP.Net Core, så denne erfaring er også relevant til det fremtidige valg af teknologi.

1.2.2 JavaScript - Node.js/Express

Node.js[4] er et runtime environment, der gør brug af styrkerne fra JavaScript i dets backend-udvikling. Fordelen ved brug af Node.js er at frontend og backend vil have muligheden for at blive skrevet i samme sprog, så der er færre variabler at overveje når der skal ses mellem de to blokke. Med et JavaScript baseret framework er det derfor også meget let at finde packages/moduler i open-source biblioteker, hvilket gør det let at finde rundt i hvilke packages man måske gerne vil gøre brug af. Node.js agerer som en server som eksekverer JavaScript, og det kan være svært at skalere hen ad vejen på en cloud server. Der er gjort brug af Node.js og Express på studiet til samarbejde med frontend udvikling på studiet, dog på en mindre grad i forhold til .Net Core.

1.2.3 Firebase

Firebase[5] er en mobil- og webudviklingsplatform, platformen har mange forskellige tjenester som backend i en applikation med færdigudviklet materiale til at opbevare data, autentifikation, og gemme filer som brugere af applikationen kan få adgang til igen senere. Firebase har en dokumentbaseret cloud-storage database, som var en mulighed til database teknologi.

1.2.4 Delkonklusion

Til udarbejdelse af projektet ønskes der en backend der er fleksibel og kan håndtere ønskede funktionaliteter som set i Krav Specifikationen[6]. Her til vælges ASP.Net Core som det framework der vil arbejdes videre med. Med tidligere erfaringer har ASP.Net Core været et godt fundament til backend-udvikling, og har været stærk nok til at kunne håndtere den pågældende kommunikation der ønskes til projektet. Både Firebase og Express var fine alternativer til mulige backend teknologier, men de restriktioner der kommer i Firebase i form af en hel færdiglavet backend gør at man ikke selv kan bestemme hvordan

backenden fungerer og kommunikere med resten af systemet. Express har også den mindre restriktion at den eksekvere JavaScript, så at skalere op på en cloud server kan blive et problem senere hen.

1.3 Test-teknologier

Det beskrives i kapitel 5 af bogen The Art of Unit Testing[7], at under test gøres der brug af et isolation framework, som tager sig af at kunne teste de enkelte klasser og deres metoder, uden at være afhængige af deres dependencies' implementation.

1.3.1 NUnit

NUnit[8] er et framework til test af kode som har test-integration med Visual Studio. Frameworket er brugt på studiet med .Net Framework, og har derfor medfølgende erfaring til brug i projektet. NUnit har den funktionalitet under test, at den skaber en test-klasse som så kører hver enkelte test, her er det så muligt at skrive test efterfølgende af hinanden, som deler noget data hvis ønsket. Ifølge test-driven development ideologien[9] er det ikke en god ide at dele data mellem test, da det gør dem afhængige af hinanden. NUnit har også mulige TestCases[10] som gør det lettere at genbruge testkode, i stedet for at skulle skrive de samme tests igen og igen med få variationer.

1.3.2 xUnit

xUnit.net[11] er et framework til test af kode som har test-integration med Visual Studio. Frameworket er har den forskel til NUnit at den skaber en ny instans af test-klassen for hver test, så alle testene holder sig isoleret fra de andre tests. Det hjælper med at overholde ideologien i test-driven development[9]. Også selvom XUnit isolerer hver test, så er det muligt at dele data mellem testene gennem forskellige metoder. XUnit har også mulige Theory[12] som gør det lettere at genbruge testkode, i stedet for at skulle skrive de samme tests igen og igen med få variationer.

1.3.3 Moq eller NSubstitute

De to isolation frameworks Moq[13] og NSubstitute[14] har begge fordele og ulemper. Moq gør meget brug af lambda notation[15], hvilket kan gøre testkoden mindre læsbart, Moq har dog den fordel at man kan være meget eksplicit med ens kode, så man er sikker på

hvad man laver når man laver Moq metode-kald. NSubstitute er mindre eksplicit, hvilket dog giver den en fordel af simplicitet i kode.

1.3.4 Delkonklusion

Af test-teknologier vælges der XUnit som framework for dets håndtering af test-eksekvering som sikrer en god overholdelse til test-drive development.

Isolation framework vælges NSubstitute, det er mere fleksibelt i forhold til Moq, og har den fordel at gruppen har arbejdet med NSubstitute på studiet før.

1.4 Database teknologier

Til opbevaring af data skal der gøres brug af en database samt en struktur for den. Dette afsnit vil kort diskutere fordele og ulemper ved forskellige database teknologier som kan gøres brug af til projektet. Der tages udgangspunkt i en cloud-baseret database[16][17], det gør det nemmere at kunne skalere brugerbasen og andet indhold som kunne være relevant at opbevare i databasen.

1.4.1 Azure MySQL

Azure MySQL[18] er en relationsdatabase[19] som baseres på entiter og key-value pairs til at referere til andre entiteter. Entiteterne består af tabeller med indexeringer som kan queries på for at få den relevante data. Relationsdatabasen gør brug af "keys"der skal kunne referere til entiteter, hvor "foreign keys"benyttes til at referere til specifikke entiteter man ønsker at kunne bruge/anvende data fra. En relationsdatabase er meget tæt knyttet sammen og har et meget rigidt design, hvilket giver orden over en database, men kan også være en ulempe i den skærpede fleksibilitet til at kunne ændre på entiteters attributter og index.

1.4.2 Firebase

Som tidligere nævnt i afsnit 1.2 har Firebase også en cloud-storage databasefunktionalitet[20], der gør den brugbar til projektet. Dog har Firebase en speciel intern database-struktur, der gør den mindre overskuelig at arbejde med. Med nested- og top-level-collections kan det blive kompliceret i hvad skal være hvor i databasen og hvordan. Skulle Firebase gøres brug af til database skulle database-strukturen gennemtænkes noget mere end hvis man

selv ville designe den overordnede struktur. Måden Firebase fungerer på er som en dokumentbaseret database: en entitet i databasen er et dokument; en samling af den samme type dokumenter er en collection; collections kan være i forskellige niveauer, som også påvirker søgehastigheder. Der er meget man skal tænke på når man har et fastsat framework man skal sætte sig ind i for at kunne udnytte databasen bedst muligt.

1.4.3 MongoDB Atlas

MongoDB Atlas[21] er en cloud-baseret dokumentdatabase[22][23]. I modsætning til en SQL server, opererer en dokumentdatabase med mindre rigide regler, hvilket dog kræver mere arbejde i planlægningen af arkitekturen, da der ikke er nogle foreign-keys til at kunne referere til andre "entiteter", her kaldet dokumenter. Dokumenter er så her delt op i collections, delvis sammenlignbar med Firebase Firestore. Der kan dog i MongoDB Atlas gøres brug af sub-dokumenter eller indlejret JSON objekter, som så kan indeholde data der kan være relevent, eller også referere ved brug af et dokuments ID på samme måde som en foreign key benyttes.

1.4.4 Delkonklusion

De tre databasestrukturer har deres styrker og svagheder. På studiet er der gjort brug af alle tre, og erfaring er overordnet ligelidt delt op mellem dem, på nær MongoDB Atlas, som er brugt i tidligere projekter i mere dybdegående tilgang, som giver os mere tryghed i at gøre brug af MongoDB Atlas. Der vælges derved at gøre brug af MongoDB Atlas, selvom Firebase Firestore har en hurtig query time vægter det ikke nok i mod arbejdet der skal ligges i planlægningen af top-level collection, sub collection osv. Da MongoDB Atlas er mere fleksibel vil det være oplagt at bruge til udviklingen af projektets database

1.5 GDPR og lovgivning

Når der skal persisteres data er det vigtig at kende til GDPR[24] og anden relevant data-sikkerhedslovgivining[25] og standarder[26][27]. Der beskrives der hvilke data der skal vægtes i det beskyttelse og samtykke til opsamling. Lovgivningen lægger derved tryk på personfølsomme data og samtykken derom. Der er til projektet ønsket en form for profilbillede, der her kan være et billede af en potentiel bruger, derfor tages der hensyn til at opbevare billedet forsvarligt og at den potentielle bruger giver samtykke og er klar over hvad samt hvor billedet bliver brugt og opbevaret.

Andet end billedet ønskes der ikke anden personfølsom data, da offentlig information som e-mail ikke indgår som personfølsom data. De endelige overvejelser til hvilke data der til sidst skal opbevares i systemet er derfor:

- Profilbillede (potentiel personfølsom data)
- E-mail
- Brugernavn
- Kodeord (ikke i plain-text)
- Spilhistorik (indeholder spil og gældende statistik)

1.6 Spil-logik

Placeringen af af spil logikken er et interessant spørgsmål med overvejelser som performance og ressource implikationer, snyderi og tilføjelse af nye spil.

1.6.1 Backend

Placeres spillogikken i backend vil det være nemmere at sikre i mod snyderi da det vil være nemmere at at verificere at et træk som er blevet af frontenden er et valid træk, og ikke er blevet modificeret i mellem frontendens tjek og trækket modtages af backenden. Derudover er det en fordel at spillets stadie ikke kan modificeres af andet end et gyldig træk. Ressourcekravene i backenden vil også stige i en ret stor grad da hver spillobby skal udvides med ikke bare reglerne for det pågældende spil med også spillets stadie. Selvom dette ikke er store mængde data når en enkelt spillobby betragtes er dette en ting som hober sig op med potentielt tusindevis af brugere. Sidst vil muligheden for at tilføje nye spil til platformen være mere omstændigt. Det vil både kræve implementation i både back- og frontend for at tilføje et nyt spil. Den endelige logik ville ligge i backenden men ville alligevel skulle implementeres i en grad i frontenden for at sikre UI til spillet fungere ordenligt og for eksempel ikke tillader ugyldige træk.

1.6.2 Frontend

Lægges spillogikken i frontend giver det visse fordele i henhold til ressourceforbrug samt mulighed for nemt at kunne tilføje nye spil. Det udløser dog også visse ulemper som skal håndteres.

Skulle der sikres imod snyd ville det kræve at et træk bliver dobbelt tjekket ved modtagelse. Dette vil betyde det er nødvendigt med en protokol til synkronisering af spillet stadie efter hvert træk for at sikre at et givet træk resultere i samme nye stadie på begge klienter. Dette er altså meget mere komplekst at skulle sikre sig mod snyd når der ikke er en upartisk mellemmand som holder styr på stadiet.

Når det kommer til performance og ressourcer er det både godt og dårligt. Det er godt i den forstand at alt arbejdet er flyttet til klientens maskine. Dette betyder at med en minimal løsning er alt hvad der er krævet af serveren er at den videresender beskederne mellem klienterne. Resten af processeringen kan så foregå på klient maskinen. Det problematiske ved at det ligger i frontenden er indvirkningen på applikationens størrelse og derved den tid det tager at indlæse siden til at starte med. Dette kan blive et problem hele i den med systemet er at lave en platform som kunne tilbyde mange forskellige spil og hvor det var nemt at tilføje nye spil til. Angular frameworket kan dog hjælpe til her med dets lazy loading[**?**] funktionalitet. Lazy loading lader applikation vente med at hente moduler indtil de skal bruges. Det vil altså være mulig at implementere hvert spil i deres eget modul som først hentes når brugeren det specifikke spil.Dette betyder derfor at med lazy loading vil mængden af spil have meget lille påvirkning på størrelsen af applikationen når den hentes til at starte med. Når det kommer til tilføjelsen af nye spil er det ideelt at at have det hele samlet et sted således at både regelsæt og UI implementeres samlet i stedet for både front og backend skal involveres.

1.6.3 Delkonklusion

Med gode argumenter for begge løsninger kigges der til målene for projektet. Her er et af dem at udvikle en platform hvor det er nemt at tilføje nye spil til platformen. Af denne grund vælges der at ligge spillogikken i frontenden.

1.7 Websocket

Det er valgt at bruge websocket til tovejs kommunikation mellem klient og server. Derfor skal det afklares hvorvidt en ren implementation af websocket protokollen anvendes eller om der anvendes et bibliotek som SignalR[**?**].

1.7.1 Websocket protokol

At implementere websocket protokollen selv ville være en større opgave da det ville kræve både at implementere det i C# for for backend og TypeScript i frontenden. Dette ville være en tidskrævende opgave for blot at opnå grundlæggende websocket funktionalitet.

1.7.2 Websocket bibliotek

Den største fordel ved at benytte et tredje parts bibliotek til at websocket er tidsbesparelsen på ikke selv at implementere det. For uden tidsbesparelsen følger der ofte ekstra funktionalitet med. Et eksempel her er biblioteket SignalR[**?**] som er udviklet af Microsoft. SignalR giver mulighed for nemt at lave gruperinger af websocket, en feature som er perfect til for eksempel spillobbyer.

1.7.3 Delkonklusion

Til websocket vælges der an benytte biblioteket SignalR. Dette fordi der både er en version til ASP Core og en version til Typescript så det passer perfekte ind i den valgte teknologi stack. Herudover tilbyder SignalR også ekstra funktionalitet som Rooms hvilket er perfekt til flere af systemets features.

1.8 Udviklingsværktøjer

Dette afsnit vil hurtigt gennemgå de udviklingsmiljøer der er relevante for projektets udvikling ud fra de valgte sprog/frameworks/biblioteker fra tidligere afsnit.

1.8.1 Backend

Til udvikling af en ASP.Net Core server jvf. afsnit 1.2 gøres der brug af Visual Studio Enterprise 2019[28]. Der har i løbet af studiet været brug af Visual Studio som IDE i forskellige kurser med forskellige versioner. Grunden til brug af 2019 versionen baseres på opdateret funktionalitet og quality-of-life tilføjelser til brugergrænsefladen for udviklere[29].

1.8.2 Database

Til brug af MongoDB Atlas jvf. afsnit 1.4 bruges MongoDB's webplatform, som agerer brugergrænseflade for den gængse interaktion med databasen. Her kan databasen sæt-

tes op og kobles til backend. Der behøves derfor ikke et bestemt udviklingsværktøj til bearbejdning af databasen.

1.9 Projektstruktur

Jævnført afsnit 1.1 og afsnit 1.2, kan der konkluderes ud fra de to afsnit, at projektets udførelse kommer til at ske over 2 forskellige del-projekter, hver med deres relevante udviklingsværktøjer, mappestruktur, og mappefordelinger. Projektet kommer til at blive udviklet og designet parallelt, og skal stadig kunne kommunikere med hinanden, dette skal derfor også klargøres i den omfattende dokumentation af projektet.

1.10 Lignende projekter

Når man ser på projektet, Plug-n-Play, kan man finde lignende projekter og produkter allerede på markedet. Der vil ses på 4 forskellige projekter, som implementere en spilplatform på nogenlunde samme facon som Plug-n-Play vil implementere.

1.10.1 ChessBot

Der er i 2020 udviklet en online skakplatform ved navn ChessBot[30], hvor mange basale web-udviklingsovervejelser også kan ses i dette projekt. Dette projekt abstraherer dog selve spillet væk, og fokusere mere på de bruger-interaktive funktionaliteter, som ChessBot ikke gør brug af.

1.10.2 Board Game Arena

I 2010 startede idéen med Board Game Arena[31], som var en online browser platform, hvor folk kunne spille de brætspil som var implementeret til platformen. Spillene som er tilgængelige er i løbet af platformens levetid været udviklet af ikke kun ejerne, men også frivillige brugere, der ønsker at biddrage til platformens udvikling. Mange af de funktionaliteter der fremvises på Baord Game Arena ligger meget overens med hvad dette projekt vil præstere.

1.10.3 PlayingCards.io

Platformen beskriver selv:

"PlayingCards.io falder ind under "virtual tabletop-genren af videospil, hvilket betyder, at det giver en delt spilleflade, hvor spilkort og brikker kan flyttes rundt. Når brikker flyttes, ser alle andre spillere ændringerne med det samme. Som alle virtuelle bordplader skal spillerne selv styre skift og håndhæve regler."[32]

Platformen understøtte mange af de samme funktionaliteter som dette projekt også vil implementere.

1.10.4 Dominion Games

Dominion Games er udviklet af Shuffle iT[33] og implementere spillet Dominion[34] i en online platform. Den mere specificerede online-platform implementere meget specifikt for Dominion, da der er mange nuancer til spillets regler, som kan ændre på hvert spils struktur.

1.10.5 Delkonklusion

De undersøgte projekter har alle ligheder med Plug-n-Play især Board Game Arena[31], som implementere mange forskellige spil under en samlet platform. Board Game Arena er også den platform som lettest kan sammenlignes med Plug-n-Play, da det er samme principper der arbejdes ud fra. Den sociale interaktion der vil implementeres i Plug-n-Play er der dog ikke på samme måde i Board Game Arena, og ingen af de andre projekter har det selvsamme funktionaliteter som Plug-n-Play vil implementere. Der kan så drages konklusion i, at Plug-n-Play projektet er værd at arbejdes videre med, da selvom der findes lignende projekter, så er der ikke nogle der implementere funktinaliteterne på samme måde som Plug-n-Play ønsker at implementere.

1.11 Konklusion

Ud fra de tidligere punkter som var sat op i Afsnit 0.1 blev der diskuteret og analyseret på de forskellige emner. Nogle af emnerne var mere dybdegående end andre, og nogle beslutninger vare hurtigt overstået ud fra analyse af alternative løsninger. Der vil her konkluderes med en tabel over de valgte teknologier/beslutninger jvf. de tidligere afsnit i Kapitel 1.

Emne	Problemstilling	Beslutning
Frontendteknologier	*Single Page vs. Klassisk MVC?*	Frontended udvikles separat fra backend og udvikles som en Single Page Application med Angular frameworket.
Frontendteknologier	*Hvilket sprog?*	Til de nævnte biblioteker og frameworks vil der gøres brug af TypeScript.
Frontendteknologier	*Custom CSS eller Resource Package.*	Der anvendes layout komponenter og styling fra Angular Material
Backendteknologier	*Hvilket Framework skal arbejdes med?*	Backended udvikles i C# med ASP.Net Core frameworket.
Testteknologier	*Hvilket Framework skal arbejdes med?*	Test til backend udvikles i XUnit# med NSubstitute frameworket.
Databaseteknologier	*Hvem/hvor skal data opbevares?*	Der gøres brug af en cloud-database med MongoDB Atlas.
GDPR	*Hvilke data skal/må opbevares?*	Der gemmes e-mail; muligt billede, hvor billedet kan klassificeres som følsom personoplysninger; spil-historik for oprettede brugere; og venneliste.
Spil-logik	*Hvem har ansvaret for logikken til spil?*	Ansvaret for spil logik lægges i frontenden.
Websockets	*Hvilken websocket pakke skal der bruges?*	Der er valgt at gøre brug af SignalR pakken, som passer godt med den udviklings-stack systemet udviklesi.

Emne	Problemstilling	Beslutning
Udviklingsværktøjer	*Hvilke IDE'er er der brug for til udviklingen af projektet?*	Frontend i Visual Studio Code. Backend i Visual Studio 2019. Databasen har web-baseret brugergrænseflade.
Projektstruktur	*Hvordan er projektet struktureret?*	Projektet bliver opdelt mellem frontend - server - og database.
Lignende projekter	*Er projektet værd at udvikle i sammenligning med andre projekter på markedet?*	Projektet har ligheder med andre projekter på markedet, men biddrager med en unik implementation af de lignende funktionaliteter

Kapitel 2

Litteratur

[1] Wikipedia. Model view controller. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller, 2021. URL date: 5. februar 2021.

[2] Wikipedia. Singe page application. https://en.wikipedia.org/wiki/Single-page_application, 2021. URL date: 5. februar 2021.

[3] Microsoft. .net documentation. https://docs.microsoft.com/en-us/dotnet/fundamentals/, 2021. URL date: 18. februar 2021.

[4] OpenJS Foundation. Node.js documentation. https://nodejs.org/en/docs/, 2021. URL date: 18. februar 2021.

[5] Google. Firebase documentation. https://firebase.google.com/docs, 2021. URL date: 18. februar 2021.

[6] Niclas Spas Tri Nguyen. Plug-n-play - krav specifikation, 2021.

[7] Roy Osherove. The art of unit testing, 2013.

[8] .NET Foundation. Nunit. https://nunit.org/, 2019. URL date: 12. april 2021.

[9] Wikepedia. Test-driven development. https://en.wikipedia.org/wiki/Test-driven_development, 2021. URL date: 28. april 2021.

[10] NUnit. Testcase. https://docs.nunit.org/articles/nunit/writing-tests/attributes/testcase.html, 2018. URL date: 28. april 2021.

[11] Microsoft. About xunit.net. https://xunit.net/, 2021. URL date: 12. april 2021.

[12] XUnit. Write your first theory. https://xunit.net/docs/getting-started/netfx/visual-studio#write-first-theory, 2021. URL date: 28. april 2021.

[13] Manas Clarius and InSTEDD. Moq. https://github.com/moq/moq4, 2021. URL date: 12. april 2021.

[14] Roy Osherove. The art of unit testing. https://nsubstitute.github.io/, 2013. URL date: 12. april 2021.

[15] Wikipedia. Anonymous function. https://en.wikipedia.org/wiki/Anonymous_function, 2021. URL date: 28. april 2021.

[16] IBM. What is a cloud database? https://www.ibm.com/cloud/learn/what-is-cloud-database, 2021. URL date: 26. februar 2021.

[17] Wikipedia. Cloud database. https://en.wikipedia.org/wiki/Cloud_database, 2021. URL date: 26. februar 2021.

[18] Microsoft. Azure mysql documentation. https://docs.microsoft.com/en-us/azure/mysql/, 2021. URL date: 18. februar 2021.

[19] Wikipedia. Relational database. https://en.wikipedia.org/wiki/Relational_database, 2021. URL date: 5. februar 2021.

[20] Google. Cloud firestore documentation. https://firebase.google.com/docs/firestore, 2021. URL date: 18. februar 2021.

[21] MongoDB. Mongodb atlas documentation. https://docs.atlas.mongodb.com/, 2008. URL date: 25. februar 2021.

[22] Wikipedia. Document-oriented database. https://en.wikipedia.org/wiki/Document-oriented_database, 2021. URL date: 25. februar 2021.

[23] MongoDB. What is a document database? https://www.mongodb.com/document-databases, 2020. URL date: 25. februar 2021.

[24] GDPR.DK ApS. Gdpr. https://gdpr.dk/, 2021. URL date: 3. marts 2021.

[25] Datatilsynet. Databeskyttelse. https://www.datatilsynet.dk/databeskyttelse, 2021. URL date: 3. marts 2021.

[26] ISO. Information security management. https://www.iso.org/isoiec-27001-information-security.html, 2021. URL date: 3. marts 2021.

[27] Digitaliseringsstyrelsen. Iso 27001 er sikkerhedsstandarden for statslige myndigheder. https://digst.dk/sikkerhed/iso-27001/, 2021. URL date: 3. marts 2021.

[28] Microsoft. Visual studio. https://visualstudio.microsoft.com/vs/, 2021. URL date: 26. februar 2021.

[29] Microsoft. What's new? https://visualstudio.microsoft.com/vs/whatsnew/, 2021. URL date: 26. februar 2021.

[30] Valeria Polukhina Wallejus Tri Nguyen, Niclas Spas. Chessbot - projektrapport, 2020.

[31] Sourisdudesert. Board game arena. https://da.boardgamearena.com/, 2010. URL date: 15. januar 2021.

[32] PlayingCards.io. Playingcards.io. https://playingcards.io/, 2010. URL date: 15. januar 2021.

[33] Shuffle iT. Dominiongames. https://dominion.games/, 2016. URL date: 15. januar 2021.

[34] Wikipedia. Dominion (card game). https://en.wikipedia.org/wiki/Dominion_(card_game), 2020. URL date: 15. januar 2021.