

Rapport

Magnus Jørsby | Mathias Bidstrup | Tobias Mejnertsen | Niclas Vernersen

210118116peh | 07-12-2018

Indholdsfortegnelse

1 Forord.....	4
1.1 Formål	4
1.2 Hvem er vi	4
2 Indledning	5
2.1 Problemformulering.....	5
3 Projektbeskrivelse	6
3.1 Systemkort og flowchart.....	6
4 Teknologier	6
4.1 Versionskontrol	6
4.1.1 Hvordan fungerer TFS	6
4.1.2 Hvordan gavner TFS <i>Ud i det blå</i>	7
4.2 ASP.NET Core -MVC	7
4.2.1 RazorMarkup.....	9
4.2.2 Backend.....	9
4.3 Frontend.....	16
4.3.1 MVC Views.	16
4.3.2 JavaScript	17
4.3.3 Bootstrap	18
4.3.4 Sass.....	19
4.4 Domæne, SSL og hosting.....	19
4.4.1 Hosting	19
4.4.2 Domæne.....	20
4.4.3 SSL	20
4.4.4 Permanent Redirectfor statisk domænebrug i Azure.....	20
4.4.5 Azure og sikkerhed.....	22
5 Beskrivelse af <i>Ud i det blå</i>	23
5.1 Brugerfladen	23
5.1.1 Design.....	24
5.2 Brugerfladens funktionalitet.....	25
5.3 Kommunikation imellem frontend og backend	28
5.4 UX.....	30

5.5 Modeloverblik	30
5.6 Databaseoverblik	31
5.7 Klasseoverblik	32
6 Udviklingsforløbet	32
6.1 Use cases	33
6.2 Projektstyring	34
6.3 Afprøvning	35
7 Afgrænsninger	35
7.1 Skudt under mål "_(ツ)_/"	35
8 Systemets fremtid	36
8.1 Push notifikationer	36
8.2 GPS-Tracking	36
8.3 Ekstern API data til områder og begivenheder	36
8.4 Unit testing	36
8.5 CI og CD i Azure DevOps	37
8.6 Flere loginudbydere i Azure B2C	37
8.7 Forbedre validering	37
8.8 Detaljer	37
8.9 Udvide sortiment	38
8.10 Udvidelse af bedømmelser i form af statistisk og filtrering/sortering	38
8.11 Udvide flere relevante felter på begivenheder	38
8.12 Tilknyt et specifikt udendørsområde til en begivenhed	38
8.13 Paging på udendørsområder samt begivenheder	38
8.14 Fejlhåndtering og logging	39
8.15 Udvide begivenhedssystem med tilmelding/betaling	39
8.16 Administrator funktionalitet	39
8.17 Håndtering af trolls og bandeord	39
8.18 Brugerverificering og levels	39
9 Diskussion	40
10 Konklusion	41
11 Bilagsoversigt	44

1 Forord

Formålet med denne rapport er at give læseren en dybere indsigt i projektets udvikling. Den udvikles af EUX IVS, bestående af Magnus Jørsby, Mathias Bidstrup, Niclas Vernersen og Tobias Mejnertsen. Rapportens funktion er at give læseren et indblik i, hvilke overvejelser og beslutninger der er taget under projektets forløb, samt at dokumentere og beskrive produktet.

Vi har i rapporten bestræbt os på at komme grundigt omkring de relevante aspekter af projektet, for eksempel frontend, backend og teamets brug af Azure. Rapporten er udarbejdet på baggrund af første produktionsklare version af produktet, eventuelle tilføjelser eller ændringer er beskrevet i afsnit [8 Systemets fremtid](#).

1.1 Formål

Formålet med *Ud i det blå* projektet er at udvikle et system, der gør vilkårlige personer i stand til hurtigt og effektivt at finde udendørsaktivitetsområder såsom legepladser, hundeparker, fritræningsområder eller begivenheder man kan deltage i. Det skal give mulighed for at finde stederne, se bedømmelser, billeder og beskrivelser af områderne. Systemets formål er altså at få flere mennesker og kæledyr aktiveret og komme ud og benytte de frie områder, få frisk luft i lungerne og jord under neglene.

1.2 Hvem er vi

Magnus Jørsby har 4 års erfaring med ASP.NET udvikling som Full-Stack developer, hvor han har arbejdet med C#, JavaScript, Angular.js, HTML5 og CSS, til at lave hjemmesider. Samtidig har han som Full-Stack developer arbejdet med mobiludvikling i Xamarin-forms, hvor han har arbejdet med Azure Document database.

Magnus primære interesse er indenfor Backend, men han har også fokus på at skabe brugervenligt UI.

Mathias Bidstrup startede i en virksomhed som webudvikler efter afsluttet grundforløb men tilbragte kun 2 måneder hos dem, hvorefter han startede i SKP, som han har siddet hos til dags dato. Under hans tid i SKP har han udviklet kompetencer inden for brugerservice og infrastruktur. Udover dette har Mathias en kompetence i design og opbygning af frontend, som han har arbejdet med som fritidsinteresse de sidste 5 år.

Niclas Vernersen har erfaring med projekt og porteføljestyring gennem sin læreplads, som er et konsulenthus, der er specialiseret i netop projekt og porteføljestyring med Microsoft teknologier såsom Microsoft Project, Teams, planner og lignende. Niclas sidder som in-house udvikler og har derigennem erhvervet erfaring med agil udvikling og udvikling i sprog som C#, VBA, Javascript, mf. Niclas' stærkeste side ligger primært i backend og har mest erfaring med udvikling af services eller plugins, der modificerer standardfunktionaliteten i andre applikationer.

Tobias Mejnertsen har flere års erfaring som udvikler, hvor han har arbejdet med C#, XAML, MVVM kodestruktur og MSSQL. De første par år i praktikforløbet arbejdede han primært med XAML (GUI), hvorefter han kom længere ned i backend udvikling, og det har skabt stor interesse. Tobias har opnået denne erfaring gennem sin læreplads, hvor han sidder i et internt udviklingsteam.

2 Indledning

Ud i det blå er produktet af et projekt af samme navn. Dette er en rapport, der beskriver forløbet for projektet, udvikling af projektet, besvarelse af problemformuleringen samt en diskussion og en perspektivering. Rapporten omhandler langt de fleste aspekter af, hvad der skal til for at komme fra problem til endeligt produkt, som kunne være en løsning på problemet.

2.1 Problemformulering

Det kan være svært at finde ud af, hvor det er muligt at gebærde sig i det fri. Det kan gøre det demotiverende at komme *Ud i det blå* og resulterer i, at man bliver hjemme i stedet for. Er man for eksempel hundeejer, forældre eller friluftstræner, er det derfor vigtigt, at man har kendskab til mulige udendørsaktivitetsområder og/eller begivenheder i et givent område. For ikke at risikere at man bliver skuffet, er det rart at kunne se, hvordan stedet ser ud, og høre hvad andre har at sige om det, inden man tager afsted. Det er besværlig information at opstøve, hvilket ofte kan resultere i, at udendørsprojekter eller aktiviteter går i vasken. Derudover kan det være svært at finde gode arealer, hvis man er væk hjemmefra eller ønsker at differentiere og udforske nye steder, områder og muligheder. Det er desværre sådan, til trods for at udendørs aktiviteter, sociale arrangementer og leg er rigtig godt og sundt for specielt børn og unge, men også voksne og mange kæledyr^{[1][2][3][4]}.

Dette danner baggrund for udarbejdelsen af produktet og har ledt os til følgende spørgsmål, som vil blive forsøgt besvaret gennem arbejdet med projektet.

- Hvordan kan et system udvikles responsivt, således at det præsenteres flot og er tilgængelig på enhver enhed med internet og browser?
- Hvilke UI- og UX-overvejelser skal gøres for at opnå et intuitivt design i systemet?
- Hvordan kan det sikres, at de brugerindmeldte data der præsenteres i systemet, er pålidelige?
- Kan systemet designes således, at det i højere grad motiverer brugere til at komme *Ud i det blå*?

^[1] <https://goo.gl/3u3Qfm> - Artikel - Center for sundhed

^[2] <https://goo.gl/snNZtC> - Artikel - Aktiv træning

^[3] <https://goo.gl/VgEq1E> - Artikel - Idenyt

^[4] <https://goo.gl/FztBoL> - Artikel - Vejle amts folkeblad

3 Projektbeskrivelse

Ud i det blå er en webapplikation udviklet i ASP.NET efter MVC-strukturen.

Applikationen *Ud i det blå* er til mennesker, der ønsker at udforske muligheder for at gebærde sig udendørs i diverse sammenhæng såsom leg, træning og/eller luftning af kæledyr. Applikationen gør det lettere for brugere af systemet, for eksempel, at finde en legeplads i nærheden, og vurdere om legepladsen er et sted de har lyst til at besøge. Tanken med *Ud i det blå* er at kunne finde udendørsområder, og se billeder samt bedømmelser heraf, i stedet for at skulle bruge unødvendig energi og/eller penge på at opsøge udendørsområder og håbe på, at de er i ordentligt stand og lever op til forventningerne.

3.1 Systemkort og flowchart

For at give et dybere indblik i hvordan applikationen hænger sammen, har teamet udarbejdet et systemkort og et flowchart, der beskriver flowet i applikationen. Dette skal sammen med produktbeskrivelsen og systemflowet, som kan ses i bilag 1 Kravspecifikation, give læseren en bedre forståelse af applikationens opbygning.

Flowchartet kan ses i bilag 6 – Flowchart.

Systemkortet kan ses i bilag 7 – Systemkort.

4 Teknologier

4.1 Versionskontrol

Formålet med versionskontrol er, at det hjælper ens udviklingsteam med at holde et struktureret overblik over ændringer over en periode, hvor det er muligt at udarbejde forskellige opgaver på samme tid uden at påvirke eller overskrive sine kollegers arbejde. Dette foregår ved, at alle parter har en lokalversion af koden, hvor personen kan udarbejde sine forbedringer eller ændringer inden personen vælger at tjekke det ind i versionskontrollen og efterfulgt merge det ind i hoveddelen^[5]. Alt koden er centraliseret på en server og deles op i forskellige changeset, der gør det muligt altid at kunne rulle tilbage til en bestemt version, hvis der er foretaget nogle ændringer, der fejler eller en anden årsag til at tjekke den tidligere kode. At koden er delt op i forskellige dele, gør at ens kode lagres sikkert og altid kan rulles tilbage.

4.1.1 Hvordan fungerer TFS

TFS (Team Foundation Server) kan styres på mange forskellige måder. *Ud i det blå* har struktureret deres versionskontrol ved at kreere branches til alle parter, der er personens egen lokale version af koden, hvor det kun er den person, der foretager ændringer. Når der er blevet tilføjet nogle nye funktionaliteter eller ændringer, checkes det ind i personens versionskontrol under sin branch. Her tilknyttes en beskrivende kommentar til changesettet, så det er muligt på et senere tidspunkt at kunne rulle tilbage og se ændringer på en bestemt ændring og/eller funktionalitet ud fra kommentaren^[6]. Efter der er blevet checket ind i versionskontrollen, merges det op i hoveddelen som kaldes main, hvor TFS sørger for, at det

^[5] <https://goo.gl/kLbMTp> - Understand Merging - Microsoft

^[6] <https://goo.gl/2BRPWn> - Find and View Changesets - C# Corner

bliver merget på korrekt vis. Hvis der er foretaget ændringer på samme kodelinje, vil der opstå en Merge conflict, hvor der kræves en manuelt merge. Dette gøres ved at vælge den linje med de korrekte ændringer. Herefter checkes det nye changeset, som indeholder den mergede version, ind i main.

4.1.2 Hvordan gavner TFS *Ud i det blå*

Versionskontrollen gavner projektet ved at kunne tracke samt identificere, hvem der ændrer, hvilken del af koden og skaber et overblik over ændringer, der er foretaget i kodebasen. Dette muliggør hurtigt at kunne identificere et problem på baggrund af eventuelle tilkomne fejl. Derudover gavner det teamet, da alle parter har deres egen "sandbox", hvor der kan eksperimenteres med forskellige ideforslag eller kodeløsninger uden påvirkning af de andres arbejde. Når den nye kode så er gennemtestet og konkluderes som fungerende uden problemer, kan det checkes ind i versionskontrollen og derefter ind i hoveddelen, hvor den fælleskode ligger. I tilfælde af, at der uheldigvis er blevet pushet fejl i fælleskoden, kan det relativt nemt identificeres grundet de forskellige beskrivelser på changeset, og samtidig er det muligt at sammenligne to forskellige versioner, se forskellene og derfra udlede fejlen.

4.2 ASP.NET Core -MVC

Systemet er en ASP.NET Core 2.1 webapplikation som i sin enkelthed er en hjemmeside. ASP.NET Core er et open-source framework til udvikling af webapplikationer. Det oprinder fra ASP.NET som er et closed-source framework. ASP.NET Core er derfor i konstant udvikling og udgiver nye versioner og features rigtig ofte, hvorimod ASP.NET er mere stabilt og nye versioner er sjældnere end ASP.NET Core^{[7][8]}. Som udgangspunkt benyttes disse frameworks til det samme, dog er de meget forskellige i deres understøttelse af andre værktøjer og frameworks samt understøttelse af hosting og hardware^[9]. For eksempel kan en ASP.NET Core applikation ikke benytte sig af Entity Framework, men kun Entity Framework Core. I ASP.NET kan man også benytte sig af LINQ to SQL, det kan man ikke i ASP.NET Core. Desuden har Entity Framework mulighed for at benytte sig af Mange-Til-Mange relationer indbygget ved code-first design af modellerne, hvilket ikke er tilfældet for Entity Framework Core. En fordel ved ASP.NET Core er, at applikationerne kan hostes på et bredt spektrum af både operativsystemer og hardware, hvorimod ASP.NET kun kan hostes på Windows Server og skal være på understøttet hardware.

ASP.NET Core kan udvikles i sproget C#, hvilket spiller en stor rolle for teamet bag *Ud i det blå*. Det er nemlig et af de sprog, teamet har mest erfaring med, når det kommer til backend programmering. ASP.NET Core benyttes ofte, ligesom i *Ud i det blå*'s tilfælde, sat op i en MVC-struktur. Det er fordi, det er en rigtig smart måde at dele koden op og definere hvilke dele af koden, der gør hvad, for hvem/hvad og hvornår.

^[7] <https://goo.gl/xjBDg3> - .NET Framework Versioner og Dependencies - Microsoft

^[8] <https://goo.gl/syqcMt> - .NET Core Versioner og historie - Microsoft

^[9] <https://goo.gl/STiqVF> - Forskellen på ASP.NET 4 og ASP.NET Core

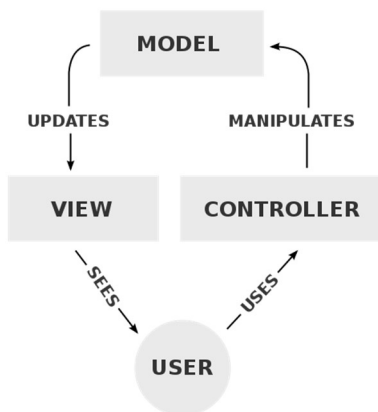
Ud i det blå er bygget op efter MVC(*Model-View-Controller*)-strukturen^[10] som bygger på, at de tre dele gør nogle forskellige ting for hinanden og på den måde arbejder sammen for at give brugeren den ønskede funktionalitet på hjemmesiden.

Model: Er den del der definerer, hvordan modellerne ser ud og arbejdes med både i viewet og controllerne, som typer, og i databasen, som entities, såfremt de altså eksisterer. I nogle tilfælde udvikles en hjemmeside både med views og en database, i andre tilfælde uden en database. Der er også tilfælde, hvor der ikke er nogle views, for eksempel, når der udvikles et API^[11] med MVC-struktur.

View: Er den del der indeholder de visuelle elementer, det er her frontenden udvikles. Det er den del, der bliver præsenteret for brugeren, når hjemmesiden tilgås. Denne del kan i ASP.net Core frameworket samtidig benyttes til at eksekvere noget backendkode, hver gang en specifik side bliver hentet af en bruger. På baggrund heraf kan indholdet af siden variere. Til dette benyttes MVC-views med RazorMarkup, som er HTML-sider af typen cshtml, der kan indeholde backend kode, og som når de bliver requestet, compiles og returneres til brugerens browser som et almindelig HTML-dokument.

Controller: Er den del der indeholder logikken, altså den rå backend kode. Her defineres hvad, hvornår og hvordan ting sker, når man navigerer rundt og benytter funktionaliteterne på hjemmesiden. Det er også her, der defineres, hvordan specifikke funktioner skal benytte modellerne til at rette, slette eller tilføje data i databasen.

Helt simpelt set kan MVC-strukturen illustreres med følgende figur:



Figur 1 MVC-struktur

^[10] <https://goo.gl/xEppR7> - MVC - Wikipedia

^[11] <https://goo.gl/kERHwU> - API Wikipedia

4.2.1 RazorMarkup

Razor er en markup syntax, der tillader at integrere serverbaseret kode (Visual Basic og C #) i MVC-views. Server-side kode kan skabe dynamisk webindhold i realtid, mens en webside er skrevet til browseren. Når en webside kaldes, udfører serveren server-side koden inde på siden, før den returnerer siden til browseren. Razor er baseret på ASP.NET og er designet til at skabe webapplikationer.

4.2.2 Backend

4.2.2.1 Database og Entity Framework Core

Valget af database er faldet på at være en SQL Server database. Dette skyldes, at Entity Framework Core og Azure services fuldt ud understøtter brugen af denne type database. Derudover er det den database type, som teamet har mest erfaring med.

Entity Framework Core er den teknologi, der, i *Ud i det blå*, benyttes til at kommunikere med databasen. Entity Framework Core er det man kalder en ORM (Object-Relational Mapper). Det betyder, at det er i stand til at binde typer i applikationen sammen med tabeller i databasen. Disse bindinger kaldes for entities i Entity Framework Core. På den måde kan *Ud i det blå* kommunikere med databasen ved blot at oprette en ny entity i koden, give de properties der skal være, og programmatisk opdatere databasen. Entity Framework Core sørger for, at der bliver genereret og eksekveret queries mod databasen på baggrund af de ændringer, man laver på de entities, der arbejdes på i *Ud i det blå*.

For at Entity Framework Core i *Ud i det blå* kan kommunikere med databasen, benytter den en såkaldt DbContext. Det er i udgangspunktet en klasse, som indeholder de informationer, der skal bruges omkring entities og databasen for at kunne foretage ændringer heri. Derudover indeholder den de metoder, der er tilgængelige for at foretage disse ændringer. *Ud i det blå's* DbContext kan ses i bilag 2 kildekodeindeksering, afsnit 2.16 uidbDbContext.cs. De klasser, der skal benytte Entity Framework Core til at kommunikere, hvilket i MVC-strukturen primært er controllerne, skal derfor have en instans af klassen for at kunne benytte konteksten til at kommunikere med databasen.

4.2.2.2 Databasekald

Der er opsat en DbContext i en separat klasse, der gør det muligt at kalde klassen og derfra kalde de ønskede metoder. Det gælder alt fra at loade alle opslagene ud, slette eller ændre specifikke opslag. På den måde gøres det simpel at lave forskellige udtræk og ændringer ved et enkelt kald. Der er fremvist to eksempler nedenfor for at vise, hvor simpelt det kan gøres.

Den nedenstående kodelump viser, hvordan Entity Framework Core benyttes til at udtrække en liste af specifikke entities fra databasen. I dette tilfælde gøres det ved asynkront at kalde DatabaseContexten (`_context`) efter alle entities af typen `OutdoorActivityArea` og konvertere resultatet til en liste af `OutdoorActivityAreas`. Der er i dette tilfælde også lavet nogle Includes, som benyttes for at sikre, at de properties også bliver loadet, fordi lazy loading ikke udføres som standard for de properties^[12]. Eksemplet kan ses i Bilag 2 Kildekodeindeksering, afsnit 3.3.3 `Index()`.

```
var outdoorActivityAreaIndexVM = new OutdoorActivityAreaIndexViewModel
{
    AreaList = await _context.OutdoorActivityAreas
        .Include(o => o.PrimaryImage)
        .Include(o => o.Ratings)
        .ToListAsync(),
    Filters = new OutdoorActivityAreaFilterModel()
};
return View(outdoorActivityAreaIndexVM);
```

Nedenstående eksempel viser, hvordan en specifik entity af typen `OutdoorActivityArea` bliver slettet. Først findes entiteten på baggrund af et givent id. Herefter tjekkes der for, om den givne entity er blevet oprettet af den user, der forsøger at udføre sletningen. Hvis det er sandt, benyttes DatabaseContexten til at slette den fundne entity, hvorefter den givne ændring for entity-samlingen gemmes i databasen via DatabaseContexten. Eksemplet kan ses i Bilag 2 Kildekodeindeksering, Afsnit 3.2.15 `DeleteConfirmed()`

```
var outdoorActivityArea = await _context.OutdoorActivityAreas.FindAsync(id);

if (outdoorActivityArea.UserWhoCreatedId == ClaimsHelper.GetUserGuid(User))
{
    _context.OutdoorActivityAreas.Remove(outdoorActivityArea);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

4.2.2.3 Kodeopdeling – (Uden for MVC-strukturen)

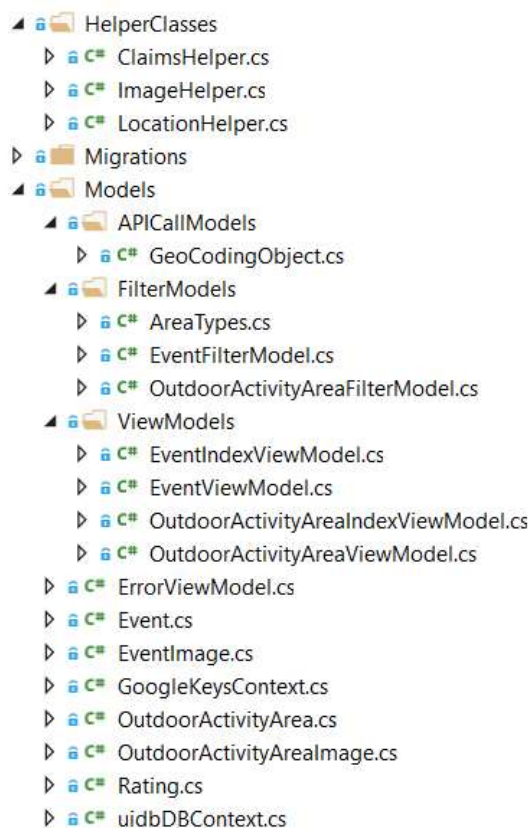
Kodeopdeling er en vigtig faktor, ikke for brugerne af produktet, men for udviklerne af produktet. En god kodeopdeling igennem hele udviklingsfasen og programmet gør det meget nemmere at overskue sin kode, samt finde og identificere hvilket element, der tilhører hvilken del af koden. Det er noget, der skal udføres fra start, da det tager lang tid at refaktorere på et senere tidspunkt, og det gør også selve udviklingsfasen meget hurtigere, hvis tankegangen er opsat på det fra start af.

Reglen er så vidt muligt at lave løst koblet kode. Der er etableret nogle classes, der indeholder fælles metoder, som bruges flere steder i applikationen. Det giver færre linjer kode og er langt nemmere at overskue for udvikleren. Det er vigtigt, at når der laves løst koblet kode, at metoderne laves så generiske, at de kan benyttes i alle de tiltænkte

^[12] <https://goo.gl/buVXAG> - Lazyload i Entity Framework – Entity Framework Tutorials

sammenhæng. Det har også den betydning, at udvikleren kan lave en ændring i metoden, hvorefter det automatisk har betydning for alle de steder, metoden bliver brugt.

Kodeopdelingen er lavet som vist på nedenstående billede. De forskellige typer af classes er opdelt i mapper, der igennem navnet beskriver, hvilken type de er. For eksempel er der en mappe med *HelperClasses*, der bliver genbrugt i de forskellige controllers og MVC-views igennem RazorMarkup. Der er også såkaldte undermapper, et eksempel på dette er mappen *ViewModels*. *ViewModels* er en undermappe i mappen *models*. *Models* indeholder systemets modeller, og *ViewModels* undermappen indeholder en specifik type af modeller, som bruges til enten at udvide en model, så den understøtter specifikke properties, der skal eksistere i viewet, eller til bare at indeholde nogle properties der skal benyttes i et view uden understøttelse af en reel MVC-model.



4.2.2.4 Third-party API'er

4.2.2.4.1 DAWA API

Webapplikationen gør også brug af API-kald til at hente data fra DAWA. DAWA står for Danmarks Adressers Web API, hvor Styrelsen for Dataforsyning og Effektivisering udstiller adressedata fra hele Danmark. Udover adresser og postnumre stilles politikredse, sogne og regioner også til rådighed.

Data, der hentes, fra DAWA er open-source og kræver kun API-URL'en for at hentes^[13]. *Ud i det blå* gør brug af DAWA til det formål at lave autocomplete forslag, når en bruger indtaster en by eller adresse. Til at få fat i byer hentes postnummer data fra DAWA, som indeholder både postnummer og bynavn, hvorefter postnummeret filtreres fra. Dataene, som DAWA returnerer, er af formatet JSON som frontenden håndterer og bearbejder. Brugen af DAWA i *Ud i det blå* kan ses i Bilag 2 Kildekodeindeksering, afsnit 6.1 DAWA Api kald funktioner.

JSON står for JavaScript Object Notation, og er et dataformat, som er let at læse og skrive for et menneske, men også nemt for en computer at generere og manipulere^[14]. JSON's dataformat består af par af nøgler og værdier, som herudover også kan samles i arrays af disse. JSON håndteres dog som tekst, til trods for at dataformatet er delt op som ovenstående beskrevet.

JSON-dataene i *Ud i det blå* ser således ud ved et opslag på en adresse til brug i autocomplete. Dette er et eksempel på det JSON-data, som DAWA returnerer.

```
{
  "tekst": "Nielsine Nielsens Vej 41A, 2400 København NV",
  "adresse": {
    "id": "067464df-c79f-48b0-8750-5c1efc089494",
    "href": "https://dawa.aws.dk/adresser/067464df-c79f-48b0-8750-5c1efc089494",
    "vejnavn": "Nielsine Nielsens Vej",
    "husnr": "41A",
    "etage": null,
    "dør": null,
    "supplerendebynavn": null,
    "postnr": "2400",
    "postnrnavn": "København NV",
    "stormodtagerpostnr": null,
    "stormodtagerpostnrnavn": null
  }
},
```

Der er valgt at bruge DAWA i *Ud i det blå*, fordi API-dataene er udstedt af en offentlig instans, så det kan forventes, at adressedataene er korrekte. Herudover har DAWA lavet et JavaScript-bibliotek, der gør det muligt at lave autocomplete ved adresseindtastning som *Ud i det blå* gør brug af. Det er således med til at hjælpe brugeren med at indtaste korrekte data.

^[13] <https://goo.gl/7McENA> - Danmarks Adresser - Dawa

^[14] <https://goo.gl/kKymXc> - JavaScript Object Notation - Json

4.2.2.4.2 Google Maps API

Hjemmesiden benytter sig af Google Maps Geocoding API, der blandt andet kan omregne koordinater til adresse og omvendt. Først deklareres API-nøglen i appsettings.json filen, der hemmeliggøre nøglen for brugerne igennem kildetekst, så nøglen ikke kan stjæles.

```
"GoogleKeys": {  
  "MapsAPIKey": "AIzaSyCfBOCnNCJH800IfIqfVLxHn1_DHHIr-fi0"  
},
```

For at skabe forbindelsen til nøglen, opretter vi en ContextModel med property navnet *MapsApiKey*, hvorefter modellen kan initialiseres i en vilkårlig controller og bruges til API-kaldene. Se bilag 2 Kildekodeindeksering, afsnit 2.12.1 Attributter.

```
5 references | Niclas Rafn Topp Vernersten | 1 author, 1 change | 2 incoming changes  
public class GoogleKeysContext  
{  
    6 references | Niclas Rafn Topp Vernersten | 1 author, 1 change | 2 incoming changes | 0 exceptions  
    public string MapsApiKey { get; set; }  
}
```

Her initialiseres modellen med API-nøglen og kan derfra bruges igennem den deklarerede property *googleKeysContext*. Se Bilag 2 Kildekodeindeksering, afsnit 3.2.2 Constructor.

```
private readonly UidbDBContext _context;  
private readonly GoogleKeysContext _googleKeysContext;  
  
0 references | Niclas Rafn Topp Vernersten | 1 author, 1 change | 2 incoming changes | 0 exceptions  
public OutdoorActivityAreasController(UidbDBContext context, IOptions<GoogleKeysContext> googleKeysContext)  
{  
    _context = context;  
    _googleKeysContext = googleKeysContext.Value;  
}
```


I den nedenstående metode bruges API-nøglen. Først deklareres den korrekte URL til at arbejde med API'et, hvorefter der oprettes en HttpClient, som benyttes til at lave API-kald. Heri sættes der i headeren, at det svarformat der accepteres, er JSON.

```
string retCoords = "";

string urlParameter = "";
string mapsGeoCodeBaseUrl = "https://maps.googleapis.com/maps/api/geocode/json";
string staticParameter1 = "?address=";
address = address.Replace(' ', '+');
string staticParameter2 = "&key=";

urlParameter = staticParameter1 + address + staticParameter2 + apiKey;

HttpClient client = new HttpClient();
client.BaseAddress = new Uri(mapsGeoCodeBaseUrl);

// Add an Accept header for JSON format.
client.DefaultRequestHeaders.Accept.Add(
    new MediaTypeWithQualityHeaderValue("application/json"));
```

Til sidst returneres værdierne og læses ind i programmet som et object af den type, som *Ud i det blå* har defineret ud fra svarstrukturen på netop denne type kald fra API'et. Se Bilag 2 Kildekodeindeksering, 1.4.1 GetCoordinatesFromAddress().

```
// List data response.
HttpResponseMessage response = client.GetAsync(urlParameter).Result; // Blocking call! Program will wait here until
if (response.IsSuccessStatusCode)
{
    // Parse the response body.
    var dataObjects = response.Content.ReadAsAsync<GeoCodingObject>().Result; //Make sure to add a reference to Sys
    retCoords = dataObjects.results[0].geometry.location.lat + ", " + dataObjects.results[0].geometry.location.lng;
}
```

4.2.2.5 Identitets server

Til brugerhåndtering og de funktioner det indebærer såsom log ind, log ud, ændring af brugerinformationer samt validering i programmet, gør *Ud i det blå* brug af Azure servicen B2C (Business To Consumer). Azure B2C er en variation af Active Directory, der fungerer som en cloudbaseret identitetsserver, der hostes i Azure, hvor eksterne brugere kan oprette sig som en bruger med forskellige typer af udbydere^{[15][16]}. Azure B2C er baseret på Active Directory, således at håndtering af brugere fungerer på samme måde. Det er derfor muligt for oprettede brugere af systemet at oprette et login, logge ind, logge ud og redigere deres profil, uden applikationen skal håndtere eller har adgang til brugerens password. Da denne service som udgangspunkt er en identitetsserver, er disse ekstra funktionaliteter, som udbydes fra servicen.

^[15] <https://goo.gl/Ju8QVa> - Azure Business To Client - Microsoft

^[16] <https://goo.gl/9aQpDJ> - Azure Business to Client - Microsoft

En identitetsserver er nemlig som udgangspunkt en database med et API, der muliggør at verificere og/eller returnere efterspurgt identitetsdata, hvorefter programmer selv skal stå for at vedligeholde sessionsdata, som indeholder brugerverificering. Når en bruger prøver at udføre en controller handling, der kræver at brugeren er logget ind, udføres der et tjek i metoden om brugeren er logget ind. Udover at Azure B2C leverer en valideringsmetode, udstilles der også informationer omkring brugeren, der er logget ind. Disse informationer ligger i såkaldte claims, som er udfyldt via Azure B2C's log ind- og log ud politikker^[17]. Denne information bruges i *Ud i det blå* til at knytte brugere sammen med områder, begivenheder eller bedømmelser ved oprettelse af en ny entity, således webapplikationen kan håndtere, hvem der må redigere.

Brugen af Azure B2C i *Ud i det blå* sikrer webapplikationen imod, at uvedkommende personer nemt kan tilgå backenden og gøre skade, fordi der sker en validering af forbindelsen. Fordi der benyttes Azure B2C, overdrages håndteringen af sikkerhed i henhold til brugere, brugerdata og brugergodkendelse i webapplikationen over til en virksomhed som, i hvert fald med *Ud i det blå*'s ressourcer, leverer en bedre beskyttelse af brugernes data. Derudover får webapplikationen et velfungerende loginsystem.

4.2.2.6 Azure blobstorage

Til håndtering af custom login UI i forbindelse med Azure B2C gør *Ud i det blå* brug af Azure blob storage til at hoste CSS og HTML i skyen, som Azure B2C bruger til at vise tilpasset UI, når en bruger skal logge ind, logge ud eller ændre loginoplysninger. Blob storage er en cloudbaseret database til at opbevare blob data, hvilket er store binære dataobjekter, som for eksempel kan være billeder, filer eller større ustruktureret data. Disse data kan kun tilgås ved http-requests til en URL. Hvert element i databasen har en unik URL, som gør det muligt at få fat i det specifikke element, så længe URL'en kendes. Måden hvorpå blobstorage opbevarer dataer er magen til mappestrukturen i et filsystem. Således ligger gemte objekter i en eller flere containere/foldere, der er knyttet til en konto^[18]. Den måde *Ud i det blå* bruger Azure blobstorage på er ved, at HTML og CSS er gemt i en blobstorage, som Azure B2C henter og renderer.

Ud i det blå har valgt at bruge Azure blobstorage, fordi det er nemt og sikkert at udstille filer, samtidig med at der gøres brug af Azure B2C, så alt cloudbaseret teknologi holdes i det samme cloudmiljø.

^[17] <https://goo.gl/iNRXZz> - Azure b2c Tokens - Microsoft

^[18] <https://goo.gl/8qaCuG> - Azure Blob Storage - Microsoft

4.3 Frontend

4.3.1 MVC Views.

Til udviklingen af brugergrænsefladen gøres der brug af MVC-views. MVC-views har den fordel, at *Ud i det blå*'s brugergrænseflade separeres fra backend logikken, så der kan udføres ændringer på views uden berøring af backend logikken. MVC-views har den fordel, at de kan gøres strongly-typed ved at binde en model til viewet^[19]. Fordelen ved at binde en model til viewet er, at det er en fast type data, der kan benyttes og giver samtidig en form for datavalidering. Det gør, at det er muligt at videreføre data af den givne type videre fra en controller action til viewet. Derudover udstiller MVC også et shared layout-view, der bruges til at vise de samme elementer i alle views i webapplikationen. Det fungerer således, at der er et view ved navn `_Layout.cshtml`, der i virkeligheden er det view, der bliver kaldt fra controlleren. Det sker ved at `_Layout` viewet bliver injected med det, fra controlleren, "kaldte" view. `_Layout` indeholder så alle de elementer, der er i HTML-headeren og evt. nogle ting til bodyen, der altid skal være der. Inde i bodyen i `_Layout` viewet bliver der så eksekveret `RenderBody()` metoden som injecter HTML'en fra det "kaldte" view. Samlet ender det så med at give et fuldt HTML-dokument, som præsenteres for brugeren. I MVC er det også muligt at benytte sig af partial views. Partial views benyttes til at isolere specifikke komponenter i views og render dem efter behov^[20].

MVC-views gør også brug af Razor Markup. Razor syntaxen gør det muligt at modificere HTML-elementer via C# logik i renderingen af MVC-viewets HTML. Det er Razor, som tillader at viewet kan gøre brug af viewmodellen direkte i HTML'en^[21].

En måde at gøre brug af Razor kan således ses i *Ud i det blå*'s view `Details.cshtml`. Siden gør brug af Razor til at verificere, om brugeren er logget ind, og om brugeren er den som har oprettet udendørsområdet ud fra modelværdien `"UserWhoCreatedId"`. Det er en værdi, der ligger i den type, der er knyttet til viewet som strongly-typed view. Hvis værdierne er ens, fastslår Razor om viewet skal render en knap. Se bilag 2 Kildekodeindeksering, Afsnit 5.2.6 `Detail.cshtml`

```
@model UdIDetBlaa.Models.ViewModels.OutdoorActivityAreaViewModel
@if (User.Identity.IsAuthenticated)
{
    if (Model.Binding.UserWhoCreatedId == ClaimsHelper.GetUserGuid(User))
    {
        <button data-toggle="modal" class="btn btn-primary" style="justify-content: flex-end;"
    }
}
```

Kodestump 1 Kodeeksempel på implementeringen af MVC-view og Razor i Ud i det blå.

^[19] <https://goo.gl/cLQCbl> - Views in ASP - Microsoft

^[20] <https://goo.gl/KGnbYp> - Partial Views in ASP - Microsoft

^[21] <https://goo.gl/WqgHFY> - Razor Syntax i ASP - Microsoft

4.3.2 JavaScript

Til at udføre realtid manipulation af HTML-elementer, gør *Ud i det blå* brug af JavaScript. Programmeringssproget JavaScript gør det muligt at udføre klient-side kode, hvilket vil sige, at det ikke har nogen indvirkning i, hvad der sker serverside. JavaScript gør det også muligt at køre scripts i browseren eller som server-side programmering sammen med Node.js. Sidstnævnte gør *Ud i det blå* dog ikke brug af, men bruger JavaScript til at manipulere HTML-elementer. *Ud i det blå* gør også brug af JavaScript-biblioteket JQuery, der udvider sproget med flere funktioner og tilføjer også logiske metoder såsom at lave HTTP-kald. JQuery benyttes blandt andet i webapplikationen til at lave en autocomplete tekstboks til at indtaste byer i Danmark. Det sker ved hjælp af et HTTP-kald^[22] og en JQuery UI-manipulation. Metoden der gør dette hedder `searchPostnr()`, som bliver kaldt når siden loader. Denne metode tager imod CSS-klassen `".city"` som parameter. JQuery's funktion `ajax` bliver så brugt til at udføre et HTTP-get request, som returnerer postnumre og tilhørende bynavne i JSON-format. JQuery itererer over disse postnumre, separerer bynavnene fra og samler bynavnene i et array. Dette array benyttes så som kilde til den autocomplete funktionalitet som JQuery benytter til at tilføje til tekstboksen. Se Bilag 2 Kildekodeindeksering, afsnit 6.1 DAWA API kald funktioner.

```
searchPostnr('.city');|
//tager inputet fra elementet med klassen city
function searchPostnr(input) {
    //Opretter et Http get kald til Dawa Danmarks Adressers Web API
    $.ajax({
        cache: true,
        url: 'https://dawa.aws.dk/' + 'postnumre',
        dataType: "json",
        error: function(xhr, status, errorThrown) {
            var text = xhr.status + " " + xhr.statusText + " " + status + " " + error
            alert(text);
        },
        success: function success(postnumre) {
            var items = [];
            //hvert postnummers by sættes i et array og sorteres, så der ikke er dublikater af n
            $.each(postnumre, function(i, postnr) {
                items.push(postnr.navn);
            });
            var set = new Set(items);
            var array = Array.from(set);
            //Autocomplete funktionen tilføjer array til en forslags liste på html inputet
            $(input).autocomplete({
                source: array,
                autoFocus: true,
                minLength: 1
            });
            return false;
        }
    });
}
```

Kodestump 2 Kodeeksempel på en JQuery funktion der kalder et Api og manipulere et html element ud fra data.

Langt det meste af JavaScripten i *Ud i det blå* er samlet i en enkelt .js fil, dette er for at disse scripts bliver eksekveret på hver eneste side, eller fordi metoderne skal være tilgængelige i alle views. Frontend-logikken kunne laves rent i JavaScript, men det er fravalgt, fordi HTTP-kald fra JavaScript udstiller kode i browseren.

^[22] <https://goo.gl/XysfLk> - jquery

Det gør det muligt og nemt for uvedkommende at stjæle keys til betalte og/eller sikrede API'er. Derudover er ASP.NET Core MVC-webapplikationer rettet imod brugen af MVC-views og Razor til visning af HTML- og HTTP-kald til controlleren.

4.3.3 Bootstrap

For at gøre *Ud i det blå* responsivt således at webapplikationen ser godt ud på desktop og mobile enheder som tablets og telefoner, benytter *Ud i det blå* blandt andet Bootstrap. Bootstrap er et CSS-bibliotek, som er lavet til at skabe responsive hjemmesider med prædefinerede CSS-klasser til styling af blandt andet knapper, tekstbokse og modaler^[23]. Bootstrap er godt til responsivt design, da dens prædefinerede CSS-klasser primært bruger procenter til at definere elementers størrelse. Bootstrap indeholder også prædefinerede JavaScript-funktioner til oprettelse af interaktive billed-karruseller og fremvisning af modaler.

Bootstrap benyttes i *Ud i det blå* som basis for, hvordan webapplikationens udseende og responsive egenskaber er. For eksempel bruges klasserne Col- til at sætte en bredde på et element på baggrund af en vis procentdel af skærmens størrelse eller forældreelementets størrelse. Således opstår der en form for gittersystem.

Et eksempel på hvordan Bootstrap i særdeleshed bliver benyttet i webapplikationen er, når der benyttes modaler. Her benyttes der stort set kun Bootstrap-klasser og scripting til at vise en modal med en transitionseffekt, som åbnes af HTML-dataelementer. Se bilag 2 Kildekodeindeksering, 5.2.2 _Create.cshtml.

```
<div class="modal fade2" id="createActivityModalCenter" tabindex="-1" role="dialog" aria-labelledby="createActivityModalCenterTitle" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="createActivityModalLongTitle">Opret</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
```

Kodestump 3 Eksempel på brugen af Bootstrap i *Ud i det blå*.

En af de smarte ting ved at benytte et responsivt design er, at det giver mulighed for at skabe en bredere og dermed større brugerbase, da brugere ikke hindres af enheden eller operativsystemet der benyttes. *Ud i det blå* bruger dog også egne CSS-klasser til at tilpasse webbappens udseende og egenskaber efter behov.

^[23] <https://goo.gl/4ZLSqZ> - Bootstrap

4.3.4 Sass

Til udvikling af *Ud i det blå*'s egne CSS-klasser, gøres der brug af Sass som udvider CSS med variabler, arvinger af andre klasser og import af andre Sass-filer.

Det som *Ud i det blå* bruger Sass til, er muligheden for at compile Sass-filer ned til .css og .min.css^[24]. Filen af typen .min.css er en formindsket version af CSS-filen, som benyttes i en produktionsversion af appen for at øge performance, men benyttes ikke i udviklingsfasen da den er uoverskuelig at arbejde i, da den ikke indeholder nogen formatering. Derudover er det også Sass som Bootstrap bruger til at lave CSS-klasser i. Så ved at gøre brug af Sass følger webapplikationen den måde Bootstrap laver CSS på. I og med at Bootstrap er et af de mest brugte CSS-biblioteker, er *Ud i det blå* med til at følge en standard inden for UI-design.

4.4 Domæne, SSL og hosting

4.4.1 Hosting

For at en ASP.NET Core webapplikation kan blive tilgængelig for brugere, skal det hostes. Der eksisterer mange løsninger i mange forskellige prislejer, og inkluderer løsninger både on-premise og cloud. Der er mange styrker og svagheder på dette punkt. *Ud i det blå* benytter sig af en Azure Service ved navn "Web app", som er i stand til at hoste en webapplikation helt gratis til at starte med. Det sker på et gratis *.azurewebsites.net domæne med SSL. Denne løsning er en cloudløsning og er særligt attraktiv, fordi der er et meget overskueligt kontrolpanel til at styre, hvad der er hosted i servicen, og hvordan den service er sat op. Derudover er der ikke meget sikkerhed at tænke over, da det ved hjælp af nogle meget sikre standardindstillinger, helt automatisk er sat op til at være sikret mod langt de fleste serversideangreb. Langt de fleste af disse indstillinger kan selvfølgelig ændres, hvilket kan resultere i både bedre og ringere sikkerhed.

Derudover bruger *Ud i det blå* også en database til at holde på systemdata. Derfor bruger *Ud i det blå* også en service i Azure ved navn "SQL Server", som er en cloudløsning til hosting af database. Denne service kommer også i en gratis version for studerende, hvilket er rigtig praktisk til brug under udviklingsfasen. De samme ting der gør Web appen hos Azure sikker, gør sig også gældende for deres SQL Server Service. Det der er smart ved at have begge disse hosted i et Azure Directory er, at de er i fuld stand til at kommunikere med hinanden uden at databasen tillader nogle forbindelser udefra. Det gør sig dog kun gældende, medmindre man selv specificerer andet. Det gør altså databasen rigtig sikker.

Alle disse services i Azure kan også skaleres op og ned efter behov. Det vil sige, at de kan allokere flere ressourcer på forskellige hardware og gøres georedundant samt lave workload-balancing. Priserne varierer helt fra gratis til nærmest ubegrænset store beløber om måneden afhængigt af behov. Det har *Ud i det blå* benyttet sig af under udviklingsfasen, hvor der er blevet arbejdet med gratis prisleje på alle tjenesterne lige til det punkt, hvor det for alvor skulle ligne et produktionsmiljø.

^[24] <https://goo.gl/ZUDm6C> - SASS CSS Documentation

4.4.2 Domæne

Når et produkt skal hostes til produktion og være tilgængeligt for brugere, så er det godt at brande sit produkt ordentligt og udstråle professionalisme og enkelhed. Et klingende navn/domæne der er nemt at huske og let at skrive er alfa og omega, når et produkt skal brandes. *Ud i det blå* har selvfølgelig den ulempe, at det indeholder et å. Karakteren å fungerer ikke i sammenhæng med hostede domæner. Det er de fleste danskere heldigvis klar over og samtidig bestrider viden om, at et å blot byttes ud med aa. Til gengæld lyder det fængende at sige UdIDetBlaa.dk (*Ud i det blå* Dot Dk), det gør det let at huske og skrive ind i sin browser eller finde i en søgemaskine. Derfor har *Ud i det blå* valgt at købe domænet udidetblaa.dk. Dette domæne tilknyttes selvfølgelig til den Webapp *Ud i det blå* er hosted i. Det kræves dog, at der skrues op til et lidt højere betalt niveau på Webappen at benytte custom domæner.

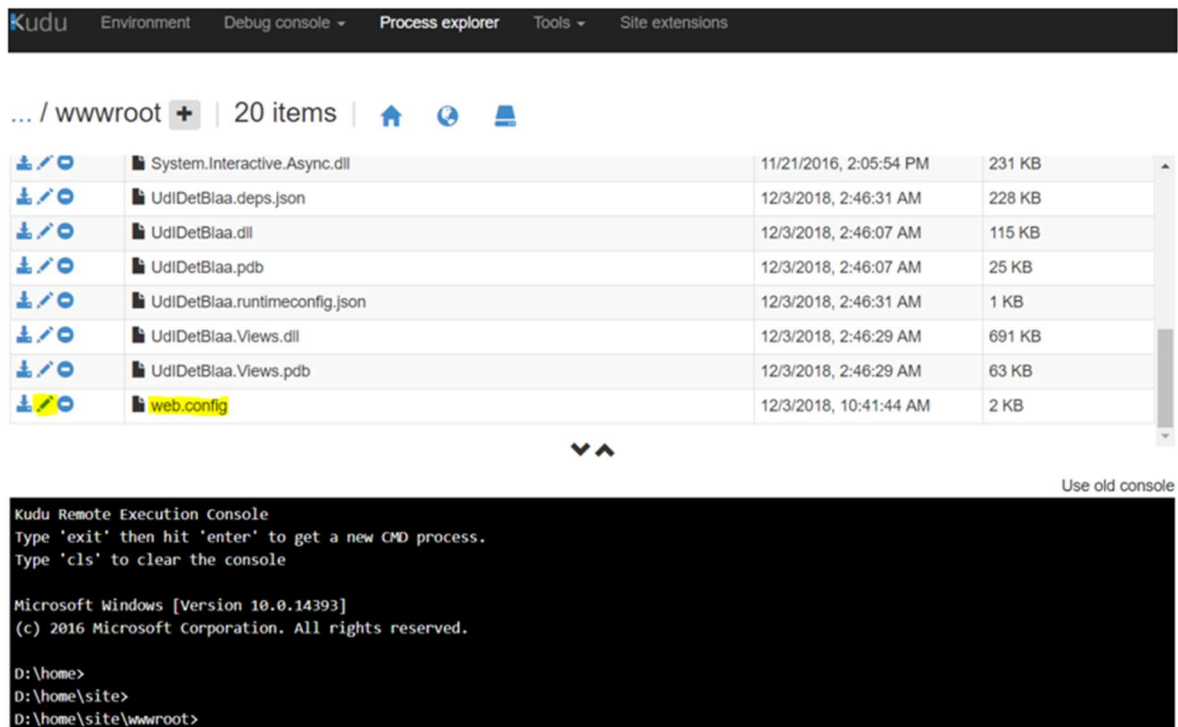
4.4.3 SSL

Når et custom domæne er knyttet til webapplikationen, og DNS records er sat til at pege på den rigtige IP, er det muligt for en bruger at tilgå *Ud i det blå* ved at skrive udidetblaa.dk ind i adressefeltet i sin browser. Brugere vil dog opdage at siden, fra en browsers synspunkt, ikke er sikker, hvilket skyldes at der ikke er nogen verificering af, at det der ligger hosted på domænet er det, som ejeren af systemet/domænet ønsker. Det kræver derfor et SSL-certifikat til domænet, som kan knyttes til webapplikationen. Sådant et certifikat har *Ud i det blå* anskaffet og knyttet til Webappen, så alle brugere trygt kan stole på, at det er det rigtige *Ud i det blå*, der er hosted på det domæne og ikke noget ondsindet. Siden er derudover kun mulig at tilgå via HTTPS, hvilket også er en indikator for et sikkert websted for brugere.

4.4.4 Permanent Redirect for statisk domænebrug i Azure

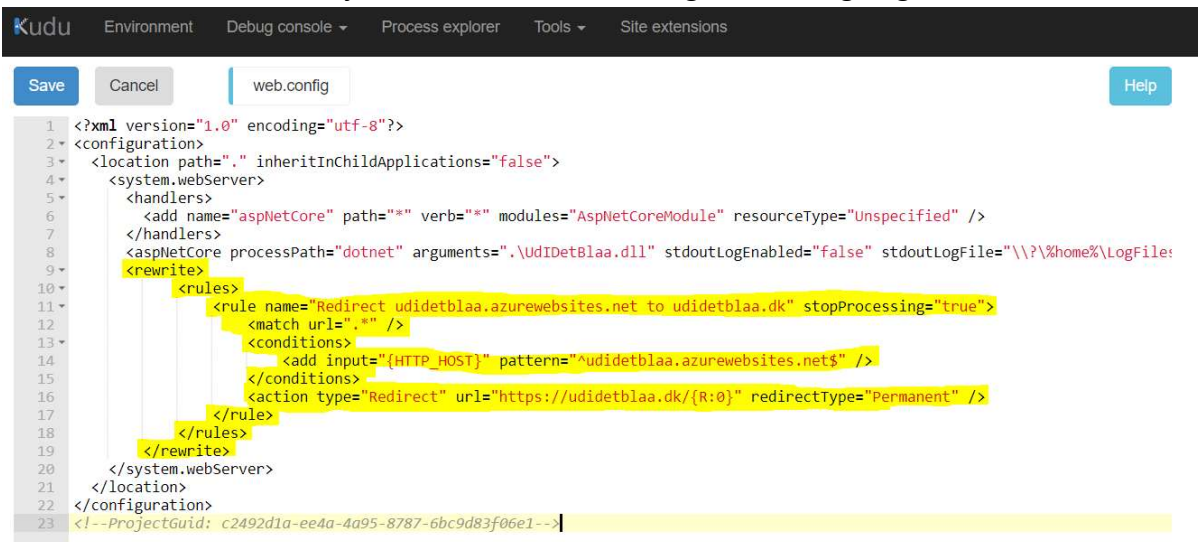
For at brugere ikke skal kunne blive forvirret over at det gamle domæne, udidetblaa.azurewebsites.net, stadig er aktivt, har vi *Ud i det blå* valgt at lave en 301-redirect. En sådan redirect er for mange bedre kendt som en "Permanent redirect". Det er altså en måde at give tegn til søgemaskiner, DNS og cache om at det ikke er meningen, at den redirect nogensinde skal laves om. Den redirect fører så alle adresser, der ligger i domænet "udidetblaa.azurewebsites.net" til den samme adresse, som de forsøger at angive, bare med "udidetblaa.dk" domænet i stedet for. Den er sat op direkte i serveren som Webappen kører på, ved hjælp af noget manuel kode i en hosting fil i wwwroot ved navn "web.config". Det er stort set ligeså let at tilgå i Azure som var det en on-premise server, der hostede det.

Herunder er et billede, der illustrerer kontrollen af filerne i wwwroot på hosten.



Billede 1 Illustration af kontrollen af filerne i wwwroot på hosten

Filen indeholder allerede nogle informationer, men er i stand til at tage imod flere parametre for hvordan webservicen skal opføre sig. Følgende er filen efter vores tilføjelse af 301-redirect hvor den tilføjede del er markeret med gul overstregning.



Billede 2 Web.config redigering direkte i Azure til at have 301-redirect

Denne ændring kan foretages direkte i administrationsværktøjets teksteditor. Efterfulgt af et restart af den webservice, er denne 301-redirect aktiv og ethvert forsøg på at tilgå `udidetblaa.azurewebsites.net` vil føre til tilsvarende URL i `udidetblaa.dk`. Det er også vigtigt for, at systemet kan køre med B2C identitetsserver på det nye domæne.

4.4.5 Azure og sikkerhed

Ud i det blå benytter flere forskellige funktioner i Azure, heriblandt SQL Server, B2C (Identitetsserver) og "Web app". Det har den betydning, at alt der hører til den hostede version af *Ud i det blå* ligger i skyen.

Azure er i sig selv en gigantisk cloudløsning, hvori det er muligt at hoste et utal af forskellige services og tilmed hele produktionsservere i form af virtuelle maskiner til selv at administrere, hvis det skulle være ønsket. Derfor tilbyder Azure mange forskellige sikkerhedsforanstaltninger, man kan benytte sig af, for at sikre de ting der er hosted i deres løsning. Heriblandt er firewall, virtuelle netværk, DDoS-beskyttelse, CORS, Directory V-LANs og mange flere. Nogle sikkerhedsforanstaltninger er dog meget ressourcekrævende og tilbydes derfor kun for services på nogle specielle prisniveauer. Som standard leverer Azure dog de vigtigste sikkerhedsforanstaltninger, det er også for at sikre deres egne datahuse og servere mod angreb, heriblandt kan DDoS beskyttelsen nævnes.

Azure tilbyder også servicen Application insights på app-services, for at monitorer fejl og performance på applikationer. Azure tillader også penetrations test, som gør det muligt at se, hvilke steder i Azure domænet, der mangler beskyttelse^[25]. De tjenester *Ud i det blå* gør brug af, er en firewall der sikrer, at det kun er services i netop *Ud i det blå's* Azure Directory, der har adgang til databasen. Det er en stor sikring af data, da det gør det rigtig besværligt at hacke sig ind i databasen. CORS (Cross-origin resource sharing) bruges til at åbne en sikker forbindelse imellem blobstorage og Azure B2C, fordi Azure B2C ligger i sit eget Azure Directory. Til at beskytte forbindelsen til App Servicen hvor hjemmesiden hostes, er der også slået Only HTTPS til, som Azure også tilbyder. Azure leverer også en anden type sikkerhed ved at stille en garanti på 99% opetid om måneden, så *Ud i det blå* hjemmesiden er kørende hele tiden^[26]. CORS er valgt, fordi det ikke er muligt at bruge blobstorage til Azure B2C Custom UI uden CORS er slået til.

Ud i det blå gør dog kun brug af standard sikkerhedstjenester på grund af projektets omfang, fordi mange af tjenesterne er tiltænkt til større domæner og løsninger end *Ud i det blå* er på nuværende tidspunkt, men også fordi teamet ikke har specielt meget erfaring med de mere avancerede sikkerhedstjenester.

^[25] <https://go.microsoft.com/fwlink/?linkid=2148261> - Sikkerhed med Azure - Microsoft

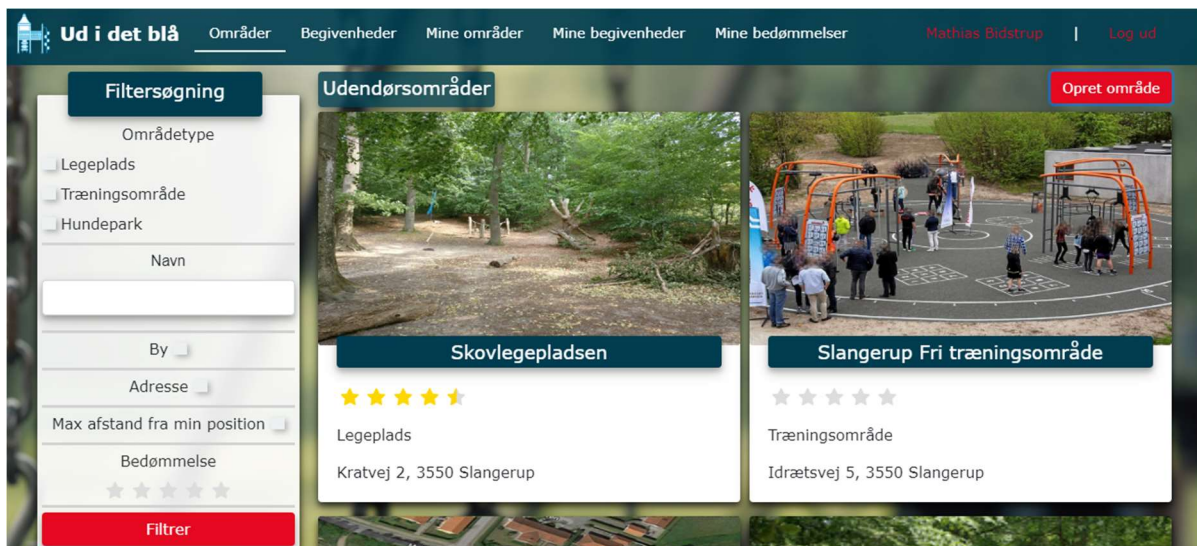
^[26] <https://go.microsoft.com/fwlink/?linkid=2148261> - Service Level Agreement med Microsoft - Microsoft

5 Beskrivelse af *Ud i det blå*

5.1 Brugerfladen

For at være brugervenlig er det vigtigt, at brugerfladen er let og intuitiv at navigere i. Af den årsag har EUX IVS valgt at designe applikationen så minimalistisk som muligt. Der er to farvetemaer i applikationen. "Områder" har ét tema, mens "Begivenheder" har et andet tema, for at gøre det nemmere at differentiere mellem de to dele af applikationen. Plads og mellemrum mellem elementerne på siden betyder, at applikationen føles åben og let. Tekstbokse, knapper og lignende ting der går igen, har samme udseende på hele platformen, for at gøre det overskueligt at forstå hvad det er for et element.

Et eksempel kan være "Index" siden til Udendørsområder, hvor man ser det tilhørende farvetema. Tekstboksene er fremhævet med skygger for at henlede brugerens opmærksomhed og for at fortælle, at her er der noget input, der kan udfyldes. Knapperne er designet således, at de er fremhævet for at vise, at der er funktionalitet bag.



5.1.1 Design

For at hjemmesidens design holder en rød tråd, har frontend-teamet udarbejdet en design guideline, der har til formål at opridse nogle rammer, som designet skulle holdes indenfor. Der er både beskrevet farvetemaer og styling af de forskellige elementer. Udover det der står beskrevet herunder, er Bootstrap blevet benyttet til at udvikle denne applikation.

Farver

Udendørs-område tema:

Primær farve: Sherpa Blue, Hex: #003D50, RGB: 0, 61, 80.

Begivenheder tema:

Primær farve: Verdun Green, Hex: #394F01, RGB: 57, 79, 1.

Tekstfelter

Box-shadow: Horizontal-offset: 6px, Vertical-offset: 20px, Farve: 20% sort transparent

Margin-bottom: 10px

Border: 1px, solid, Farve: Ghost, #CED4DA, 206, 212, 218

Border-radius: .25rem

Knapper:

Background-color: Farve: Red Ribbon, Hex: #E40920, RGB: 228, 9, 32.*

Box-shadow: Horizontal-offset: 6px, Vertical-offset: 20px, Farve: 20% sort transparent

*Ved :hover på knapperne skifter værdierne for farve i background-image til en mørkere rød.

Tjekbokse:

Background: #ECF0F1

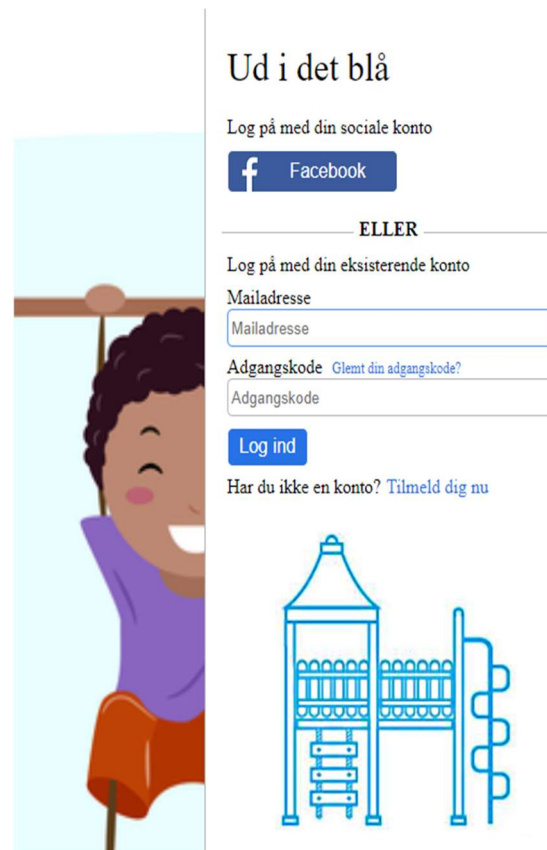
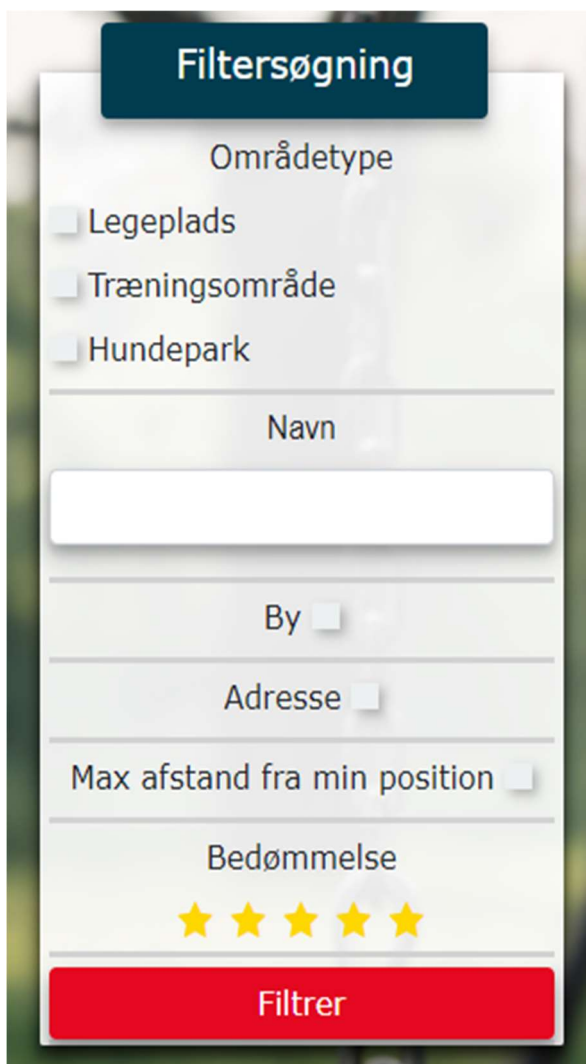
Width: 15px

Height: 13px

Box-shadow: Horizontal-offset: 6px, Vertical-offset: 20px, Farve: 50% sort transparent

5.2 Brugerfladens funktionalitet

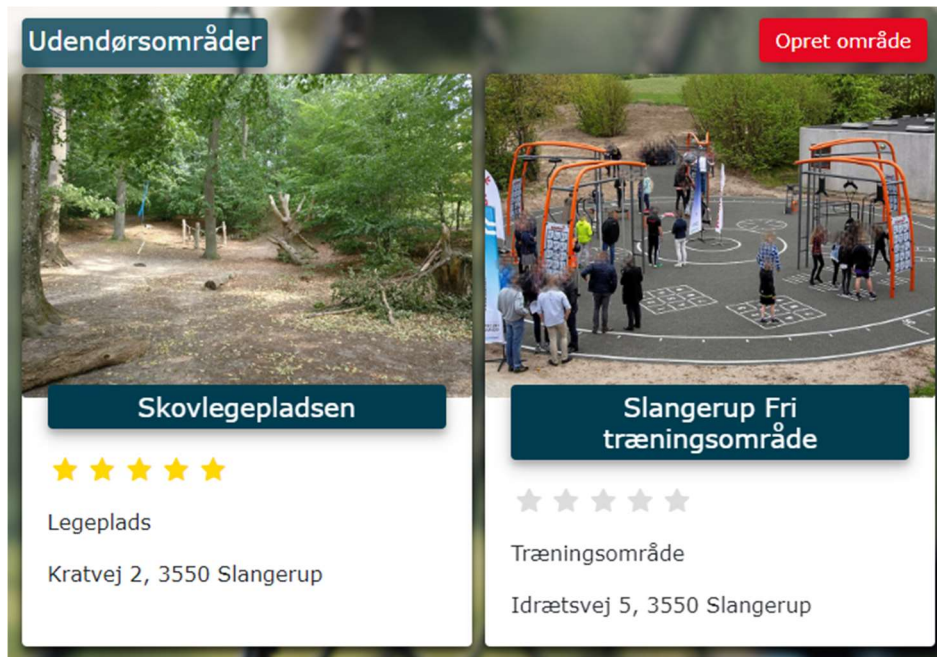
Efter at have landet på "Index" siden vil det næste oplagte skridt være at logge sig ind på siden. Dette kan gøres gennem Facebook, eller man kan oprette sin egen konto. Når der trykkes "Log in" på index siden, bliver man videresendt til denne Azure B2C side, hvor der kan logges ind med Facebook eller en lokal bruger.



På "Index" siden findes filter-boksen, der bruges til at filtrere de forskellige begivenheder og udendørsområder. Ved at trykke på "By" eller "Adresse" åbnes der op for flere muligheder at filtrere på. Et eksempel kan være, at hvis der trykkes på "Max afstand fra min position", åbnes der op for en "slider", der bruges til at afgøre størrelsen på brugerens søgeområde.

Skulle brugere benytte sig af en mobil eller anden enhed med en lille opløsning, vil filterboksen kunne åbnes ved at trykke på en knap i bunden af siden.

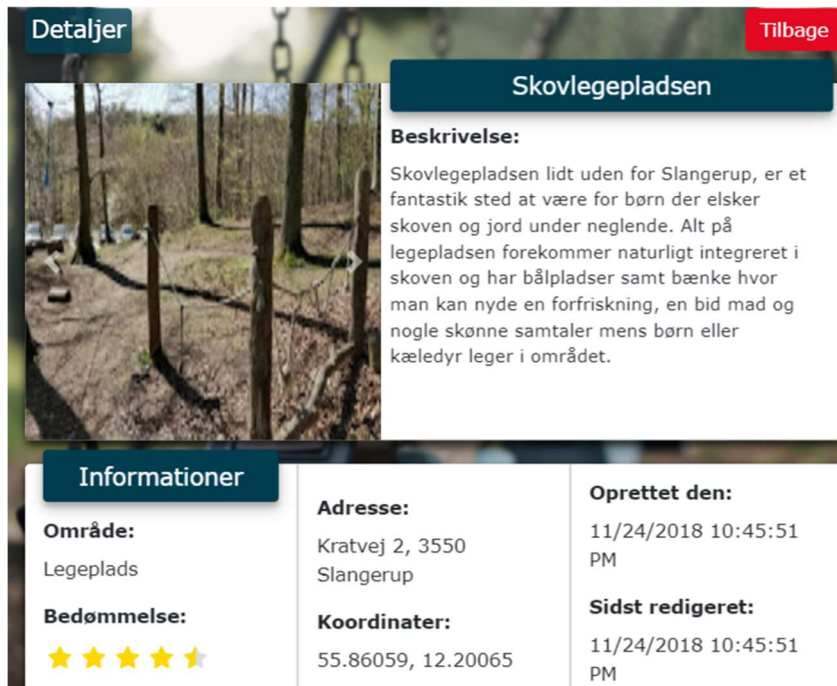
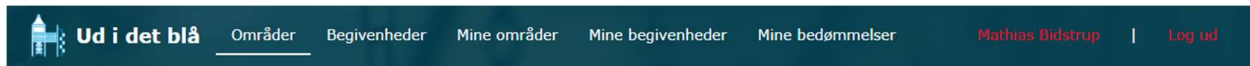
I midten af siden findes de forskellige områder eller begivenheder, her får brugeren vist det primære billede, bedømmelsen, typen og adressen tilhørende den begivenhed eller det udendørsområde. Dette bliver præsenteret i et "Card", og hvis brugeren "hover" over billedet med sin cursor, vil billedet blive fremhævet. Her får brugeren muligheden for at oprette et område eller en begivenhed, der derefter vil blive vist ligesom de områder, der ses på billedet nedenunder. Trykkes der derimod på billedet på et af disse "Cards" vil brugeren få vist Details siden, hvor brugeren kan finde detaljer omkring det område eller den begivenhed.

The image shows a 'Create' (Opret) modal form. It contains the following fields: 'Aktivetsnavn' (text input), 'Områdetype' (dropdown menu with 'Legeplads' selected), 'Adresse' (text input), 'Koordnater' (text input), 'Beskrivelse' (text input), 'Billeder' (section with a 'Billednavne:' label and a 'Browse' button), and 'Primært billede' (text input with a 'Browse' button). A red 'Opret område' button is located at the bottom left of the form.

For at kunne oprette noget på hjemmesiden skal brugeren være logget ind. Hvis brugeren ikke er logget ind vil den "modal" eller pop-up, der åbnes indeholde en besked om, at der skal logges ind og en knap, der videresender brugeren til loginsiden.

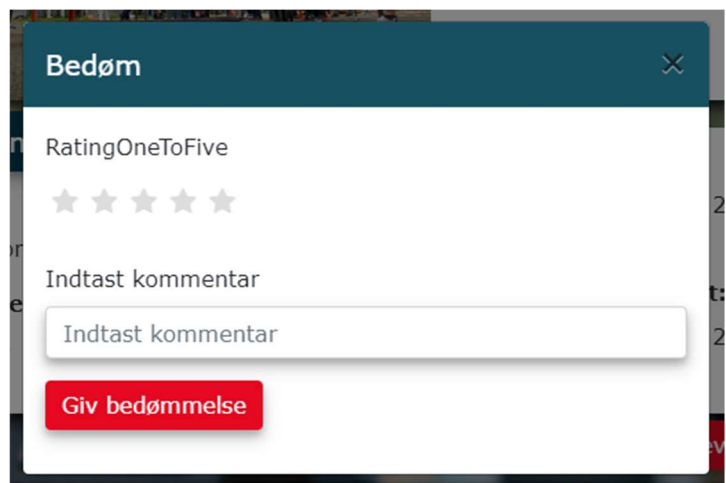
Oprettelsen af begivenheder og udendørsområder foregår inde i en "modal", som er en form for pop-up, der åbnes, når brugeren trykker på "Opret" knappen. Derefter bliver brugeren præsenteret for nogle felter, der skal udfyldes og billeder, der skal uploades, hvorefter dette bruges til at lave de "Cards", der bliver vist til brugeren på "Index" siden.

I navigationsbjælken eller "headeren" findes der links til de forskellige dele af applikationen. "Områder" indeholder den del af applikationen, der har med legepladser, hundeparker og udendørstræningsområder at gøre, mens "Begivenheder" indeholder de begivenheder, der oprettes af brugere. "Mine områder", "Mine begivenheder" og "Mine bedømmelser" indeholder de ting, brugeren selv har oprettet. På mobilversionen vil navigationsbjælken blive vist som en dropdown med de samme links.



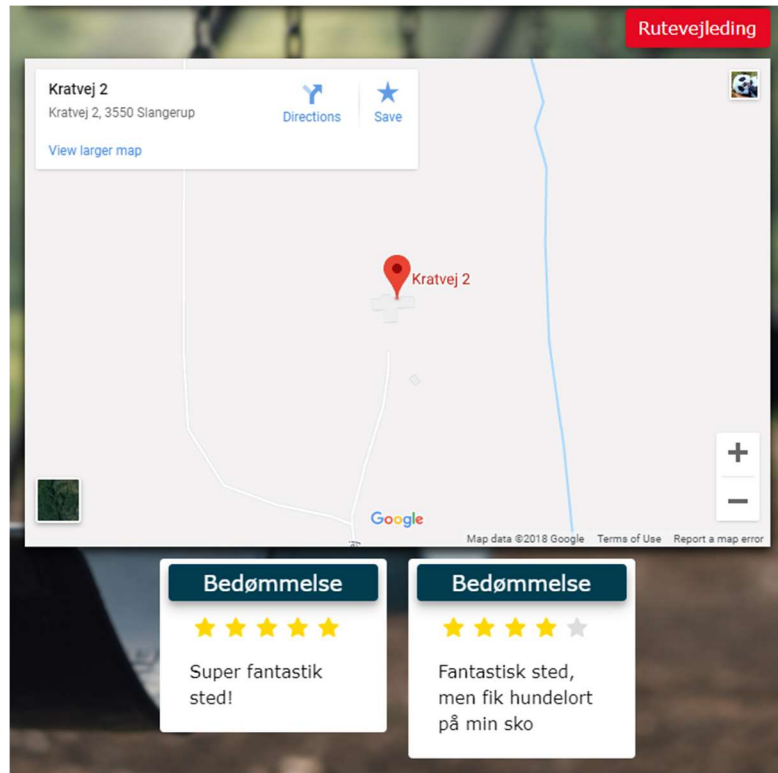
Når brugeren ser på Details siden, får brugeren en masse information omkring denne specifikke begivenhed eller område. Ydermere kan brugeren, ved enten at trykke på stjernerne eller klikke på "Bedøm" knappen, bedømme området.

Her har brugeren mulighed for at vælge imellem 1-5 stjerner og skrive en kommentar til det område, de har valgt at bedømme.



Ydermere på detaljesiden bliver brugeren præsenteret for et kort, der viser områdets- eller begivenhedens lokation, derudover har brugeren mulighed for at trykke på knappen "Rutevejledning" og få direktioner til lokationen fra sin nuværende position.

Derudover bliver der også vist bedømmelser og kommentarer fra brugere.



5.3 Kommunikation imellem frontend og backend.

Fordi *Ud i det blå* er en ASP.NET Core MVC-webapplikation sker der webkommunikation imellem frontend og backend. Denne kommunikation sker dog anderledes i forhold til Web API-kald, som standard sender og modtager JSON- eller XML-dataformater. I ASP.NET Core MVC sendes data fra controlleren sammen med viewet, så det ikke er separeret data, som ses ved kald af web API for at udfylde et view. Datakommunikationen fra view til backend bliver kommunikeret igennem et submit request, hvor dataene fra et form HTML-element, som er knyttet til den model, der tilhører viewet, videreføres over til en controller action. Det er muligt at sende form data i dataformatet Multipart/form-data, der gør det muligt at sende store mængder data i form af tekst eller binæredata. Derudover er det også muligt at sende kompliceret datatyper som filer, som andre formater som JSON og XML ikke kan udføre. Form-data formatet sender view data, som et key/value par i kommunikationen til backend. Formatet er svært at læse for et menneske, i forhold til JSON og XML, hvor strukturen gør det mere synligt for hvilke modeller eller værdier der sendes^[27]. I og med Form-data har mulighed for at sende meget større mængder data, og det ikke er kompliceret at sende filer med i HTTP-requesten, passer det godt til *Ud i det blå*'s behov.

^[27] <https://goo.gl/hDe7cr> - HTML Forms W3School documentation

Dette eksempel viser et udkast af skrukturen, som form-data sendes i, når en bruger filtrerer på udendørsområder i *Ud i det blå*.

```
-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.AreaTypes.TrainingArea"

true
-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.Name"

-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.CityChecked"

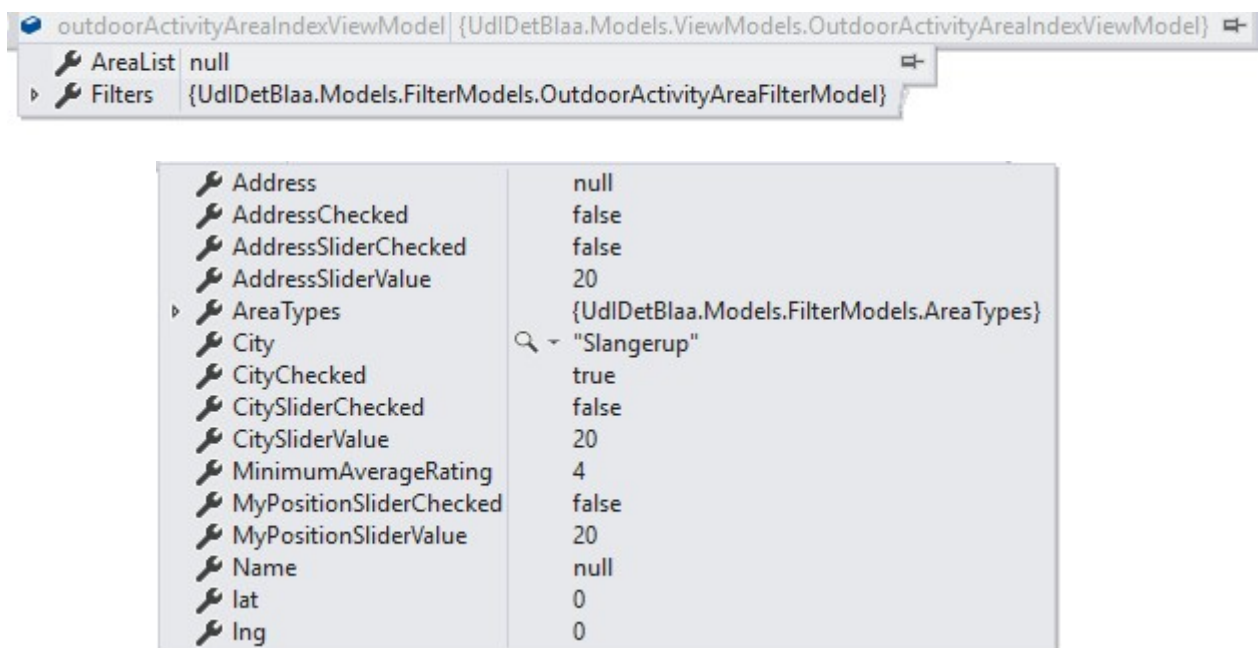
true
-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.City"

Slangerup
-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.CitySliderValue"

20
-----WebKitFormBoundaryEJKVFA7Sqx0TRNex
Content-Disposition: form-data; name="Filters.Address"
```

Dette form-data bliver brugt sammen med asp-for tag for at binde form-data sammen med MVC-viewets datamodel og sende det.

Dette eksempel viser, hvordan kontrolleren ser og viser det sendte form-data fra en udendørsområdefiltrering. Her har asp-for tags i MVC-viewet bundet filter form-dataen sammen med modelparameteren i controller metoden.



The screenshot shows the Outgoing Messages window in Visual Studio. The message is from 'outdoorActivityAreaIndexViewModel' to '{UdiDetBlaa.Models.ViewModels.OutdoorActivityAreaIndexViewModel}'. The message body is a JSON object representing the 'Filters' model.

```
{
  "AreaList": null,
  "Filters": {
    "UdiDetBlaa.Models.FilterModels.OutdoorActivityAreaFilterModel": {
      "Address": null,
      "AddressChecked": false,
      "AddressSliderChecked": false,
      "AddressSliderValue": 20,
      "AreaTypes": {
        "UdiDetBlaa.Models.FilterModels.AreaTypes": {
          "City": "Slangerup",
          "CityChecked": true,
          "CitySliderChecked": false,
          "CitySliderValue": 20,
          "MinimumAverageRating": 4,
          "MyPositionSliderChecked": false,
          "MyPositionSliderValue": 20,
          "Name": null,
          "lat": 0,
          "lng": 0
        }
      }
    }
  }
}
```

5.4 UX

Et af de mange mål hos EUX IVS er at holde applikationen let og overskuelig for brugeren.

Dette er opnået ved at sørge for, at alle elementer passer ind i en designplan, der har til formål at gøre applikationens udseende luftigt, mens brugeren tydeligt kan se for eksempel knapper, tekstfelter og lignende. For at gøre dette, har teamet researchet og analyseret andre hjemmesider, for at hente inspiration og idéer til *Ud i det blå's* layout²⁸.

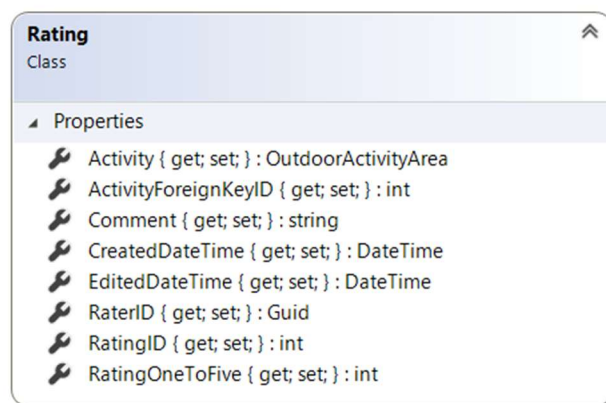
Mens organiseringen af elementer har været planlagt fra starten af udviklingsforløbet, er farvetemaet og design guiden udarbejdet og tilpasset flere gange, så teamet var sikre på at give den bedste brugeroplevelse. Derefter har applikationen været præsenteret til enkelte udvalgte brugere for at få mundtlig feedback omkring udseende og funktionalitet, hvorefter eventuelle rettelser er lavet for at tilfredsstille brugerne. Dette kan ses i bilag 5 Brugerfeedback.

5.5 Modeloverblik

Både i sammenhæng med MVC, men også i sammenhæng med Entity Framework Core, bliver der skabt nogle modeller med forskellige formål. De modeller der danner kernen for systemet er dem, som både MVC og Entity Framework benytter til at give systemet nogle lagrede data at benytte sig af. Det er altså også de modeller, der sammen med tabellerne der er et overblik over i [5.6 Databaseoverblik](#), giver de såkaldte entities.

Det fulde overblik kan ses i bilag 9 Modelklassediagram.

Herunder er et eksempel på en enkelt modelklasse.



Billede 3 Eksempel på modelklasse (Rating modelklassen)

^[28] <https://goo.gl/nDE1oJ> - UX artikel

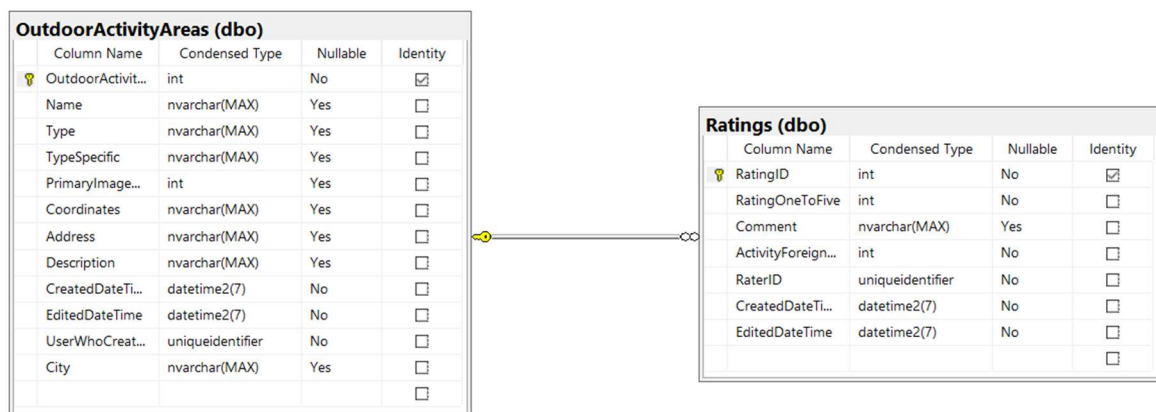
5.6 Databaseoverblik

For at skabe et overblik over databasen, er der udarbejdet et ER-diagram. Dette ER-diagram viser, hvilke tabeller databasen indeholder, og hvilke relationer de forskellige tabeller har til hinanden. Derudover viser det, hvilke datatyper de forskellige kolonner i tabellen er, om de er nullable og om de er identity. Hvis en kolonne er nullable, betyder det, at kolonnen kan være tom for data. Hvis en kolonne er identity, betyder det som udgangspunkt, at den kan bruges til at identificere den specifikke kolonne frem for en anden. Det har ofte den betydning, at den har det, der kaldes et increment og et increment seed. Det betyder, at den for eksempel starter ved et increment seed på 1 og har et increment på 1, så starter den først indsatte kolonne med automatisk i feltet at få sat 1 ind, den næste får så automatisk 2 og så videre. Identity kan sættes på andre kolonner end for eksempel en primarykey kolonne, det er dog typisk for en primarykey kolonne at have identity på sig. Ligesom beskrevet i [5.5 Modeloverblik](#), er det disse tabeller, der sammen med modelklasserne giver de entities, der arbejdes med i systemet.

De relationer, der er mellem tabellerne i *Ud i det blå*, er udelukkende En-Til-Mange relationer, der dog i tilfælde ligger dobbelt, altså at relationen også går den anden vej. En En-Til-Mange relation mellem to enheder er som følgende eksempel. I de hypotetiske tabeller A og B, hvori et element af A kan være knyttet til mange elementer af B, mens et element af B er knyttet til kun et element af A.

Det fulde ER-diagram kan ses i bilag 8 ER-diagram.

Herunder er et eksempel på to tabeller i ER-diagrammet, hvor der imellem dem er en enkelt En-Til-Mange relation. Relationen beskriver, at en OutdoorActivityArea er knyttet til mange Ratings, og en Rating er kun tilknyttet én OutdoorActivityArea.



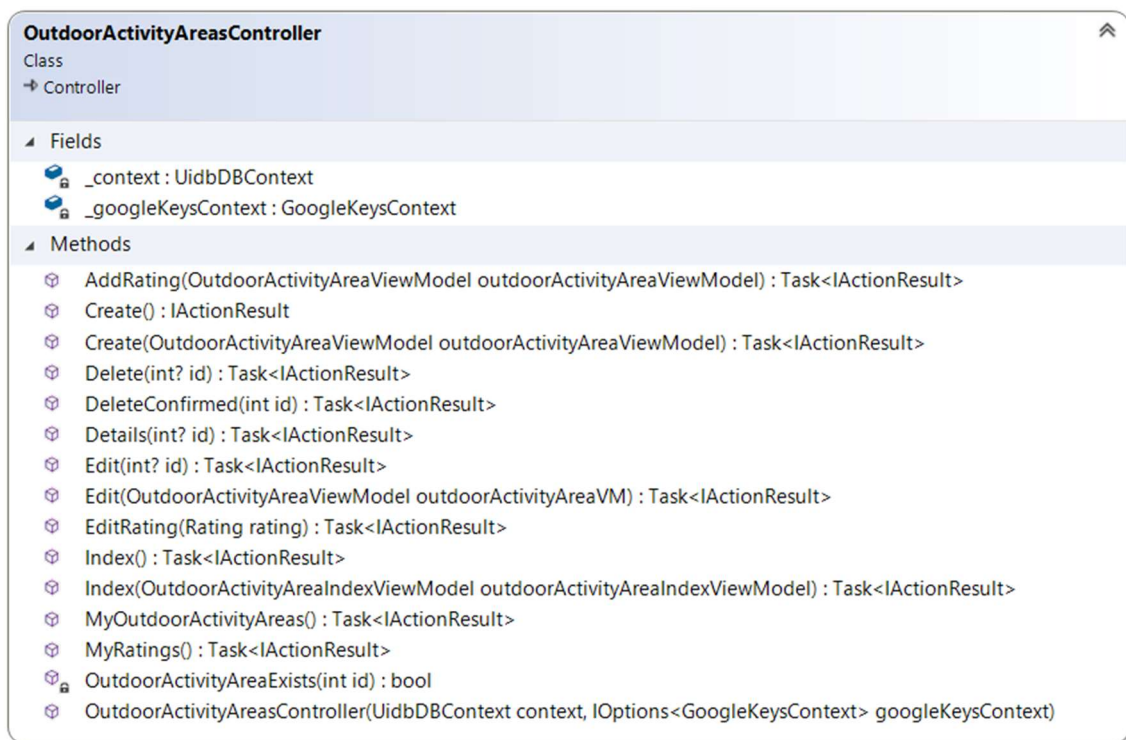
Billede 4 Eksempel på ER-diagram med En-Til-Mange relation

5.7 Klasseoverblik

For at give et overskueligt overblik over alle klasser i programmet, deres properties/fields og de metoder de indeholder, er der udarbejdet et fuldt klassediagram. De er sorteret efter klassetype i bilaget, således at det er tydeligt, hvilke der har mest med hinanden at gøre. Det er en praktisk og minimalistisk måde at skabe et overblik over systemet og dets funktionaliteter uden rent faktisk at læse koden.

Det fulde klassediagram kan ses i bilag 10 Fuldt klassediagram.

Herunder er et eksempel på en enkelt klasse i det fulde klassediagram for at give et indtryk af, hvad det indeholder af information.



Billede 5 Eksempel på klasse i det fulde klassediagram (*OutdoorActivityAreaController*)

6 Udviklingsforløbet

Udviklingen af applikationen er forløbet ved, at teamet til at starte med lavede et simplificeret flowchart, der havde til formål at opridse de funktioner, der skulle med i programmet. Derefter blev modellerne udviklet, så frontend teamet havde en idé om, hvordan denne data skulle fremvises, og backend teamet havde en idé om, hvordan databasen skulle se ud.

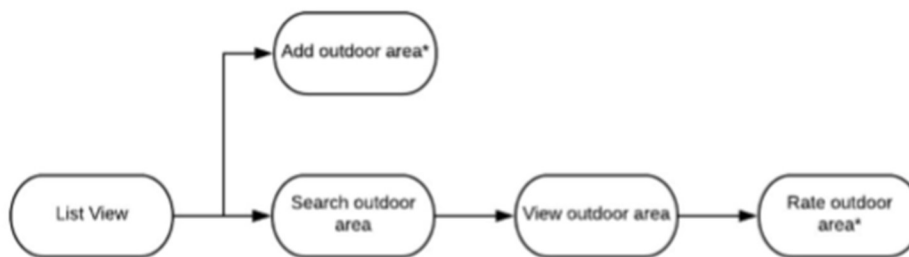
Udviklingen har fulgt principperne i Extreme Programming med et frontend team og et backend team, der arbejder agilt, der begge har brugt et fælles kanban board til at holde styr på arbejdsopgaver. For at dette kunne lykkedes, har teamet brugt Azure DevOps' TFS til at oprette flere branches af projektet, hvor der kunne arbejdes selvstændigt af alle

medlemmer i teamet. Arbejdsopgaverne er blevet fordelt i 2 kategorier - backend og frontend med hver deres team.

Frontend teamet har bestået af Magnus og Mathias, mens backend teamet har bestået af Niclas og Tobias. Denne arbejdsfordeling har betydet, at de to teams har kunnet arbejde uafhængigt af hinanden, og derved være mere effektive. Der er blevet holdt daglige Stand-up møder, der havde til formål at hjælpe med forskellige problemstillinger, og for at begge teams havde et generelt overblik over projektets fremskridt. Derudover er der blevet holdt retrospektive møder hver tredje dag, hvor hele teamet har taget stilling til, om produktet kunne nå i mål inden for deadline, og sammenlignet backloggen med resolved tasks for at se om der var nogle funktionaliteter, der ikke kunne udvikles indenfor tidsrammen.

Det simple systemkort set nedenfor, er det, der blev udarbejdet i starten af planlægningsfasen. Det er efterfølgende blevet udvidet til at være mere detaljeret og indeholde alle funktionaliteterne. Det udvidede systemkort kan ses i bilag 7 Systemkort.

* Registered user through B2C



6.1 Use cases

Udover systemkort og flowcharts, blev der også udarbejdet use cases i planlægningsfasen for at finde ud af, hvilke funktionaliteter applikationen havde behov for. De use cases er blevet brugt som en vejledning under udviklingsfasen til at holde en rød tråd i hvilke funktioner, der skulle udvikles.

Use casen der ses nedenfor, er blot et eksempel på mange af de use cases teamet har udarbejdet til denne applikation. Resten af use cases kan findes i bilag 3 Use cases

Brugscenarie: Login	Ident: UC-2
Aktør: Bruger	
Beskrivelse: Aktør logger ind for ekstra funktionaliteter.	
Forudsætning: Aktøren er oprettet.	
Forventet resultat: Aktøren logges ind.	
Forløb: 1. Aktøren kender sin e-mail og sit kodeord. 2. Aktøren logges ind.	
Alternativt forløb: 1. Aktøren indtaster ugyldig e-mail eller kodeord. 2. Aktøren har glemt sit kodeord, se UC-3.	

6.2 Projektstyring

For at holde styr på projektet har teamet udnyttet flere forskellige metoder og værktøjer.

Herunder findes blandt andet Azure DevOps, som er blevet brugt til kanban board og TFS/Versionskontrol. Udover dette er der arbejdet efter principperne i Extreme Programming på trods af, at de ikke er fulgt til punkt og prikke^[29].

Azure DevOps har gjort det muligt at arbejde i to teams, parallelt med hinanden. Dette betyder, at frontend og backend er blevet udviklet samtidigt i stedet for små brudstykker ad gangen. Azure DevOps TFS, som er integreret med Visual Studio, har tilladt teamet at oprette fem branches, en til hver udvikler og en main branch, der kunne merges ind i for at undgå at overskrive hinandens kode.

Derudover har kanban boardet gjort, at applikationens udvikling er blevet delt op, så det har været lettere at overskue de forskellige arbejdsopgaver, og hvilket team der bør gå i gang med den specifikke opgave. Derudover overskueliggøre det arbejdsopgaverne og hvor langt der er til mål. Kanban boardet opdateres løbende med opgaver, hvilket også betyder at udviklingen i princippet aldrig behøver at få en ende.

Som nævnt før er der arbejdet efter Extreme Programmings principper. Der har været stor fokus på kommunikation og at holde overblikket over projektet, og alle i teamet har været med i hele udviklingsfasen, på trods af at der har været en opdeling i forhold til udvikling af frontend og backend. Dette er lykkedes ved at hele teamet har holdt daglige stand-up meetings, og retrospektive møder en gang om ugen sammen, og derved har frontend udviklerne også haft et indblik i hvilke funktioner, der er blevet udviklet i backend og vice versa. Derudover har teamet forsøgt at holde koden så let og overskuelig som muligt, samtidigt med at følge YAGNI(You Ain't Gonna Need it) og DRY(Don't repeat yourself) principperne.

Dette betyder, at redundant kode så vidt muligt er blevet undgået, og at der ikke er blevet spildt tid på funktioner, der ikke var behov for.

Fordelen ved møderne teamet har holdt er, at det gør det muligt for andre medlemmer at komme med feedback og derved finde den bedste vej frem, for at øge kvaliteten af produktet uden at bruge lang tid på det. Alle medlemmer i teamet har haft et fælles mål under hele udviklingsfasen, og har bidraget bedst muligt til at nå det fælles mål. Derudover er der ofte blevet benyttet "Par-programmering", specielt i backend udviklingen, hvilket betyder at to personer sidder sammen om én skærm og ét tastatur og skriver kode, derved kan udviklerne optimere koden med det samme, og bruge langt mindre tid på fejlsøgning.

^[29] <https://goo.gl/ympJhT> - Extreme Programming - Altexsoft

6.3 Afprøvning

Afprøvningsfasen er stort set ligeså vigtig som udviklingen af produktet. Det er vigtigt at få lavet en korrekt og ordentlig afprøvning ved hjælp af nogle bestemte afprøvningsmetoder. Der er brugerbaseret test, og så er der andre strukturerede afprøvningsmetoder, der gør, at alle de forskellige elementer på hjemmesiden bliver testet igennem. Brugertest er en af de bedste afprøvningsmetoder, der kan foretages, når det kommer til at teste ens produkt. Måden en bruger navigerer rundt på, og hvilken rækkefølge brugeren vælger at tilgå de forskellige ting er ofte anderledes end udvikleren, der har siddet og arbejdet med produktet igennem mange uger/måneder. Vi har fået nogle af vores forældre samt søskende til at lege lidt rundt på hjemmesiden for at høre, hvad deres mening er mht. de forskellige funktionaliteter og samtidig layoutdelen. Generelt var deres mening, at produktet så godt ud og var gennemført igennem hele hjemmesiden, dog med nogle små finesser/detaljer, der efterfølgende er blevet implementeret.

Udviklerne af produktet har tjekket alle generelle links på hjemmesiden, om de fører brugeren til det korrekte sted, og om opdateringshastigheden er acceptabel og ikke resulterer i for meget ventetid. Efterfølgende tjekkes HTML / CSS, så tjekkes der om koden er optimeret på bedst mulig måde, og så tjekkes der om hjemmesiden er responsiv i forhold til forskellige skærmopløsninger og enheder. Hjemmesiden er udviklet, så den kan bruges på alle slags mobilenheder, tablets og computerskærme, så det er uafhængig af, hvor brugeren befinder sig, og hvilken gadget brugeren har til rådighed.

En anden vigtig ting er også alt det visuelle som at tjekke, at de forskellige pages er strømlinet med samme font, fontstørrelse, afstand imellem de forskellige elementer og om farvekoderne er gennemtænkt. Derudover skal forbindelserne tjekkes imellem de forskellige udviklingsværktøjer, om de kører efter hensigten, og om forbindelsen er stabil og laver de rigtige kald, når der udføres en action i controllerne.

7 Afgrænsninger

I dette afsnit beskrives der hvilke funktioner, som er blevet tilsidesat i udviklingen af webapplikationen, og hvorfor de ikke er blevet implementeret.

7.1 Skudt under mål "_(ツ)_/"

I henhold til afgrænsninger har teamet formået at opfylde samtlige krav specificeret i kravspecifikationen. De i forvejen fastsatte begrænsninger fra kravspecifikationen er dog ikke udviklet, men er beskrevet i [8 Systemets fremtid](#). Der er dog nogle løst definerede krav i kravspecifikationen, som muligvis kunne have været levet op til i en lidt højere grad. Teamet finder det dog selv tilfredsstillende. Derudover har teamet i modsætning til afgrænsning udvidet systemet med flere funktionaliteter såsom sider til overblik over brugerens egne oprettede aktiviteter, bedømmelser og begivenheder samt tilføjet kommentar til bedømmelser og mulighed for at redigere brugerens egne områder og begivenheder.

Til gengæld har teamet siden opdaget, at der af kravspecifikationen ikke fremgår hverken af use cases eller specificeringer af overblikkene over områder eller begivenheder og ej heller for detaljesiderne for specifikke områder eller begivenheder, hvilket ellers alt sammen var tiltænkt.

8 Systemets fremtid

De funktioner, som beskrives i dette afsnit, er funktionaliteter som teamet ønsker implementeret i produktet i fremtiden.

8.1 Push notifikationer

Ved oprettelse af nye begivenheder ville det være optimalt for brugeroplevelsen, at personer kan melde sig til at få push notifikationer, således brugere ikke selv skal tjekke, når nye begivenheder bliver slået op i *Ud i det blå*. Opsætningen for push notifikationer ville ikke kræve meget at opsætte, men der skal tages stilling til brugen, og hvordan det skal sættes op sammen med den resterende webapplikation. Dette er en af de funktionaliteter, der allerede i kravspecifikationen var blevet fravalgt som en afgræsning grundet, at teamet ikke var sikre på at kunne nå en fyldestgørende implementering inden deadline.

8.2 GPS-Tracking

GPS-tracking er en funktionalitet, der ville være brugbar i mange sammenhæng med applikationen. *Ud i det blå* benytter allerede GPS data til diverse funktionaliteter, men gør ikke decideret brug af GPS-tracking.

GPS-tracking ville give mulighed for blandt andet at have et interaktivt kort med nærtliggende områder, hvis detaljesider kunne åbnes ved at klikke på dem, eller eventuelt automatisk sortering af områder på forsiden på baggrund af aktuel lokation. Selve ideen om at benytte GPS-tracking er en af de funktionaliteter, der allerede i kravspecifikationen var blevet fravalgt som en afgræsning grundet, at teamet ikke var sikre på at kunne nå en fyldestgørende implementering inden deadline.

8.3 Ekstern API data til områder og begivenheder

Udnyttelse af eksterne API data, omkring henholdsvis udendørsområder og begivenheder, ville muligvis kunne øge aktiviteten og brugbarheden af *Ud i det blå*. Denne hypotese er baseret på, at sammenspillet imellem brugerdata og eksterne data ville give en mere fyldestgørende base for brugere at benytte. Denne funktionalitet kan også give en udfordring vedrørende duplikater, og hvordan disse skal håndteres. Disse data kunne for eksempel komme fra Google, Børn i byen eller andre virksomheder, der udstiller data.

8.4 Unit testing

Ved fremtidig udvikling af *Ud i det blå* ville det give god mening at implementere unit testing til at teste backend og frontend kode. Unit testing er en smart måde helt automatisk at få testet den kode, der er blevet udviklet. Det gør det også muligt for en udvikler at lave testdriven development, altså at lave testen først og så udvikle koden efterfølgende. Det havde have været optimalt at implementere dette ved starten af produktets udvikling, så

teamet følger test-driven development princippet. Men dette kræver dog en vis erfaring med, hvordan man følger dette princip og benytter unit testing således, at det ikke gør udviklingen langsommere end den ellers ville have været foruden. Denne erfaring besidder teamet ikke som en helhed, hvilket har resulteret i, at det er blevet fravalgt at benytte Unit testing under netop dette forløb.

8.5 CI og CD i Azure DevOps

CI (continuous integration) og CD (continuous delivery) i Azure DevOps ville, både af hensyn til principperne i XP (eXtreme Programming), men også for udviklernes skyld være en rigtig fed udvidelse af *Ud i det blå*. Det benyttes til at skabe et automatisk flow, der er i stand til automatisk at teste, publicere og kanalisere succesfulde indtjekninger af kildekoden til systemet. Et eksempel ville være, at når koden bliver checket ind i main, vil CI/CD i Azure DevOps først forsøge at bygge koden. Hvis det bliver bygget succesfuldt, vil det efterfølgende udføre specificerede unit tests. Hvis de fejler, vil en mail blive sendt ud til projektlederen. Hvis det derimod lykkes, vil koden blive publiceret til et testmiljø i Azure. Hvis projektlederen godkender dette ved hjælp af et enkelt klik i DevOps i kontrolpanelet, vil koden blive publiceret til produktionsmiljøet, hvorefter en ny version er fuldt publiceret. Alt dette imens programmørerne fortsat arbejder på at udvikle systemet, så tiden på alle disse ting ikke går til spilde på manuelt arbejde i mellemtiden. Alt dette er grundlaget for, at det klart er en fremtidig vision for *Ud i det blå* at have CI/CD implementeret i udviklingen.

8.6 Flere loginudbydere i Azure B2C.

For at give et større incitament for brugere til at oprette et login i *Ud i det blå*, ville det være passende at implementere flere loginudbydere, så det ikke kun er Facebook og Azure AD B2C, der er tilgængelige. Får at opnå dette, ønskes det at loginudbydere udvides med muligheder som Google konto, Microsoft konto, LinkedIn eller Twitter kunne bruges, så forbrugere ikke bliver afskåret.

8.7 Forbedre validering

Der er blevet implementeret validering rundt omkring på hjemmesiden, hvor brugeren får nogle generelle svar, der samtidig resulterer i, at applikationen ikke crasher. Nogle forbedringsforslag mht. bedre validering kunne være bedre beskrivelse af netop, hvorfor den har fejlet, og hvad brugeren mangler for at kunne fuldføre den ønskede handling. En anden ting kunne være implementering af større grad af validering af enkelte felter, så brugeren ved, hvilke felter der er krævet udfyldt eller er blevet udfyldt ukorrekt.

8.8 Detaljer

Der kunne sagtens udføres mange små finesser, der ville sætte det sidste præg på produktet, men grundet tidspresset er der blevet prioriteret. Der kunne udføres små detaljer, så som når der vælges *Login*, så redirecter den til siden, brugeren netop befandt sig på, fremfor at den returnerer brugeren til startsiden på *Ud i det blå*. En anden ting, der kunne forbedres, var stylingen af *Register User*, så den var mere strømlinet med det resterende designlayout. Dernæst kunne filtreringsmuligheder udvides, så det var muligt at filtrere efter senest tilføjet, efter egne oprettede opslag med mere.

8.9 Udvide sortiment

På længere sigt kunne der sagtens både udvides med flere udendørsområdetyper, men samtidig også et helt andet sortiment så som betalte oplevelser. Tilføjelser af udendørsområdetyper kunne for eksempel være afslapning, idylliske lokationer, historiske monumenter, sportsgrene med mere. Folk har mange forskellige interesser, og derfor ville det give god mening at udvide dette på et tidspunkt for at fange en bredere målgruppe. Derudover ville et nyt sortiment som betalte oplevelser også sagtens kunne implementeres, det kunne være lige fra minigolf til kitesurfing, guide i et bestemt idyllisk område eller lignende. Det er i realiteten kun fantasien, der sætter grænser, når det kommer til udvidelser af dette produkt.

8.10 Udvidelse af bedømmelser i form af statistisk og filtrering/sortering

Det giver en fed oplevelse for brugere at have en klar visualisering af, hvor mange bedømmelser, på hvor mange stjerner, der giver den samlede gennemsnitsbedømmelse. Derudover er det rart at kunne filtrere eller sortere bedømmelser på diverse parametre såsom seneste, ældst, antal stjerner, længste beskrivelse med flere. Derfor er dette en del af fremtiden for *Ud i det blå*.

8.11 Udvide flere relevante felter på begivenheder

Der mangler helt tydeligt nogle relevante felter på begivenhedsdelen. Nogle felter, der ville give god mening at have tilføjet ville for eksempel være dato, starttidspunkt og sluttidspunkt så brugerne er klar over, hvornår begivenheden finder sted, og hvornår den starter og slutter.

8.12 Tilknyt et specifikt udendørsområde til en begivenhed

Det kunne være en rigtig praktisk løsning for brugere, når de skal oprette en begivenhed, at de kan knytte den til et i forvejen oprettet område, hvis begivenheden også finder sted der. Det ville gøre, at man som opretter af en begivenhed, var fri for at skulle indtaste disse informationer og indsætte en masse billeder, da de informationer allerede er tilgængelige i forbindelse med de, der allerede er tilknyttet det tilknyttede område.

8.13 Paging på udendørsområder samt begivenheder

En anden ting, der helt sikkert skal implementeres, hvis hjemmesiden blev populær var paging. Paging er en nødvendighed i *Ud i det blå*. I teorien skulle der opnås en områdebase på for eksempel 300 udendørsområder. Det ville for det første skabe dårlig performance uden paging, idet den skal loade alle elementerne inden siden bliver vist, og samtidig ville det give bedst mening i sidehåndtering fremfor at scrolle langt ned på en side og have langt til navigationsbaren samt filtreringsdelen.

8.14 Fejlhåndtering og logging

Der er allerede implementeret noget fejlhåndtering, men der kan sagtens fejlhåndteres bedre end, hvad der allerede er gjort. Der er en del funktioner, der ikke er beskyttede mod uventede fejl, som hvis de opstår, resulterer i en fejlside, som ikke siger noget relevant til hverken brugeren, eller *Ud i det blå's* team omkring fejlen, der opstod. Derfor bør der i produktets fremtid implementeres en større og bedre fejlhåndtering, hvortil der bør knyttes logging og mulighed for at indstille forskellige log-levels. Dette gør det nemmere at rette fremtidige fejl i nye releases, og brugerne vil få en mere glat og fejlfri oplevelse med systemet.

8.15 Udvide begivenhedssystem med tilmelding/betaling

En anden god implementering havde været tilmelding af begivenheder og samtidig udføre betaling igennem hjemmesiden, så brugeren kunne gå direkte til den ønskede begivenhed med en kvittering for betalingen.

8.16 Administrator funktionalitet

Administrator funktionaliteter til blandt andet verificering af nyoprettede områder og begivenheder inden de gøres tilgængelige og synlige for systemets brugere. Funktionaliteter til at slette og rette hvad der passer dem direkte i systemet uden at skulle til at redigere direkte i databasen. Ban af brugere. Her kan der helt sikkert også komme flere til, men en administratorrolle, nogle administrationsmuligheder og nogle administrator funktioner er helt sikkert en del af fremtiden for *Ud i det blå*.

8.17 Håndtering af trolls og bandeord

Med trolls menes der mennesker, som udnytter, at alle som udgangspunkt kan oprette en bruger og oprette områder, begivenheder og bedømme områder. Med udnyttes menes der, at nogle måske kunne finde på at lægge pornografiske billeder op, bruge bandeord, give dårlige eller rigtig gode anmeldelser uden grundlag herfor. Disse kan langt hen ad vejen stoppes ved hjælp af programmerede filtre, der undersøger opslag på siden for uønsket materiale. Den resterende del af vejen kan og bør stoppes igennem administrator funktioner såsom IP- og bruger-bandlysninger og verificering af opslag, før de gøres tilgængelige på siden.

8.18 Brugerverificering og levels

Mulighed for eksempel at verificere sin bruger med NemID for ikke at skulle vente på at de opslag, der laves af brugeren behøves at verificeres af for eksempel en administrator. Der kunne implementeres bruger levels, der siger noget om, hvor troværdig og/eller aktiv en bruger er i systemet, hvilket på sigt kunne have indflydelse på diverse opslags verifikationsfaktorer og give incitament til at oprette nye områder eller begivenheder i systemet.

9 Diskussion

Ud i det blå har arbejdet meget med at gøre sidens design responsivt, hvilket giver en god dynamik på siden, når den skaleres. Systemet er gjort responsivt ved hjælp af Bootstrap, men det er ikke det eneste CSS-library, der hjælper med at lave responsive design. Et andet godt eksempel på dette er Bulma, hvilket også er et let og responsivt library at arbejde med. Fordelen ved at benytte for eksempel Bootstrap er, at det er velkendt, og giver i højere grad brugere en oplevelse af genkendelighed. En ulempe ved Bootstrap er at det kan virke meget upersonligt for et site, da det til en vis grad kommer til at ligne andre velkendte hjemmesider.

Det kan, og synes *Ud i det blå* selv at de i høj grad har, dog gøres personligt ved selv at tilføje og ændre den styling, der følger med Bootstrap som standard, således at Bootstraps primære funktion er at gøre elementer responsive, men at resten af designet er lavet i egen-definerede stylesheets. En helt anden løsning der måske havde været værd at overveje, hvis tidspresset ikke var så højt, er at lave hele designet selv fra bunden. Dette er en lang proces, der kan give super fede customiserede resultater, men det har simpelthen været for stor en opgave, hvis det skulle have givet et flottere resultat indenfor den tidsramme, der var givet. *Ud i det blå* har dog lavet en stor mængde egen-defineret CSS, som gør siden personlig, responsiv og til dels velkendt. Teamet bag føler selv, at det giver en rigtig god balance mellem de tre parametre.

Derudover er designet på siderne, og opdelingen af for eksempel områder og begivenheder strømlinet, men stadig gjort forskellige ved hjælp af forskellige farvetemaer hele designet igennem. Det giver en god dynamik og overskueliggør for brugeren, hvilken del af webapplikationen de befinder sig i.

Webapplikationen er brugerdata baseret og frit tilgængelig. Det giver nogle fordele og nogle ulemper. På den ene side gør det, at siden har rig mulighed for at blive beriget med rigtig mange data. På den anden side kan det have den effekt, at den kan virke flad og mangelfuld for en bruger, hvis denne brugers ønske blot er at finde et område eller en begivenhed i nærheden af sig selv. Det kræver, at det bliver lanceret på den helt rigtige måde, og at det bliver udbredt og populært, så det kan ende med at være brugernes egen databank for udendørsområder. Hele pointen er jo netop at skabe et større incitament til at komme "*Ud i det blå*".

10 Konklusion

På baggrund af de udarbejdede dele af rapporten, udviklingen af produktet, udarbejdelsen af projektet i det hele taget og med udgangspunkt i problemformuleringen, kan vi drage nogle konklusioner.

At arbejde i et gruppeprojekt på fire mennesker kan såvel være en byrde, som det kan være en velsignelse. Det er på godt og ondt, at vi kan konkludere, at teamet bag *Ud i det blå* har fungeret rigtig godt, til trods for uenigheder og skæve arbejdstider. De fire parter har arbejdet både individuelt, i grupper af både to og tre på diverse tidspunkter, men også som en hel gruppe af fire. Det meste af tiden har teamet dog arbejdet i grupper af to, frontend og backend, hvilket har fungeret fantastisk. Det har det, fordi begge dele af gruppen har levet op til hinandens forventninger og deadlines.

Valget af at lave en responsiv ASP.NET MVC Webapplikation har været et fantastisk valg, da de forskellige dele af et sådan projekt harmonerede rigtig godt med de forskellige erfaringer og evner, der besiddes af hver enkelt medlem af teamet. Det bar også frugt af, at det var en smart måde at bearbejde det problem produktet skulle løse, på den måde skabte det motivation, da produktet rent faktisk følte, som noget vi reelt ville kunne gå videre med og sætte i verden som en løsning på netop det af problemformuleringen, beskrevne problem. Det har også medført, at teamet har arbejdet effektivt med produktet, i en sådan grad at der som udgangspunkt ikke er nogle afgrænsninger udover de af kravspecifikationen, i forvejen fastsatte begrænsninger. Det betyder altså, at *Ud i det blå* har levet op til specifikationerne i kravspecifikationen og tilmed er blevet udvidet med endnu mere funktionalitet. Det gav en super fed motivation og følelse omkring projektet for hele teamet. Det har, af netop disse grunde, også givet anledning til en yderst omfattende liste over funktionaliteter, vi ønsker at implementere i *Ud i det blå* i fremtiden.

Valget af Entity Framework Core som ORM, SQL Server som database server, Azure DevOps som projektstyrings- og versionskontrollsværktøj og Azure Cloud Services som hosting til hele systemet har alle harmoneret fantastisk med projektets type og formål. Det har givet et rigtig fedt arbejdsflow med hurtige skridt fra debugging test til test i et produktionsmiljø, hurtig udvikling samt overblik over nuværende og fremtidige arbejdsopgaver.

Det at vi valgte at benytte os af en ekstern identitetsserver til validering af brugere og brugerdata, og at vi valgte netop Azure B2C, har også været en super fed oplevelse. Det var et forsøg på at være pionerer, da netop Azure B2C er en relativt ny spiller på markedet. Det viste sig at være en god ide, specielt nu hvor alt andet også fungerer gennem Azure. Det var nemt at håndtere, styre og implementere i løsningen. Det viste sig faktisk at være så stor en fordel, at det forkortede hele udviklingsprocessen, da der er en stor del af sikkerhedsansvaret og udviklingen, der fralægges selve projektet og pålægges servicen i stedet for. Selvom det kan virke, som om det er at springe over, hvor gærdet er lavest, er det faktisk det præcis modsatte. Det er sikrere, hurtigere og smartere. Det tillader nem interaktion mellem de data, der er i Azure B2C, og de data man også ønsker at have lagret i egen database uden at gå på kompromis med både sikkerhed og udviklingstid.

En ting vi i løbet af projektet har haft for lidt fokus på, har været at finde mere konkrete svar på de spørgsmål vi i problemstillingen satte op. Dermed ikke sagt at vi ikke har fundet svar og kan konkludere på dem, men det har været et ubevist nedprioriteret fokus grundet vores store engagement med at få et produkt og en rapport i hus, vi kunne være både glade for og stolte af. Det har bevirket, at vi er endt med et godt slutprodukt, der i sig selv langt hen ad vejen besvarer disse spørgsmål. Vores fokus har nemlig ligget i at leve op til spørgsmålene i højere grad end at besvare dem.

Projektet har som helhed været super fantastisk, vi har levet op til at løse problemet og på en måde, hvor det også lever op til den kravspecifikation, vi satte op til at lave et produkt, der løste problemet. Vi føler selv, at hvis man begyndte at benytte *Ud i det blå*, og det kunne opnå den rette popularitet og der i længden ville være en tilstrækkeligt stor base af områder og begivenheder, ville det helt klart kunne være en motiverende faktor til netop at få flere mennesker *Ud i det blå*. Afslutningsvis vil vi gerne konkludere direkte på de, af problemformuleringen, opstillede spørgsmål.

- **Hvordan kan et system udvikles responsivt, således at det præsenteres flot og er tilgængelig på enhver enhed med internet og browser?**

Vi kan konkludere, at den løsning vi har valgt, med at benytte Bootstrap som basis for vores UI-design og udvide med vores egne personlige stylesheets, har været en virkelig god løsning. Det har gjort, at alle elementer hurtigt blev stilet flot og gjort responsive. Det spiller rigtig godt med en ASP.NET Core løsning, da det UI-lag der er, består af HTML som modificeres server-side inden præsentation for brugeren. På den måde er hele systemet blevet responsivt og flot præsenteret stort set ligegyldigt, hvordan browseren bliver sizeret, og hvilken enhed der benyttes til at tilgå systemet.

- **Hvilke UI- og UX-overvejelser skal gøres for at opnå et intuitivt design i systemet?**
På denne problemstilling kan vi konkludere, at de overvejelser der er vigtigst at gøre for teamet er, hvem der er den relevante målgruppe at spørge. Disse overvejelser bør ikke ligge udelukkende hos teamets udviklere, da udviklerne har en klar opfattelse af præcis, hvordan hvert enkelt klik på en hvilken som helst del af siden præcist påvirker systemet. Derfor er det vigtigt at lave nogle brugertest, der kan fortælle om de valg, der træffes, faktisk gør systemet intuitivt. På den anden side skal der selvfølgelig gøres nogle overvejelser for at gøre det muligt at lave et første udkast. Her er det rigtig smart at se på, hvad man selv synes er smart, hvad andre har gjort, og hvad der populært set synes er smart. På den måde får man som udvikler og designer et godt indblik i, hvad der gør et system intuitivt og lækkert at arbejde i/benytte.

- **Hvordan kan det sikres, at de brugerindmeldte data der præsenteres i systemet, er pålidelige?**

På dette punkt kan vi konkludere, at vi ikke i den nuværende løsning har, hvad vi selv vil opfatte som en tilstrækkelig løsning. Den eneste sikkerhedsforanstaltning vi har på dette punkt, er brugerverificering. Vi har derimod, ligesom beskrevet i [8 Systemets fremtid](#), gjort nøje overvejelser vedrørende dette punkt. På den baggrund kan vi konkludere, at man først og fremmest bør have nogle systemadministratorer til at verificere de indkomne opslag fra nye brugere, men også at man kan have brugerlevels, der kan regulere, hvorvidt et opslag skal godkendes af en administrator, før det gøres tilgængeligt i systemet. Derudover er det smart at implementere et filtersystem, der undersøger opslag for upassende indhold såsom pornografi og bandeord. Et sådan system vil aldrig kunne finkæmme det hele uden at gøre systemet ubrugeligt, men det kan skære administratorernes arbejdsbyrde væsentligt ned, hvilket er en kæmpe fordel for sådan et system som *Ud i det blå*.

- **Kan systemet designes således, at det i højere grad motiverer brugere til at komme *Ud i det blå*?**

Som konkluderet tidligere, synes vi at netop *Ud i det blå* er et oplagt eksempel på hvordan et system kan designes således, at det i højere grad motiverer brugere til at komme *Ud i det blå*. Det er det blandt andet, fordi det giver mulighed for at undersøge områder i nærheden af brugeren, se anmeldelser, billeder og beskrivelser af stederne og på den måde få inspiration til at tage ud til nye steder og udforske sin omverden.

11 Bilagsoversigt

Alle Bilag er separate filer, for at gøre dem mere overskuelige at kigge på, under gennemlæsning af rapporten. Af denne grund er der lavet følgende overblik over bilag her i rapporten, således at der er et tydeligt overblik over, hvilke bilag der er, og hvor mange sider de er på. På denne måde er det ikke til at tage fejl af, om der mangler noget.

1. Kravspecifikation (12 sider)
2. Kildekodeindeksering (110 sider)
3. Use cases (9 sider)
4. Kanban Board (2 sider)
5. Brugerfeedback (3 sider)
6. Flowchart (12 sider)
7. Systemkort (2 sider)
8. ER-diagram (2 sider)
9. Modelklassediagram (2 sider)
10. Fuldt klassediagram (2 sider)
11. Logbog (5 sider)