



UCN

Bachelorprojekt PBA Softwareudvikling  
12. april til 16 juni

Skrevet af  
Anders Friang Jensen  
og  
Magnus Just Sund Carlsen

Hold: psu0221



### **Abstract**

This report delves into the use of technology to improve the consumption of our cultural heritage. This includes research into topics such as 3D model visualizations and AR.

Modern tools for guidance and map data are used to bring education out in the landscape.

GIS and GPS tools are considered as they get more advanced, mature and precise and they fulfill the needs of leading dissemination out of the cultural institutions.

The main aim of this report is to present a potential design and implementation to this issue.



## University College Nordjylland



Teknologi og business  
Professionsbachelor  
Softwareudvikling

**Projekt case:**

Kulturarv\_PBA\_Bachelor\_Projekt



**Hold:**

psu0221



**Antal normalsider:**

48



**Antal sider i alt:**

83



**Github:**

[https://github.com/5k41d/Kulturarv\\_PBA\\_Bachelor\\_Projekt.git](https://github.com/5k41d/Kulturarv_PBA_Bachelor_Projekt.git)



**Projektdeltagere:**

Anders Friang Jensen  
Magnus Just Sund Carlsen



**Afleveringsdato:**

16. juni 2022



**Vejleder:**

Brian Hvarregaard





# Indholdsfortegnelse

<b>Abstract</b>	<b>1</b>
<b>Titelblad</b>	<b>2</b>
<b>Indholdsfortegnelse</b>	<b>3</b>
<b>Introduktion</b>	<b>7</b>
Behov	7
Overvejelser	8
Kundesegment	8
Offentlige databaser og API'er til rådighed	10
Web Scraping	11
<b>Mission</b>	<b>13</b>
Strategiske mål	13
<b>Krav og systemdesign</b>	<b>15</b>
Klargøring af krav (MoSCoW)	15
Kvalitetskrav	17
“Internet Arkæologi” og standarder for data	17
Cloud Native	18
Domain Driven Design og SOLID	19
Arkitekturprincipper	20
Buy when non core	20
Build small, release fast, fail small	20
N+2 Design	21
Design for at least 2 axes	21
Fault Isolation	21
Asynchronous Design	22
Monitorering	22
Key Performance Indicators (KPI)	22
<b>Integration i virksomheden</b>	<b>24</b>
Organisationsopstilling	24
Team design	26
<b>Projekt styrelse</b>	<b>31</b>
Application Lifecycle Management	32
Software Development LifeCycle Management	34
RASCI og DevOps	36



<b>Gateway, Micro Services og Messaging</b>	<b>37</b>
Microservices implementation	38
<b>Messaging</b>	<b>40</b>
Yderligere udvalgte design mønstre	41
Aggregator	42
Event Sourcing	43
CQRS	44
Containerisering og orkestrering	45
<b>Navigation og Augmented Reality (AR)</b>	<b>46</b>
GPS	46
GIS og kort data	46
AR teknologi og 3D modeller	47
Agnostik tilgang	49
<b>Caching</b>	<b>50</b>
Caching strategi	50
<b>Database</b>	<b>52</b>
Valgt database paradigm	52
Grafdatabase	52
Design af database	53
Delkonklusion	54
Queries	54
Optimeriseringer og indeksering	57
Indeksering	57
Queries	58
Del diskussion	64
Sikkerhed og GDPR	65
JSON Web Tokens	65
<b>Test og produktkvalitet</b>	<b>66</b>
Teststrategi	66
Test logging	67
Defekt håndtering	69
“Definition of Ready” og “Definition of Done”	70
Test discipliner	71
Moq	71
Blackbox og white box testing	72
DAMP	72



TDD	72
Test Automations	72
Boundary Testing	72
Maze	73
Den agile kvadrant	73
<b>Diskussion</b>	<b>74</b>
<b>Konklusion</b>	<b>76</b>
<b>Referencer</b>	<b>78</b>



## Problemformulering

Bachelorprojekt PBA Softwareudvikling 2022

Digitalisering af historiske fund og kilder bliver mere udbredt, og disse præsenteres hovedsagligt hos de relevante museumsinstitutioner. Så snart at formidling bevæger sig ud i landskabet, er det både kostbart, svært at navigere og formidlingen i sig selv kan være en udfordring ved de fortidsminder der er blevet eroderet igennem tusinder af år, og derfor ikke er meget tilbage af. Tidligere løsninger har været enten ved brug af gammel teknologi, heftig akademisk præget information eller mangel på fængende og ønskede features.

- Hvordan kan man videreudvikle på digitaliseringen af historiske fund ved anvendelse af mere nutidige teknologier og design mønstre?
- Hvilke teknologier og design mønstre, kan anvendes for at sikre at skaleringen af data ikke påvirker senere udvikling?
- Hvordan aggregerer vi data for at bedre understøtte brugervenlig formidling til gængse brugere?
- Hvordan kan vi bruge nuværende teknologier til at hjælpe med at guide og navigere rundt i det landskabet, og formidlingen forbliver fyldestgørende?

Team: The Grasshoppers

Anders Friang Jensen  
Magnus Just Sund Carlsen

Vejleder:  
Brian Hvarregaard





# Introduktion

Denne rapsorts indhold beskriver processen, der har foregået gennem dette projekts forløb. Fra den initierende ide til den mere definerede case, der resulterede i en problemformulering. Herefter vil der blive identificeret og analyseret videre på hvilke behov der eksisterer, og hvordan en organisation vil blive starte og vokse omkring dette projekt. Dette munder ud i et systemdesign, som inkluderer diverse krav og principper der er nødvendige både nu, men også på sigt. Der vil derefter blive dykket ned i hvordan at dette system vil blive integreret i en bredere sammenhæng, samt hvordan blandt andet messaging integration kan hjælpe med at opnå diverse mål. Dette betyder at ikke bare systemet, men organisationen bliver håndteret på sigt. Der vil blive diskuteret mulige teknologier, der kan blive relevante i systemets levetid, og hvordan dette skal håndteres.

## Behov

Idag har museer, og diverse andre kulturinstitutioner i landet omfavnnet lysten til bruge flere ressourcer på netop deres videreförmidling i et teknologisk aspekt, for at nå ud til en bredere masse, og skabe en anderledes oplevelse (1).

Selvom teknologi i videreförmidling ikke er et nyt koncept, har man igennem årene stadig fået nye muligheder for at innovere netop indenfor området, og det er her muligheden for et nyt innovativt teknologisk formidlingsværktøj, kan træde i karakter. Idag har museerne mest fokus på intern formidling, og derfor begrænsrer sig til on-site besøgende i deres museumshuse.

Dette projekt har derfor til formål ikke kun at begrænse brugere til on-site besøg hos museerne, men at tage med dem ud i landskabet og besøge de fortidsminder, som der bliver fortalt om hos museerne (2).

En kombination af indlæring med et visuelt, og fysisk aktivt aspekt har stor indvirkning i oplevelses spektret hos brugerne. I samtale med Vesthimmerlands Museum og Moesgaard Museum opleves et behov, hvorpå at formidlingen kan hjælpe, guide og muliggøre et visuelt relationsskabende værktøj.

Det danske landskab indeholder mange fortidsminder, og er tilgængelig hvis man kender deres lokation, og her optræder behovet et behov der kunne opfyldes af et redskab.

Slots- og kulturarvsstyrelsen (3) har opsat en database (Kulturarv.dk (4)) med disse lokationer, som er én samling hvorpå fortidsminder er kortlagt og beskrevet.

Men disse er ikke præsenteret sådan, at man med begrænset teknologisk viden har let adgang til disse, selvom at dataen er offentlig tilgængelig.

Derfor er der også et behov for at kunne hente denne data, og beskrive den let sommeligt for slutbrugerne.



Derudover ønskes der også af projektet at turister og amatører, som kan bidrage til kulturarven ved at kunne sende data til de relevante eksperter der varetager diverse fortidsminder.

## Overvejelser

Slots- og kulturarvsstyrelsen har med egen database muliggjort, at man kan hente GIS (Geografisk Informations System) (5) data, og vise det på et virtuelt kort på selv de mindste enheder.

Et GIS system har adskillige løsninger, og tilføjelser for at kunne få så meget information vist på et kort. Eftersom at danske fortidsminder er repræsenteret i landskabet, og kan skabe problemer både med viden om deres tilstedeværelse, men også navigeringen vil et GIS system kunne assistere.

I forlængelse af dette, er adskillige fortidsminder enten kraftigt eroderet, forsvundet i landskabet, delvist- eller helt ødelagte, eller svære at sammenfatte i en større sammenhæng ved lokationen (6), derfor er GIS igen et stærkt værktøj i at spotte disse i landskabet.

Derudover er brugen af AR (Augmented Reality) (7) teknologien også blevet mere moden, og populær, hvilket i nogle formidlingsprocesser, bærer en stor rolle i netop at løse fornævnte problem. Derfor er dette en teknologi der både er interaktiv, men også en god løsning i netop at sammenfatte information, når brugeren besøger diverse fortidsminder.

Hvis museer skal kunne bruge produktet, som et videre formidlingsværktøj uden for museumshuset, vil en GPS rute, og guide også være at foretrække. Dette ligger til grund i fornævnte, hvorpå nogle fortidsminder kan være svære at finde i landskabet.

Oplæsning og letlæselighed af information er også en feature, der skal overvejes, da spektret for kundesegmentet indebærer både børn, ældre og visse funktionsnedsatte brugere.

Tilsammen vil disse teknologier hjælpe med at videreforsmidle til alle interessenter indenfor kultur og fortidsminder i landet, og produktet kan spredes til resten af Europa.

## Kundesegment

Siden at dette projekt omhandler videreforsmildingen af kulturarv ikke bare i Danmark, men på sigt hele Europa, så vil det give mening at identificere projektets interesser. Hvis man tager udgangspunkt i Danmark, er den åbenlyse interessent kulturstyrelsen, da de er hovedansvarlig for fortidsminderne i Danmark (8).



En stor interessen i dette projekt er også landets museer og kulturoplevelser, der netop varetager landets fortidsminder, og relevante tilknyttede artefakter. Dette skyldes at systemet kan hjælpe med at oplyse blandt andet turister og andre interesserende grupper. Dette kan gøres, ved at vejlede dem om, hvilke kultursteder der er i nærheden, men kan også videreforside information om stederne og eventuelle tilknyttet events, der kunne være spændende.

Systemet skal derfor tilbyde en række funktionaliteter til museer, der vil tiltrække et større publikum til deres udstillinger, men også give et spotlight til det givne museum's events, der vil blive præsenteret sammen med relevant information omkring deres udstillinger.

Turister vil derimod få en platform der gør det nemmere at finde kultur-relevante oplevelser, og lokationer. Det kan enten være i nærområdet eller i forbindelse med en ferie. De vil få en central lokation til både at opnår information omkring mulige lokationer, men også en simpel løsning der kan hjælpe med at navigere derhen. Dertil skal projektet også give mulighed for interessenter at udforske steder med relevans indenfor specifikke tidsalder.



## Offentlige databaser og API'er til rådighed

For at projektet kan igangsættes, er der et behov for data, som slutbrugeren kan gøre nytte af. Men at skulle indsamle data fra Danmarks og Europas fortidsminder, kan være en langsommelig og kostbar process, hvis dette gøres manuelt.

EU har siden 2008 lanceret og sponsoreret et projekt, hvorpå man forsøger at samle data om fortidsminder i én database, kaldet *Europeana* (9).

Denne database er til rådighed for offentligheden, og via deres REST API'er vil man kunne tilgå information, dokumenter, 3D modeller og bøger mfl. Hver API har deres eget formål, hvorpå søgekriterierne skifter karakter i form af søgekriterier.

Dertil har den danske stat også databaser, og API'er til rådighed. Her er der tale om blandt andet kulturarv.dk (3). Fordelen ved at gennemsøge Kulturarvsstyrelsens database, hjælper med specifikt at få historisk data fra danske fortidsminder, hvilket er første skridt i dette projekt. Europeana vil derfor kun være i undersøgelses stadiet, indtil produktet spredes til andre lande i Europa.

Ved at udnytte lokationsdata fra kulturarvs databasen, og eventuelle information der er sammenkædet, vil gøre det muligt at skabe GPS ruter, GIS kortdata og lokationsbaseret formidling, i det danske landskab.

Men for at godtgøre Europeana senere i projektet, skal det være muligt at sende data til deres API'er om ændringer eller tilføjelser af historisk data, som EU verificere og gemmer hvis disse accepteres som gyldige. Her tænkes der hovedsageligt 3D modeller. Dette gør det muligt for amatører og turister at bidrage til kulturarven i europa. Denne feature har Kulturarv.dk dog ikke. Derfor skal der udtænkes en midlertidig løsning, baseret på 3D modeller af danske fortidsminder.

At indsamle information fra fornævnte API'er og websteder, vil det derfor være relevant at undersøge *web scraping* (10).

Det vil gøre det nemmere at visualisere de relevante API'er, og deres funktion.



## Web Scraping

I sammenhæng med fornævnte offentlige databaser og API'er, er konceptet web scraping relevant (11).

På denne måde kan indholdet i diverse filer (HTML, CSV mfl.) blive gennemsøgt efter relevant information til formidling, og visualisering i det kommende produkt.

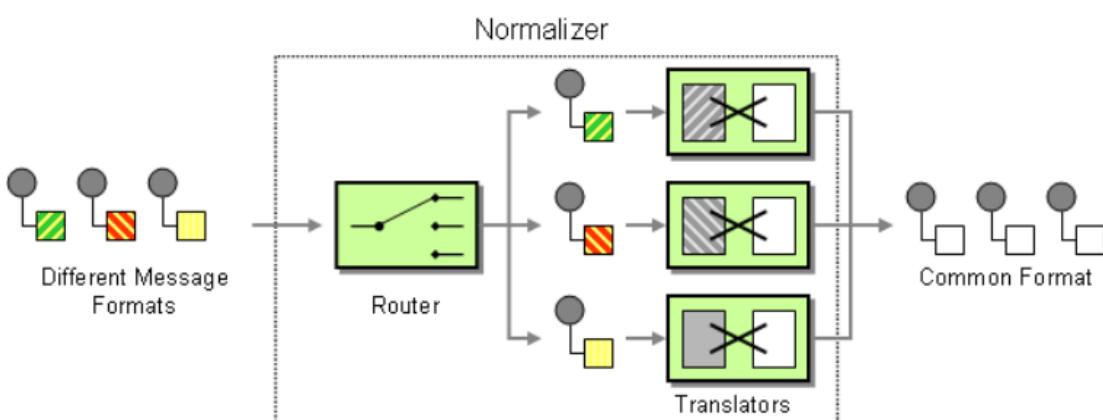
Kulturarvs hjemmeside returnerer en HTML fil, når der udføres en forespørgsel med deres URN (Uniform Resource Name). I denne HTML fil, ligger der information omkring et fortidsminde.

Værktøjer såsom HTML Agility Pack (HAP) (12) for .NET, kan parse igennem HTML filen, og angive de HTML tags, hvor den ønskede information ligger.

`https://www.kulturarv.dk/fundogfortidsminder/Lokalitet/{id}`

*URI for søgning efter lokalitet for diverse fortidsminder. Efter lokalitets path angives et ID, der er tilknyttet et fortidsminde.*

Implementerer man en web scraping automatiseringsrobot, kan man bruge andre værktøjer sammen med, afhængig af det data der bliver returneret..



Billede 1. Normalizer mønsteret (13).

Web scraping skal derfor indeholde et *Normalizer* implementering, for at oversætte diverse eksterne dataformater, om til et fælles system data format.

Dette vil hjælpe med skalering, også med *Maintainability* af produktet



```
...
<td align="right">
    ...
    <a class="LocalNavigation"
        href="http://www.krak.dk/query?what=route&addr2=Lindholms
Højene|9.91218335598494|57.0773217870838&addr1=|||" target="_blank">Ruteplanlægger</a>
    </td>
</tr>

<tr><!-- Empty row for spacing before location description text-->
    <td colspan="2">&ampnbsp</td>
</tr>
<!-- Start free text for location -->
<tr>
    <td colspan="2">
        <font class="locNormalText">
            På Lindholm Høje i Nordjylland ligger en stor gravplads med
            begravelser fra en hen ved 600-årig periode. De ældstegrave er fra 400-årene, og de yngste er fra vikingetiden i
            900-årene.
            Limfjorden var i hele oldtiden et vigtigt farvand for rejser til og fra de nordiske lande. Fjorden var i vikingetiden også
            udgangspunkt for togter til England og resten af Europa. Dertilens skibe kunne let sejle op ad Lindholm Å til Lindholm
            Høje. Her var den sandede jord let at dyrke, men samtidig sårbar over for sandflugt.
            Omkring år 1100 har sandflugten været så voldsom, at Lindholm Høje blev forladt. Sandet dækkede stenene indtil
            fortidsmindet blev fredet i 1901 og udgravet i 1950'erne.
        </font>
    </td>
</tr>
<!-- End free text for location -->

</table>
</div>
</div>
...

```

Billede 2. Eksempel på HTML hvor (markeret med grøn) er gemt information der ønskes.

Resultatet kan derefter gemmes som lokal fil, eller i en database, efter hvilken integrationsstrategi man foretager sig. Dette uddybes yderligere i afsnittet *Micro Service implementation*.

Ligesom med HTMLAgilityPack findes der også XML parsers, JSON parsers mfl, der med samme princip stripper et dokument, til de værdier man ønsker.

Disse har ikke været relevante endnu, men er vigtige at være ørvågen over for hvis sådanne dokumenter skulle forekomme, og indgå i *normalizeren*.

Dermed skal ændringer til nye datatyper kun foretages ét sted, og ikke i flere steder i det bagvedliggende system.

Europeana databasen har som før nævnt også en tilgængelig API, som robotten kan lave forespørgsler til.



Denne API kræver en API-nøgle, som kræves ved hver forespørgsel.

Denne API returnerer derimod et JSON dokument, hvor (efter hvor specifik URI'en der givet) indeholder en række informationer der matcher ens søgekriterier.

---

```
{"apikey":"igisoryle", "success":true, "requestNumber":999, "itemsCount":12, "totalResults":501, "items":[{"completeness":0, "country":["United Kingdom"], "dataProvider":["Archaeology Data Service"], "dcCreator":["P Rowe"], "dcCreatorLangAware":{"en":["P Rowe"]}, "dcDescription":["A Bronze Age burial mound stands at the north-west corner of the Upleatham Hills. A loose cup marked boulder was identified within a small quarried area to the north of the mound in 1999. It is presumed that this has eroded from the mound or has been removed in an unrecorded 19th or early 20th century excavation."]}]
```

...

---

*Billede 3. Eksempel på dele af et returneret JSON dokument, med informationer om gravhøje (burial mound) i Europa.*

*Der er sendt en række søgekriterier med, hvor en af dem (markeret med grøn), er API nøglen.*

I web scraping er man nødt til at kende til de givne API/hjemmesider, man ønsker at forespørge.

Efter man har fået kendskab til disse kan robotten bygges.

Men inden at robotten tages i brug, skal projektets strategiske mål, og formelle dele af virksomheden designes.

## Mission

For bedst at kunne styre projektet i den rigtige retning, vil det være nødvendigt at udforme en mission for organisationen.

*"Målet for projektet er at øge interessen i den danske og europæiske kulturarv, ved at videreforsmidle information på en let tilgængelig måde."*

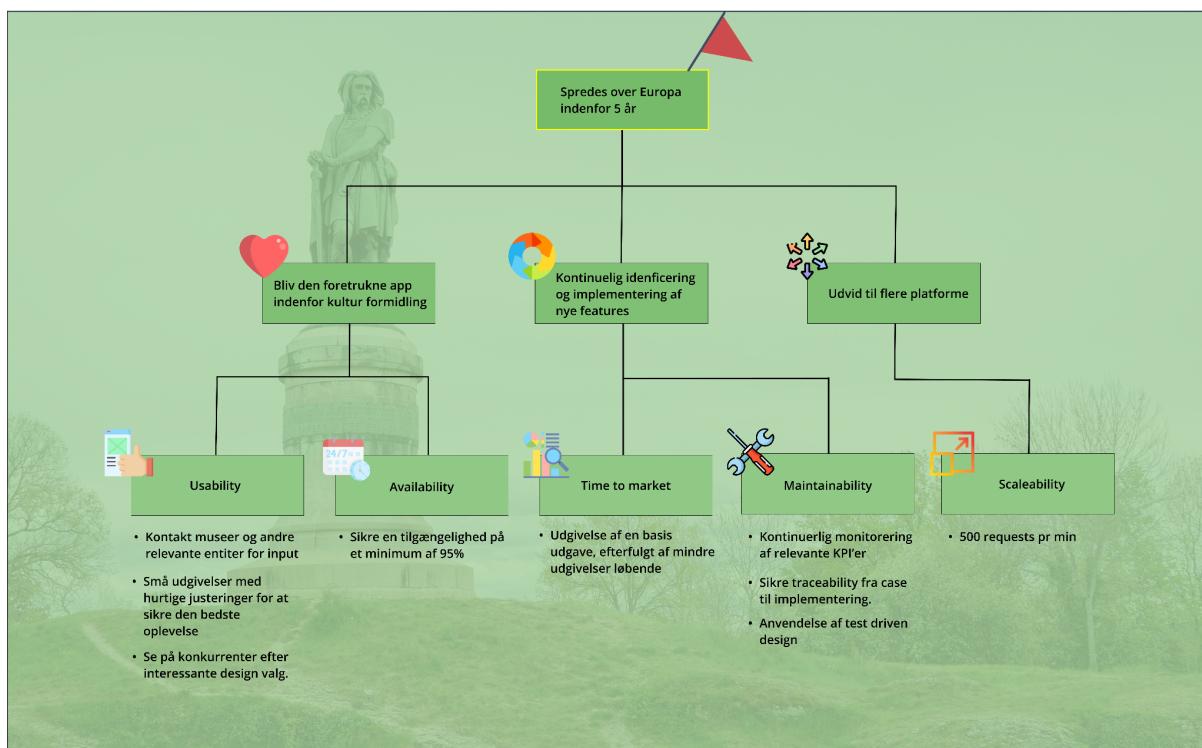
Efter at missionen er opstillet, vil man ud fra denne kunne opsætte mål og strategier for at opnå denne.

## Strategiske mål

Når man ser på ønsket omkring vækst, virker det meget generelt, og ikke særlig håndgribeligt. Til at konkretisere dette vil en opsætning af strategiske mål for at gøre organisationens vækst mere håndgribelig. Derudover vil strategiske mål også give en fælles forståelse (på tværs af organisationens interesser) for de mål og målbare handlinger der skal foretages, for at opnå den mission der er stillet. Disse mål bliver



bliver opdelt yderligere for at kunne dele det op i mere håndgribelige opgaver der kan bedre defineres, og indgå i udviklingsprocessen. På det følgende billede er projektets strategiske mål visualiseret.



Billede 3. Strategiske mål.

For at opfylde ønsket om at få en vækst der kan sprede sig ud til Europa over en periode af fem år, så er der blevet identificeret tre undermål. De består af at sikre kompatibilitet med adskillige platforme, så kundesegmentet rammes bredere. Der ønskes også et fleksibelt system, hvor identificerede nye features på kort tid kan indgå i produktet. Til sidst ønskes der at projektet bliver den foretrukne app til kulturformidling i Europa. For at blive den foretrukne app (med fokus på kundesegmentet), vil projektet fokusere på usability og availability, eftersom at brugerne nødvendigvis ikke er tech specialister, og at appen skal kunne tilgås på alle årets dage. Usability sikres ved at kontakte relevante interesser med erfaring indenfor kulturformidling, samt sikre en høj tilgængelighed.

Med kontinuerlig identificering af features, vil fokus være på time to market og maintainability. Her vil fokusset ligge på at producere et tilfredsstillende produkt, der stadig kan ramme markedet i tide for at opfylde projektets strategiske mål. Samtidig for at kunne have denne konstante implementering af nye features, sættes der fokus på at holde et produkt med god monitorering og traceability, samt anvendelse af test driven design.



Til sidst for at bedst kunne understøtte at udvide til flere platforme, skal der være et fokus på *scalability*. Til at sikre at systemet kan understøtte den ekstra mængde af trafik der kan forventes når produktet breder sig til flere lande, samt uforudsigelige antal brugere i startfasen, skal performance og tilgængelighed også være i fokus. Mere præcise og målbare kriterier fra fornævnte strategier, og delmål kan ses i afsnittet *Monitorering og Key Performance Indicators (KPI)*.

Disse udvalgte strategier og delmål er taget udgangspunkt i det behov, og det kundesegment projektet er rettet imod.

## Krav og systemdesign

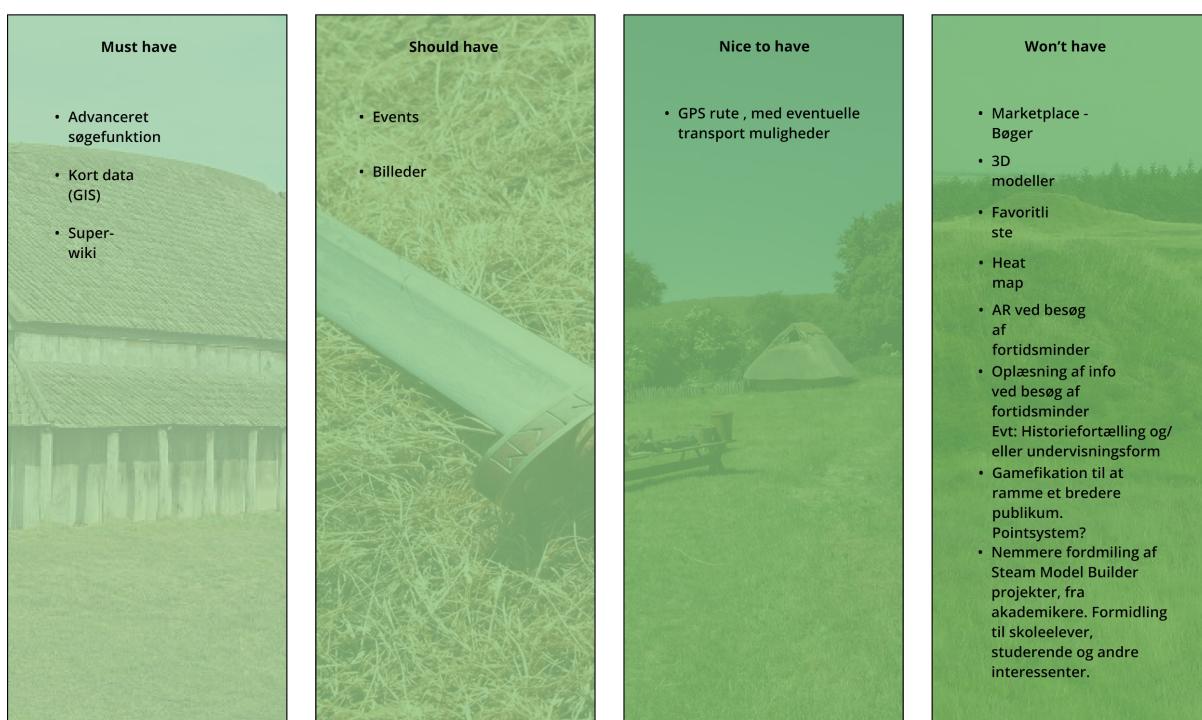
I forlængelse af mål og strategier, vil projektet nu kunne tage form, og dermed forme kravene efter det behov der er stillet, og i kommunikation med kulturinstitutionerne (hovedsageligt museer) kan yderligere behov/krav identificeres.

Når kravene er identificeret vil designet også kunne tage form. Dette betyder at system designerne kan foretage strategiske designvalg i form af arkitektur, og design mønstre for at imødekommme kravene for at implementere den mest effektive løsning. Men vigtigst af alt skal de kunne foretage designvalg for fremtidige skalering af produktet. Med dette i mente skal systemets krav først prioriteres, og dette kan gøres ved brug af MoSCoW modellen, hvor de vigtigste krav i nuværende fase fastlægges, men også krav der er overvejet for fremtiden, og dermed ikke indtræder før senere versioner eller releases af produktet.

## Klargøring af krav (MoSCoW)

For at udviklingsprocessen kan igangsættes, og at projektets interesserter har klart overblik over kerneproduktets krav i første version, er MoSCoW modellen taget i brug. I et interview med Kim Callesen fra Vesthimmerlands Museum (14), samt redegørelse for kundesegment kan man nu gå i gang med krav opsætning.

Eftersom projektet er styret af en agil projekt metodologi, og fokus er *Time to Market* (15), er produktet delt i små udgivelser/release, hvori at hver release indeholder et sæt af nye features. På denne måde kan produktet valideres hurtigere, og indsamle vigtig information om brugernes tilfredshed.



Billede 4. MoSCoW model for 1. version af produktet.

Derfor vil produktet ikke have alle features klar og implementeret i version et. Et udvalg af kerne features er derfor prioriteret, og derefter udvælges de sekundære features.

Selvom at kortdata og formidling ikke er de differentierede faktorer i produktet, er det stadig den del af produktet, de andre features bygger ovenpå.

Uden guide til fortidsminder og formidling, vil engagementet til at besøge og finde de pågældende fortidsminder være formindsket.

Produktet skal også bringe turisterne ud i landskabet til netop at besøge og opleve disse fortidsminder, og derfor er det vigtigt at kerneproduktet bygges på kortdata (GIS, GPS), og formidling.

Dernæst vil oplevelses faktoren kunne øges ved at implementere mere differentierede features, såsom 3D modeller i samarbejde med Augmented Reality (AR).

Dertil har MoSCoW modellen også den fordel, at projektets risikovurdering ved overskredet deadline, manglende kapital etc., gør det muligt at kunne vurdere arbejdsopgaver efter relevans (16).

En foreslået regel fra *BlivProjektLeder.dk* prioriterer man sine krav efter 50:30:20 reglen. Dette vil sige at 50 procent af nuværende krav placeres i *Must Have*, 30 procent i *Should* og 20 procent i *Nice to Have*. *Won't Have* er ikke en prioriteret del af succesen af version 1 af projektet, og indgår derfor ikke i denne regel. Dog er det stadig vigtigt for projektledere og andre interesserter, at vide hvilke krav der har været overvejet, og hvilke krav der kan tiltræde i kommende releases af produktet.



## Kvalitetskrav

Kvalitetskravet for produktet er afspejlet ved de designkrav, og ønsker der stilles til produktet.

Et produkt der skal videreformidle kulturarv til turister, rundt i hele Danmark og senere Europa, vil tilgængeligheden af produktet være i fokus.

Dertil skal det også være nemt for brugerne at bruge produktet for at ramme en bred kundebase, samt performance for at sikre tilfredse brugere.

Men dertil kommer der også etiske, og kvalitetsmæssige krav i form af den information der bliver videreformidlet til brugerne.

Dette forekommer som eksterne krav (FURPS+ (17)), som blandt andet indebærer standarder for den data der bliver videreformidlet, GDPR af brugerens data og præcision af 3D modeller af fortidsminderne.

## "Internet Arkæologi" og standarder for data

Eftersom produktet skal være en kilde til information, og formidle denne til turister, og interesserter der er afhængige af det data der tilbydes, skal kvaliteten af denne data derfor være høj. Men hvad betyder dette?

Kevin Garstki har skrevet en afhandling (Digital Innovations in European Archaeology (18)) der netop belyser dette område, og hvordan man som formidler i digitale og teknologiske sammenhænge, skal være opmærksomme på de forekomster af forkert, korrupt eller ikke-sammenhængende data.

Derfor er der forskellige standarder og modeller der i det professionelle miljø til videreformidling, bliver gjort brug af. Her der tale om blandt andet *Conceptual Reference Model (CRM)* (19), *The LEAP Projects* (20) og *Linked Open Data* (21).

Disse koncepter og digitale publicerings-miljøer, er et forsøg på at skabe kvalitetsdata, samt standarder for det data der bliver udgivet.

Kevin belyser også manglende samarbejde mellem landene i EU, hvorpå centraliserede databaser skal hjælpe med et struktureret og lettere tilgængeligt data.

Dog har EU's projekt med Europeana databasen, forsøgt at fikse dette problem, men succesen af dette projekt afhænger af de europæiske museer og statsstyrelser, hvilket Kevin også lægger vægt på.

Som fornævnt tilbydes der i Danmark den offentlige database, *Kulturarv.dk*, (3)) til kulturformidling. Og i forlængelse af afsnittet *Offentlige Databaser og API'er*, skal *Kulturarv.dk* virke som et centraliseret punkt, hvorpå man kan opnå lokationsdata, information i forskellige filformater for Danmark. Med tiden som at andre lande i EU følger med, og begynder at anvende en lignende format, vil dette system kunne fungere som et bindelede i videreformidlingen.



Eftersom målet for projektet er at nå ud til Europa, er det derfor vigtigt at systemet er fleksibel, og kan imødekomme fornævnte problemer. Dertil skal systemet formidle data efter de standarder indenfor etik og kvalitet, der er fremlagt i (10), (11) og (12).

I dette projekt vil dette ikke være komplekst at løse, da disse standarder er overholdt af de databaser/API'er projektet henter data fra.

Der hvor det bliver et problem at løse, vil være *Accurate Registration* (18), som netop fokusere mere på den visuelle repræsentation af historiske objekter, og deres nøjagtighed.

Som kan ses på billede 4, i MoSCoW modellen, er 3D modeller og Augmented Reality, nogle krav projektet indtager i fremtiden.

*Accurate Registration* er en standard der netop besæftiger sig med placering af objekter i et 3D miljø, samt placering i et hybrid miljø såsom Augmented Reality.

Dertil kan tilføjes en samling af rapporter og afhandlinger (redigeret af Erik Malcolm Champion), der netop også belyser dette område, og kan læses i (1).

Dette er et kvalitetskrav der bliver belyst uden for de interesserter, projektet har haft inde over.

Det skal dog nævnes at ingen af ovenstående er love der skal overholdes, men blot standarder præsenteret af arkæologer, historikere mfl, for at opnå den bedst mulige og præcise formidling.

Eftersom projektets krav (inkl. kvalitetskrav) er undersøgt, og bragt frem i lyset, vil det være en god mulighed at undersøge de design og den arkitektur der vil løse disse bedst. I dette projekt er *Cloud Native* (22) taget i brug, grundet dens fleksibilitet i både design, skalering og vedligeholdelse.

## Cloud Native

En effektiv og billig løsning ligger i brugen af tilgængelige *Cloud* (23) funktionaliteter samt løsninger.

Cloud Native opsætter nogle kriterier for hvordan en service er bygget med Cloud som fokus.

Elementer såsom Micro Service mønstret (24), spiller en afgørende rolle i skalering, og hosting i cloud og samtidigt udnytter cloud platformen effektivt.

Dertil er *containerisering* (25) af en service for at sikre kompatibilitet og kørbarhed i et isoleret miljø. Hertil menes der kontra lokale eller selv hostede miljøer.

Dette stiller også krav til udviklingsprocessen, og hvordan denne eksekveres og håndteres efterfølgende.



Her er Continuous Integration- og Deployment (26) koncepter der bringes ind i udviklingsprocessen, samt DevOps team's (26) der håndtere vedligeholdelse, monitorering, opsættelse af test- og live-miljø.

Cloud Native er mere et princip, og mål for en service frem for et konkret implementering design, og er forskelligt fortolket af virksomheder og designere.

Derfor kan man dykke ned i de nødvendige koncepter som før nævnt, og dertil kan der tillægges et designprincip, der hænger tæt sammen med Micro Service mønstret blandt andet Domain Driven Design (DDD) (27). DDD bygger på kontekst præget udvikling, og fælles sprog på tværs af fagligheder i udvikling af produktet.

## Domain Driven Design og SOLID

Eftersom at projektets fremtidige krav og kompleksitet er uvist, vil et DDD design imødekomme SOLID (28) principperne.

CQRS (29) passer også ind i DDD, da man ved nye implementationer opretter nye filer frem for ændringer af nuværende.

På denne måde vil nye implementeringer kunne foretages uden ændring af nuværende kode, samt vil test af nye implementationer forekomme isoleret fra resten af kodebasen.

Derfor vil tilføjelser ikke have nogen indflydelse på resten af systemet, som kan koste tid og ressourcer at fikse samt vedligeholde.

Kombinationen af DDD i samarbejde med CQRS vil hjælpe med at holde systemet struktureret, i takt med at størrelsen stiger i samarbejde med kompleksiteten.

I samarbejde med *Cloud-Native* arkitekturen vil man kunne dele hver service op i sin egen *Bounded Context* (27), hvorpå at en upstream relation vil være en gateway.

Et eksempel på en *Entity* (27) vil være et fortidsminde, i konteksten af Gateway, i modsætning til Search service.

Modellen skal beskrive hvad et fortidsminde er, og indeholde den information der kun er tilhørende domæne logikken til et fortidsminde.

Resterende information der pålægges i eventuelt en gateway, som også har sin egen domæne logik, kan anses for at være *Value-Objects* (27), eller endda sin egen *Entity* model.

Men hvilke kontekster indeholder projektet? Projektet kan anses for at være todelt. Der er klientsiden og tilgængelige services. Disse to er bundet sammen med gateway mønstret (29).



Eftersom at klientsiden i nuværende design kun består af mobilapplikation, skal der overvejes implementeringer af diverse andre platforme.

Og for at overholde SOLID principperne, hvor *Interface Separation* (28) princippet ikke bliver overtrådt, og derfor er der behov for en kontekstmodel for hver af platformene.

Sammen med Micro Service Mønstret kan man med *Single Responsibility* (28) princippet og message bus kommunikation skabe et *swimlane* (30) design, hvor at ændringer i en service ikke påvirker en anden, og dermed bliver testing af disse services også mere robust.

Dertil vil adskillelse af services' implementationer også give mulighed for *fault isolation* (30), som bidrager til bedre tilgængelighed af systemet.

Med disse designprincipper på plads, vil det give mening og se på hvilke principper der vil have betydning for organisationen og produktet.

## Arkitekturprincipper

Som et led i opsætningen af kravet og systemdesignet, bliver det nødvendigt at definere nogle af de generelle strategiske områder, der er relevante og hvilke principper der vil blive anvendt for bedst at opnå dem. Dette vil blive gjort ved at tage udgangspunkt i de strategiske mål, der allerede er blevet udformet tidligere i rapporten. Med dette er der taget udgangspunkt i 6 principper der vil blive fokuseret på.

### Buy when non core

Når det kommer til dette system er der mange emner og teknologier, der er udenfor det scope, der ønskes at fokusere på. Til at håndtere dette anvendes der "Buy when non core". Forklaringen af dette er at systemet kræver mange mindre dele, der ikke er en del af kerne funktionaliteten. Dette inkluderer blandt andet implementeringen af GIS teknologien, authentication, og på sigt muligheden for forretnings interaktioner. Alle disse har diverse krav for at kunne leve op til den standard, der er sat i industrien. Grundet dette anvendes der tredjeparts teknologier, hvor at dette system vil udelukkende fungere som bindeleddet (31).

### Build small, release fast, fail small

Som en del af udviklingen af systemet, vil der være adskillige ændringer undervejs. For at hjælpe med dette anvendes dette princip om at bygge småt med hurtige releases. Denne fremgangsmåde hjælper specielt med at opnå de opsatte strategiske mål om bedre time to market. Dette princip vil specielt træde i kraft efter de første versioner er



ude, da det kan give mere fleksibilitet i udviklingsprocessen. Dette kan opnås ved specielt at oprette og anvende *Continuous integration* i systemet (31).

## N+2 Design

Grundet ønsket om høj tilgængelighed, vil det være nødvendigt at anvende N+2 design. Dette design vil sikre at der er reserve instanser tilgængelige i tilfælde af at den første fejler. Dette bliver nødvendigt så at tilgængeligheden ikke tager et stort slag, i tilfælde af tab af strøm, defekt eller vedligeholdelse (31).

## Design for at least 2 axes

Med udgangspunkt i ønsket om at med tiden kunne skalere til Europa, så er *scalability* et meget centralt emne. Ved at designe for at kunne skalere på mere end *en* akse, gør det muligt at kunne håndtere de 500 beskeder i minuttet der ønskes. Dette princip bliver videre understøttet af brugen af microservices.. Siden denne arkitektur allerede promovere meget fokuserede services, gør det brugen af blandt andet x aksen nemmere. Dette sker siden at servicene allerede er meget fint opdelte, så ved skabelse af ekstra instanser, er de langt mere fokuserede (31).

Selve organisationens struktur, samt brugen af *containers/orchestration* værktøjer, vil skalering på z-aksen også blive nemmere at udføre.

Er organisationens teams individuelt process styret, og isoleret i håndtering af produktets dele, vil skalering til eksterne lokationer i Europa være nemmere.

Og duplikering af systemet vil også kunne håndteres gnidningsfrit med fornævnte værktøjer.

## Fault Isolation

I kombination med brugen af design for flere akser, vil der også anvendes fejl isolering. Dette er en anden årsag til brugen af microservice arkitekturen, og adskillelsen af diverse platforme såsom mobile, web mm. Gennem fejl isolering og disse adskillelser mellem platforme, opnås der muligheden for ikke bare at udføre vedligeholdelse og opdateringer, men også at håndtere at en fejl på andre platforms services ikke vil påvirke andre. Alt dette har til formål ikke bare at øge tilgængeligheden, men også muliggøre nemmere implementering af nye features, siden at der er en klar adskilning af services (31).

Dette princip kan også hjælpe med versionsstyring af diverse services, da man kan implementere nye version af services, og skulle disse fejle vil dette heller ikke skabe problemer (routing til tidlige version, hvis dette skulle ske).



## Asynchronous Design

Til at bedre muliggørene skaleringen og forbedre håndteringen af microservices og messaging i systemet, vil det også være nødvendigt at se på muligheden for asynkront design. Brugen af blandt andet messaging og microservices, kræver at der anvendes asynkront design, da det ikke vil være muligt at understøtte mængden af kald hvis de var synkrone. Dette er blandt andet fordi de blokere, mens de venter på svar. Dette vil være umuligt at vedligeholde specielt ved et system, der har hensigt på at gå ud over landets grænser med tiden. Dette vil selvfølgelig komme med nogen komplikationer, med løsningen, da dette vil gøre debugging sværere og koden mere komplekst (31).

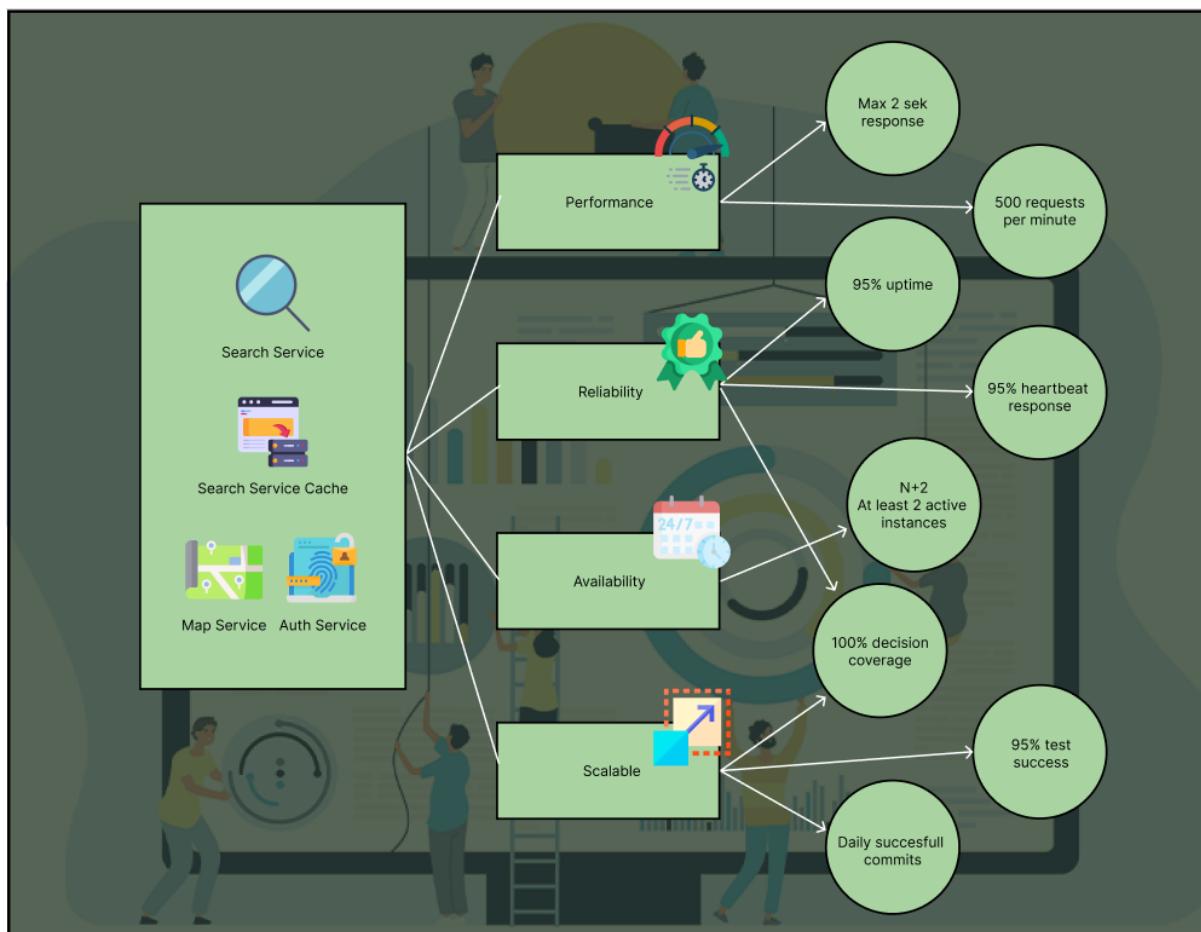
## Monitorering

Som et led i at sikre bedre kvalitet af software, såvel som at forbedre håndteringen af problemer ved at spotte dem tidligt, så er monitorering et vigtigt emne at håndtere. For bedst at håndtere monitorering og anvende den information der er tilgængelig er det meget vigtigt at anvende et værktøj der kan hjælpe med denne proces. Dette værktøj skal være med til blandt andet monitorere netværk, såvel server trafik og applikationens performance. Dette kan gøres af diverse værktøjer indenfor monitorering.

Hostes applikationen i et Cloud miljø (hvilket er planen) ejet af en 3. part, vil flere af disse håndteres af deres platform, herunder server trafik, netværk og performance. Derudover vil den nødvendige hardware kunne opskaleres ved abonnementsaftale (32). Derfor står projektet i det stadie hvor monitorering af kerne KPI'er er mere i fokus. For at kunne anvende monitorering er det vigtigt at vide, hvad der ønskes at monitorere og hvilken værdi denne monitorering kan give. Med udgangspunkt i dette vil de valgte kerne KPI'er blive etableret.

## Key Performance Indicators (KPI)

For at kunne monitorere et system ordentligt med bedst mulige effekt er det nødvendigt at finde hvilke indikatorer der findes. De vigtigste af disse er hvad der er beskrevet som Key Performance Indicators. Til at bedst finde de KPI'er der er relevant for dette projekt, tages der udgangspunkt i de opstillede strategiske mål. Til dette er der opstillet følgende figur.



Billede 5. Opstilling af vigtige mål med mulige KPI'er.

Billede 5 viser nogle af områderne hvor at KPI'er kan spille ind og have et betydeligt impact. Her vil det give mening at lægge vægt på blandt andet performance. I forbindelse med dette vil en vigtig indikator blandt andet være den ønskede maks responstid på 2 sekunder. Dette er med til at hjælpe med ønsket omkring høj performance. Uover dette er det også vigtigt for systemet at kunne håndtere mængden af requests, der er påkrævet. Dette medvirker til at gennemløbet af beskeder bliver en anden vigtig KPI. Dette er specielt tilfældet så at der kan håndteres de 500 requests pr minut, som blev defineret i de strategiske mål. Denne KPI er specielt vigtig, da at kravet til denne kan vokse baseret på skaleringen af projektet. Den næste relevante KPI vedrører pålideligheden, da at der er et strategisk mål om at have en uptime på minimum 95%. Den næste KPI med relevans for systemet vil være et heartbeat for at sikre at diverse microservices er tilgængelige. Dette er specielt relevant når der anvendes N+2 design, som en af de udvalgte. Den sidste der vil være relevant at se på er at monitorere interaktionen med cachen. Dette inkludere både cache hits og misses, så at den kan fintunes mere, baseret på den geografiske lokation blandt andet (33).



## Integration i virksomheden

Integration handler ikke kun software produktet, men er også en vigtig del i skaleringen af selve virksomheden og de mennesker der former den.

Forskellige afdelinger i virksomheden er afhængige af forskellige typer af software til at udføre de relevante arbejdsopgaver. Menneskene i virksomheden er afhængige af samarbejde, kommunikation og vidensdeling. Dertil er menneskene i virksomheden også bundet sammen i en eller anden form for team, og når disse skal skaleres sammen med virksomheden, skal visse overvejelser gøres.

Et eksempel på en case hvor skalering af virksomheden er emnet, vil være en økonomiafdeling med softwaresystemer til at håndtere løn, omkostninger i virksomheden mfl.

Disse softwareløsninger er ikke et redskab udviklingsafdelingen i virksomheden vil have nytte af.

Men data/viden fra disse systemer kan være nødvendige, at kunne deles på tværs af virksomhedens afdelinger.

Et psykologisk aspekt i adskilt viden, ansvarsområder og kommunikation, kan i tilfælde føre til affektive relationer på tværs af afdelingerne. Dette er noget der skal tages hånd om for at få en sund- og skaleringsklar organisation.

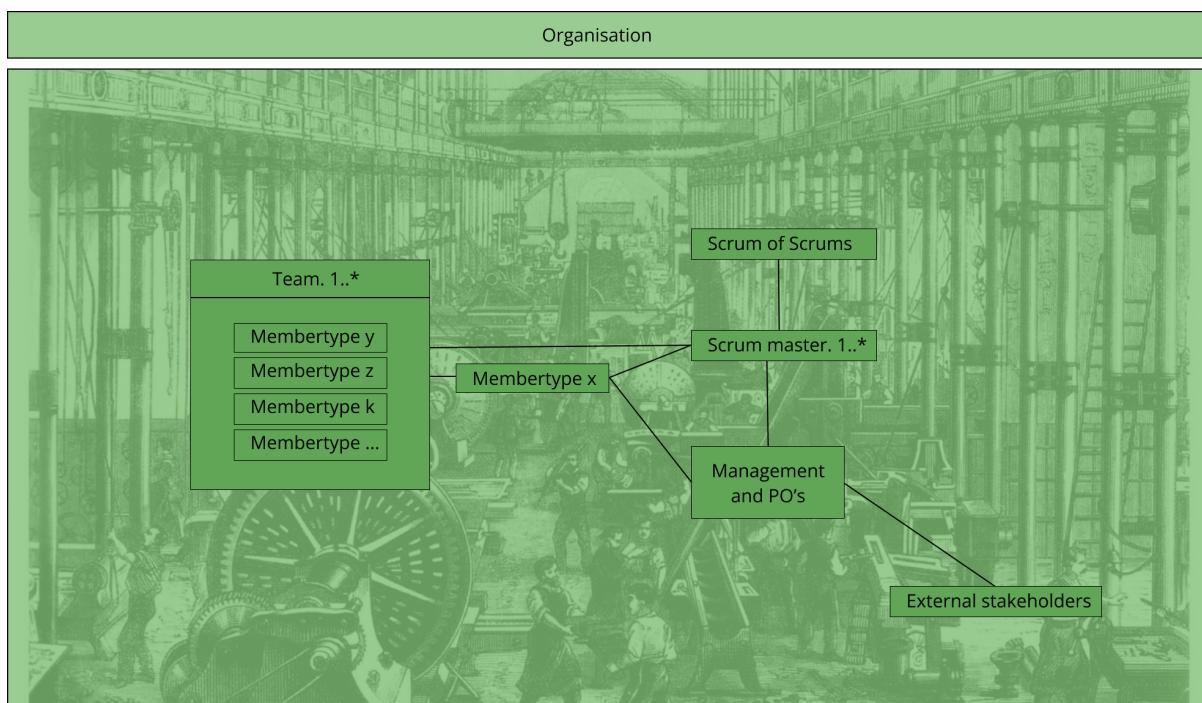
Et sidste punkt vil være skalering af teams, hvorpå man ikke bryder mennesker op, der er afhængige af hinanden eller arbejder godt sammen, da forvirring og gnidninger kan forekomme, og dette kan føre til modstand.

Men for at starte et sted, kigger man på organisationen, og hvordan strukturen skal opsættes for at imødekomme projektet/produktet krav og mål, samt type af produkt (34).

## Organisationsopstilling

Der findes tre overordnede pæle i en software virksomhed, og disse er produktet selv, virksomheden og menneskene i den.

Ser man bort fra produktet for nu, dykker man ned i de strukturelle dele af virksomheden, og dertil lave en strategisk opsætning af team størrelser, ledelsesplan og diverse afdelinger og deres sammenkobling.



Billede 6. Organisations opsætning i de første faser af projekter.

Kravet til organisationen reflekteres som regel ved at gennemgå krav, projektets form, ressourcer, time to market og eksterne forhold.

I dette projekt reflekteres nogle af disse aspekter sig i retning af en agil process.

Dette skyldes de manglende ressourcer i form af personnel, økonomiske ressourcer og chancen for kravændringer.

I billede 6 ses et udgangspunkt for virksomhedens struktur, hvor fokus ikke indebærer en funktionel hierarkisk struktur, men mere baseret på adskilte organismer.

Den agile organisation (34) er kendt for sin struktur der minder om en organisme frem for en hierarkisk struktur, hvormed der menes at virksomhedens roller er sammenkoblet i stedet for opdelte faste roller.

Det kan i startperioden være svært for en virksomhed at overholde disse retningslinjer som vist på billede 6, men er ligesom Scrum en opsætning der skal udvikle sig.

Formålet med dette projekts struktur, indebærer at virksomhedens teams, ikke skal udsættes for støj af virksomhedens andre instanser. Dertil skal hvert team selvfølgelig ikke udelukkes fra kommunikation, da en repræsentant (team member x), er udvalgt til at deltage i møder, kommunikation med ledelse, Scrum master mfl. og derfor er ansvarsområder for dette medlem større.



Dette skal dog stadig være på minimums plan, da job funktionaliteten stadig er baseret på udvikling, og det er vigtigt at dette ikke udelades, sådan at forvirring og frustration af manglende arbejdsopgaver indenfor sit relevante felt ikke udelades.

De forskellige instanser af virksomheden er markeret med 1..\* multipliciteten for simplicitetens skyld, og grundet at størrelsen at virksomheden kan variere og derfor stadig bibe holde nuværende struktur.

Dermed sagt kan flere instanser til virksomheden stadig introduceres, hvis virksomheden forventes at nå en størrelse hvor kommunikationen og fælles målretning ikke længere kan styres af nuværende instanser (grundet kompleksiteten af produktet og virksomheden), men har behov for flere kommunikationsled.

Blandt andet er *Scrum* og *Scrums* (35) introduceret i nuværende fase, da projektet allerede nu forventes inden for få iterationer, at nå en størrelse hvor vidensdeling og målretning skal foretages på tværs af teams, uden at de relevante teams skal forstyrre hinanden (34).

Derudover vil Scrum masteren have den hovedsagelige korrespondance mellem ledelsen og udvikler teams.

Og dette er igen for at reducere støj. Der er dog den lille forskel, at en udvalgt repræsentant stadig har en hvis kommunikation til ledelsen, og dette skyldes at dele af udviklingsprocessen skal inkludere disse parter i en fælles kommunikation. Blandt andet *defect management*, som er beskrevet i *Defekt håndterings* afsnittet.

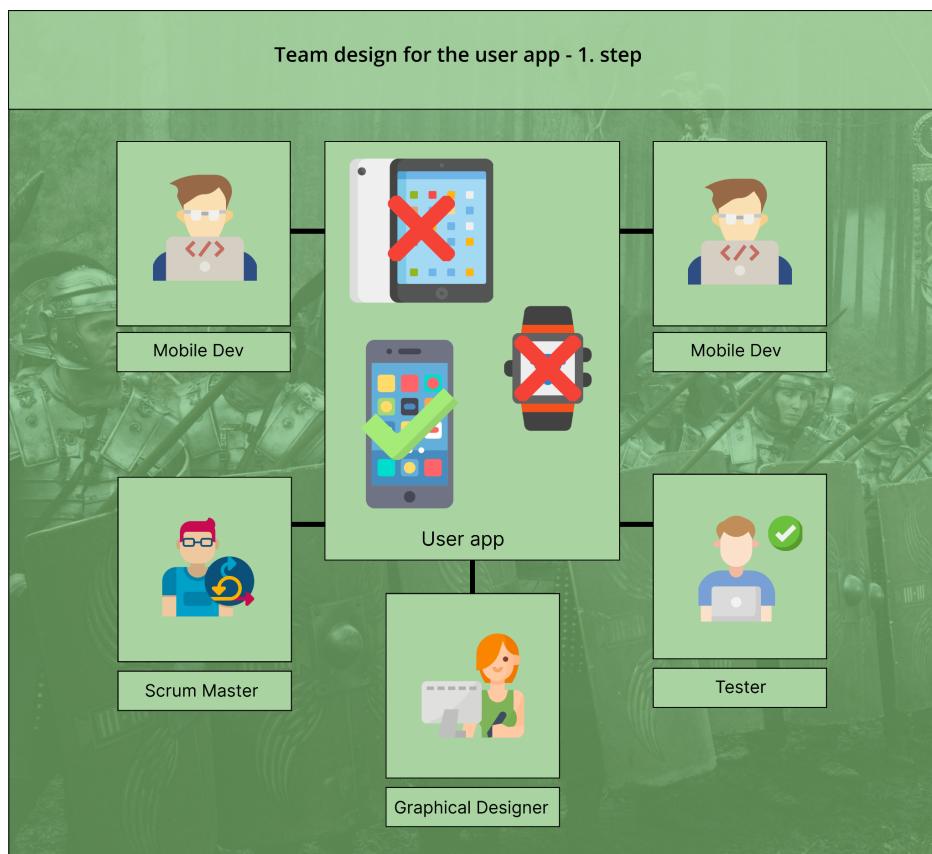
Med udgangspunkt i virksomhedens struktur fra dette afsnit, vil der nu blive dykket ned i de finere detaljer omkring diverse teams.

## Team design

Teamets størrelse er alpha/omega i dens overlevelse, og dens interne kommunikationsevne.

Er teamet for stort, kan den interne støj være produktions nedsættende, og arbejdsopgaverne kan varrirere i sådan en grad, at teamer ikke længere har samme retning.

Modsatte kan forekomme for små teams. Overbelastning, og manglende kommunikation kan føre til netop samme problemstilling, som før nævnt (34).



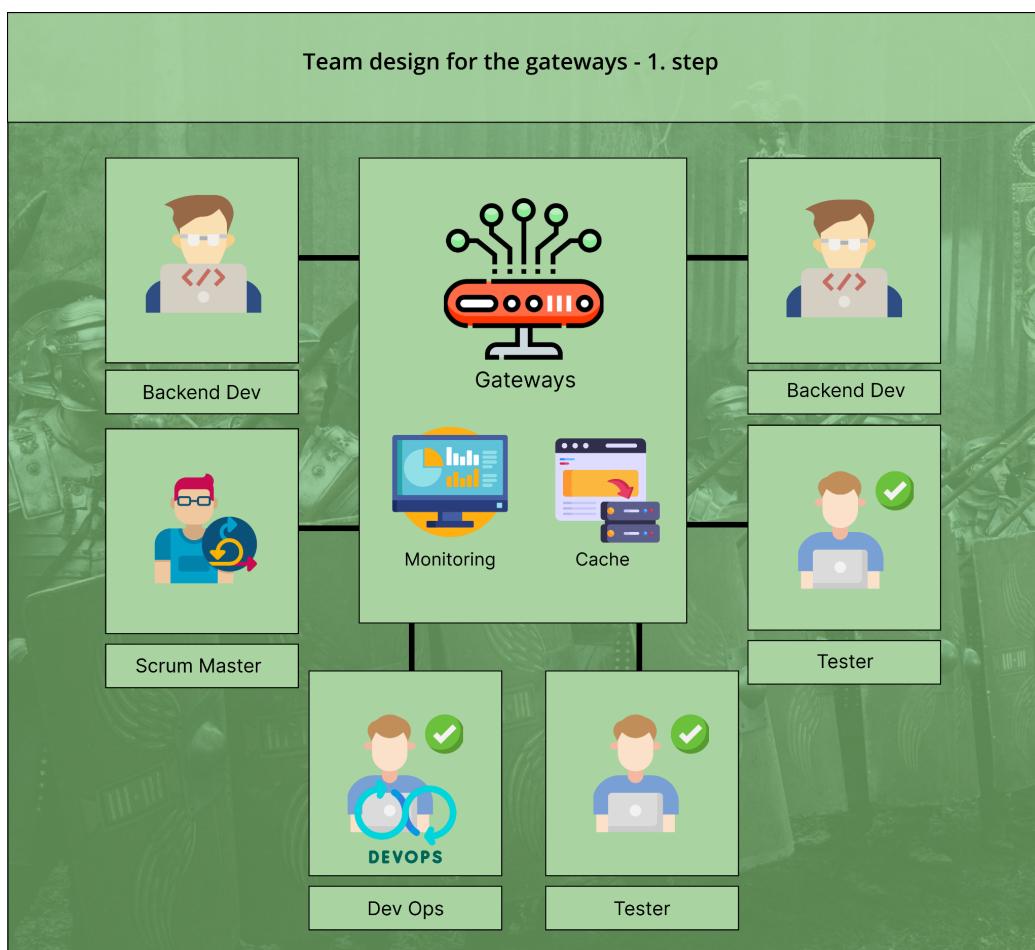
Billede 7. Team design der implementerer og vedligeholder klient applikationerne.

Med udgangspunkt i klient appen, kræves der forskellige platforme. Men i første fase af projektet er en mobilapplikation prioriteret.

Her skal sammensætning af kvalitetskrav, behov og kompleksitet sammen udgøre teamets roller, og størrelse.

Kravet til mobilapplikationen indebærer i nuværende fase en designer, som kan varetage et godt design til slutbrugeren (og dermed bidrager til *Usability* (17)). Dertil skal der være tilknyttet udviklere, der skal implementere den mobile platform, samt en tester til at vurdere og skrive tests til denne.

Og til sidst en tilknyttet Scrum Master, der skal varetage at processen foregår planmæssigt, og være frontfigur til virksomhedens interesser for at minimere støj hos teamet.



Billede 8. Team design der skal implementere og vedligeholde gateways.

De pågældende gateways i arkitekturen, skal anses som platform fokuseret. Dette vil sige at med udgangspunkt i billede 8, forekommer flere af disse teams løbene, når flere platforme/gateways introduceres til projektet.

I nuværende fase vil slutbrugeren kun drage mulighed af en mobil platform, hvorved et team der varetager den givne gateway indeholder testere, udviklere og en tilknyttet Scrum Master.



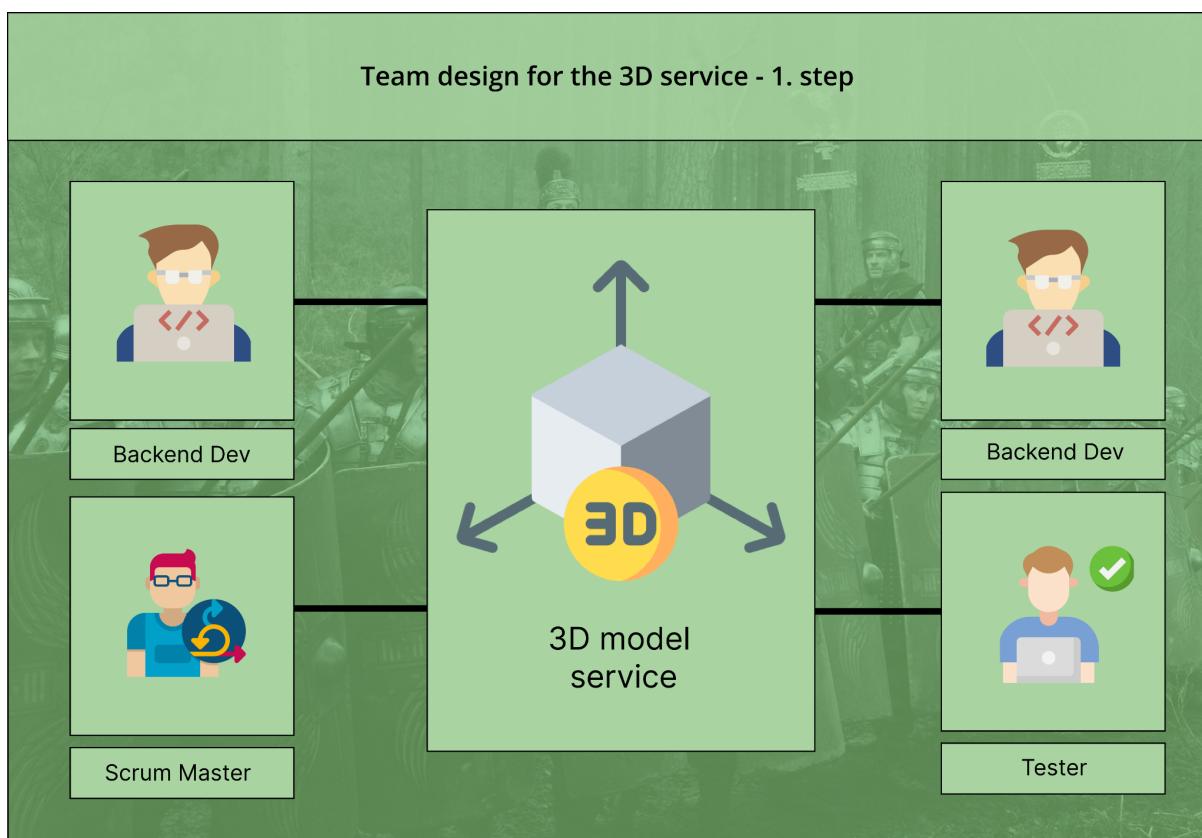
Billede 9. Team design der skal implementere og vedligeholde diverse services.

Billede 9 påviser der en samlet service opsætning. I starten af projektet vil disse services blive vedligeholdt af ét team.

Dette skal i en senere fase blive udvidet til at ét team per service. Dette skyldes at kompleksiteten af disse service bliver større.

Der har ikke været fokus på team størrelserne endnu, men der er taget udgangspunkt i de nødvendige kvalifikationer der skal til, for at kunne implementere de ønskede løsninger.

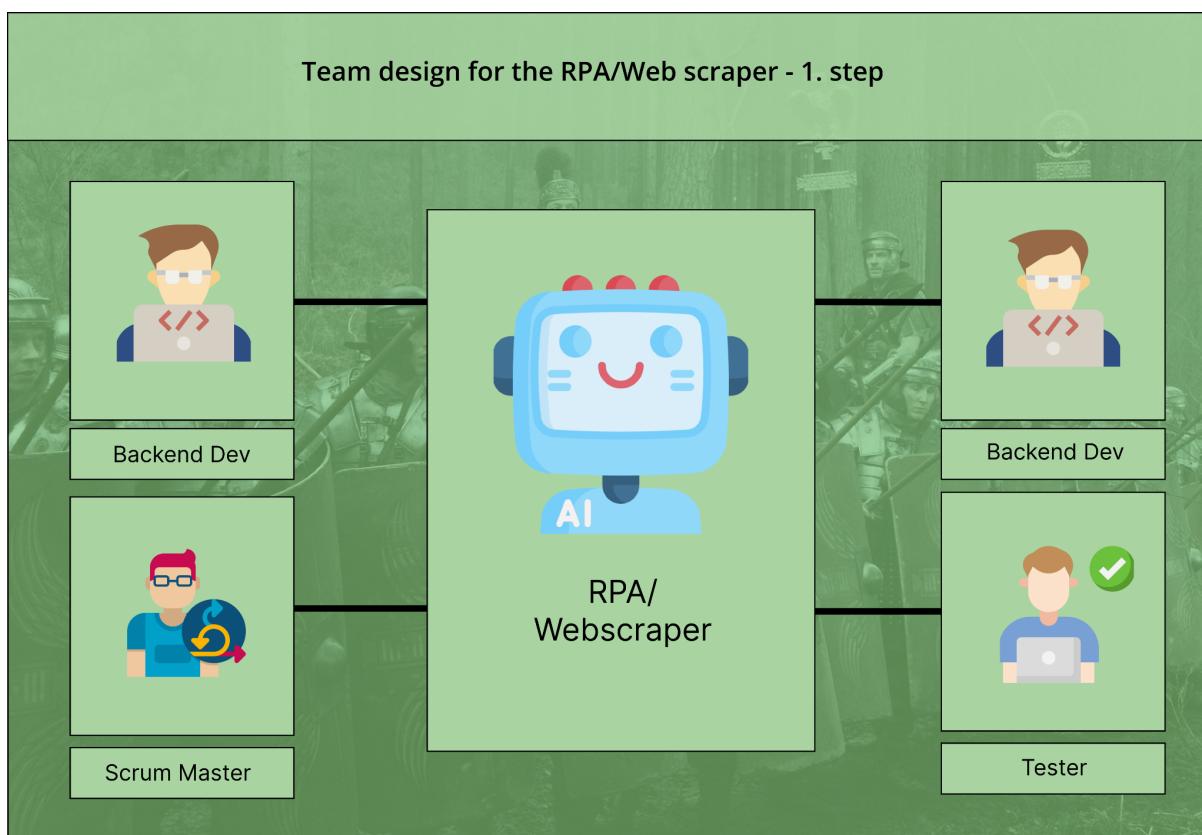
Der er dog taget forhold til at teamet ikke skal overskride 8 til 9 personer, da dette kan skabe problemerne nævnt i starten af dette afsnit.



Billede 10. 3D service team design.

Det samme princip gælder for den kommende 3D service, hvor slutbrugere skal kunne upload 3D modeller til augmented reality featuren.

Dette varetages af et mindre team, og kan anses som en planstruktur af denne service, og kan ændre sig når kravet bliver konkretiseret yderligere, når denne indtræder i udviklingsprocessen.



Billede 11. RPA/Web scraper team design.

Web scraperen er i starten ret begrænset i størrelse og kompleksitet, og teams størrelse er derfor spejlet i dette.

Dog er den ret kritisk i projektet, og skal varetages og vedligeholdes konstant igennem systemets levetid. Web scraper holdet vil som udgangspunkt også være utsat for kraftige ændringer i størrelse, i takt med systemets vækst.

To udviklere som minimum er godt at have i et team, da kommunikation og vidensdeling bliver nemmere.

Processen for de nuværende teams skal også defineres, for at opnå automatiserede og standardiserede processer og kvalitet af udviklingen.

## Projekt styrelse

I forlængelse af organisationsopstillingen og valg af projektmetodologi, er det nødvendigt at opsætte klare rammer for hvordan ledelsen vil gennemføre processerne for udviklingsfasen, og processen for før og efter at udviklingen foretages.

En applikation lever ikke kun som en statisk instans, men dynamisk udvikler sig over tid. Når applikationen udvikler sig, øger kompleksiteten og størrelsen af denne også. Derfor kræves der forskellige processer som applikationen skal gennemgå for netop at bibeholde kvaliteten.



En applikation er derfor ikke kun bundet til udviklingsprocessen, men også efterfølgende vedligeholdelse af både kodebase, men også dens data.

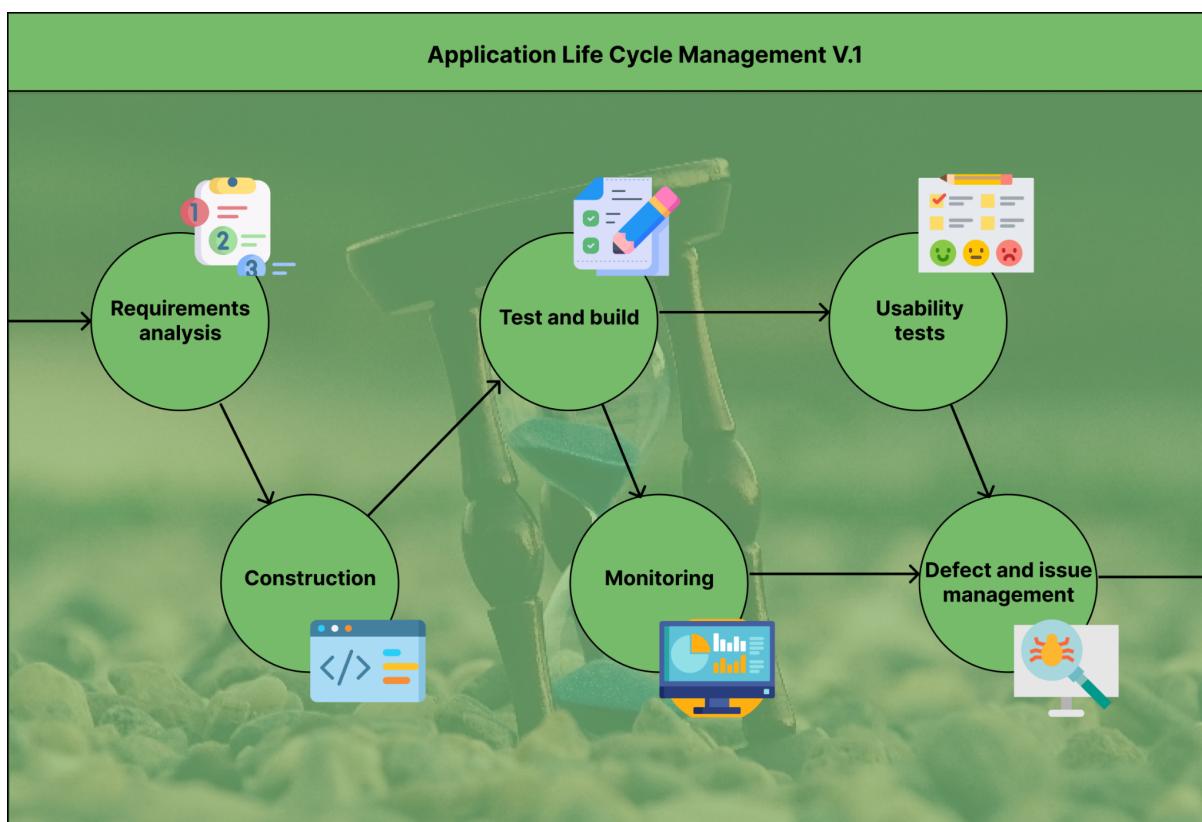
Designet af applikation kan også ændre sig hyppigt, og især hvis der foretages skræddersyet løsninger til kunder, eller kundesegmentet er bredt nok til at designet kræves i flere former (kravene kan ændre sig eftersom produktet bruges internationalt og geografisk).

Derfor er det vigtigt at kunne varetage disse ændringer og vedligeholdelser i standardiserede processor (der selvfølgelig kan forbedres over tid), men også automatiserede processer, hvor det er nødvendigt (34).

## Application Lifecycle Management

For at bedst kunne administrere systemets levetid på sigt, vil der med fordel kunne udvikles en ALM. Denne ALM vil have til formål at administrere systemet gennem dets levetid og de ændringer, der vil være umulig at undgå undervejs. For at understøtte denne administrering, er der adskillige punkter under et systems livstid, der vil have indflydelse på den. Noget af det der kan bygge fundamentet for hvorfor at Application Lifecycle Management er nødvendigt i længden er Lehmann's love. Lehmann's love beskriver, hvordan at udviklingen af software fortsætter selv efter at det er blevet deployed. Af alle de love der bliver beskrevet vil de mest relevante være den første og fjerde lov. Den første lov omhandler meget håndteringen af kontinuerlige ændringer i applikationen. Dette vil være en relevant bekymring for dette projekt, da det forsøger at udfylde et behov, hvor der er alternativer. Dette er også et område hvor, at behovene kan ændre sig over tid. Dette kan fx være at nogle af de institutioner, begynder at tilbyde et andet form for medie, som ikke er kendt på dette tidspunkt. Et eksempel på dette medie bliver blandt andet beskrevet i afsnittet omkring navigering og augmented reality. Den fjerde lov der også er meget relevant omhandler loven omkring kontinuerlig vækst. Dette projekt har til mål at gro i omfang over den kommende årrække, hvilket medfører at det bliver nødvendigt at foretage handlinger, for at imødekomme denne vækst uden det påvirker funktionalitet for meget (36).

Udover relevansen af emnet af Lehmann's love, er der flere discipliner gennem systemets levetid; det bliver nødvendigt at tage hånd om. Denne process kan blandt andet ses på følgende billede.



Billede 12. Application Lifecycle Management cyklus.

Når det kommer til diverse discipliner indenfor ALM, vil det være nødvendigt at dække forskellige dele af processen, der er efter deployment. En del af dette inkludere blandt andet *change management* og *traceability*. Når det kommer udviklingen af software er det vigtigt, at kunne spore ændringer og features, uanset hvilke stadie der er tale om. Til kodeændringer mm. vil der blandt bruges Git, hvor det er muligt at tilføje et story Id til commits. Dette vil ikke bare medføre bedre *change management*, men også sikre bedre *traceability*. Dette vil blandt andet medføre, at det er muligt at tracke en ændring fra udformningen af story, til den er testet og deployet. Et eksempel på dette kunne fx være udvidelsen af søge funktionaliteten (Search Service). Dette kan fx være håndteringen af af søgekriterier, hvor at der kunne være snakke om en brugerdefineret års række interval. Et andet eksempel på dette kan også være formningen af en ny platform, der ikke kendes til endnu. Dette ville resultere i en større ændring, hvor at det ville være påkrævet at tilføje instanser af services baseret på dens kapacitet.

Et større område i dette system vil også være håndteringen af versionskontrol. Håndteringen af dette samt fornævnte discipliner vil være Git. Grunden til at der sættes vægt på at versions kontrollen i dette system, er brugen af forskellige versioner baseret på platform. Et eksempel på dette ville være når at Search Servicen har forskellige instanser pr platform. Instansen der håndterer mobil gatewayen kan have en tidligere version af Search Servicen der er til eksempelvis en tablet platform. Denne adskilning der sker mellem instanser af samme service, muliggør også bedre *fault isolation*, som



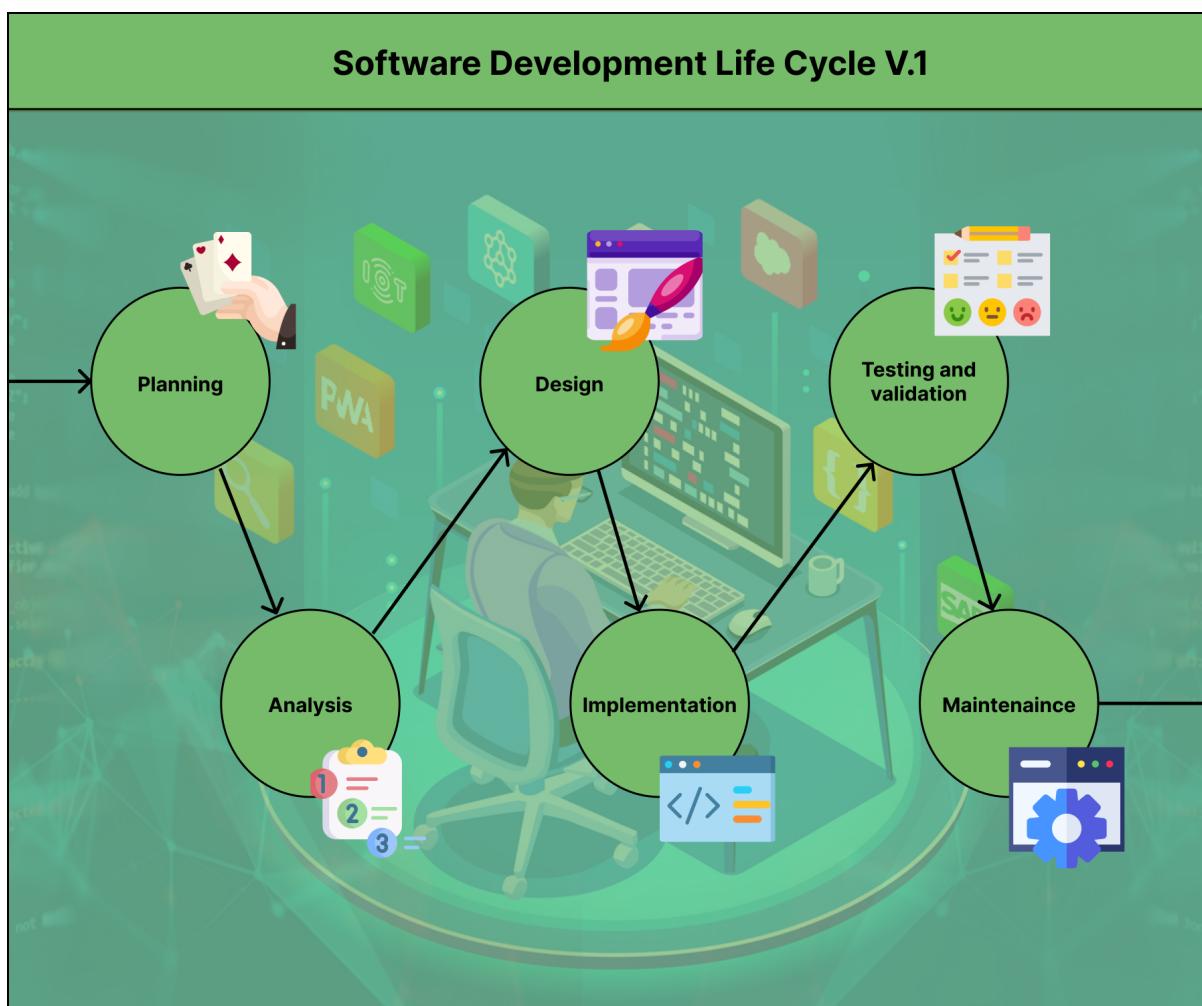
blev præsenteret tidligere. Den tydeliggør også swimlanes, hvilket muliggøre at fx en platforms Search Service kan opdateres uden at påvirke de andre platforme.

Som systemet gror i størrelse og kompleksitet, vil det også give mening at se mere ind i *Continuous Integration* og *Deployment*. Når det kommer til det at kunne levere et produkt, og kunne hjælpe med ikke bare testning, men reducere mængden af fejl og sikre bedre kvalitet (36), og skubbe ændringer og tilføjelser konstant. I de tilfælde så er automatisering i form af *Continuous Integration* og *Deployment* en vigtig faktor. Til at håndtere automatiseringen skal der anvendes et værktøj. Til dette vil der blive anvendt Jenkins som værktøj. Dette skyldes at den tilbyder et open source valg, der er nemt at opsætte, samt tilbyder mulighed for både On premise og cloud løsning. Denne håndtering af automatisering vil ikke nødvendigvis give mening i første version, som en prioritet. Det skal dog noteres at siden skalering til Europa, er et af de strategiske mål, vil dette blive nødvendigt inden for en nærmere fremtid (37).

Som det kan ses beskrevet både i målene for dette system og dets omfang samt i det dedikerede afsnit omkring monitorering og KPI'er, så er monitorering en nødvendighed for at kunne sikre bedre kvalitet i systemet. Ikke blot ved at sikre at de mål der er blevet stillet overholdes, men også som en del af den bredere arkitektur med Microservices mm. Til dette vil der anvendes Nagios (38)).

## Software Development LifeCycle Management

Som en del af det fornævnte projekt styres afsnit, så er der en anden del af Application Lifecycle Management, nemlig software development lifecycle. Modsat ALM afsnittet, så vil denne fokusere mere på processen fra planlægning, implementering og testning og senere vedligeholdelse af produktet. Den følgende illustration viser forløbet af denne proces.



Billede 13. Software Development Life Cycle cyklus.

Hvis man følger fremgangen i denne illustration, vil den første del af processen være planlægningen. Gennem hele udviklingsprocessen, vil der anvendes Jira, som er et projektstyringsværktøj, der blandt andet kommer med muligheden for at bruge planning poker til hjælp med estimeringen. Denne proces kan hjælpe med planlægningen af fremtidige sprints og releases, samt prioritering af User Stories. Til at hjælpe med at designe diagrammer, og visualisere disse designs, bruges Figma. Når det kommer til selve implementationen anvendes der blandt andet Visual Studio, Visual Studio Code og Git. Git blev tidligere præsenteret til brug af versionsstyring samt *Change Management*, hvorimod at Visual Studio anvendes som IDE. Testning og validering blev gennemgået i ALM, hvor at flere forskellige discipliner anvendes, såsom automatisering. Til sidst er der selve vedligeholdelsen, og dette kræver blandt andet monitorering for at bedre kunne vedligeholde og komme på forkant med mulige problemer. Som nævnt tidligere vil Nagios anvendes her (39).



## RASCI og DevOps

I forlængelse af ALM og SDLC, er det vigtigt at pointere de ansvarsområder, de enkelte medarbejdere i virksomheden besider.

Gøres dette ikke, kan forvirring og konflikter hurtigt opstå undervejs i processen, og skyld kan hurtigt pålægges hos andre.

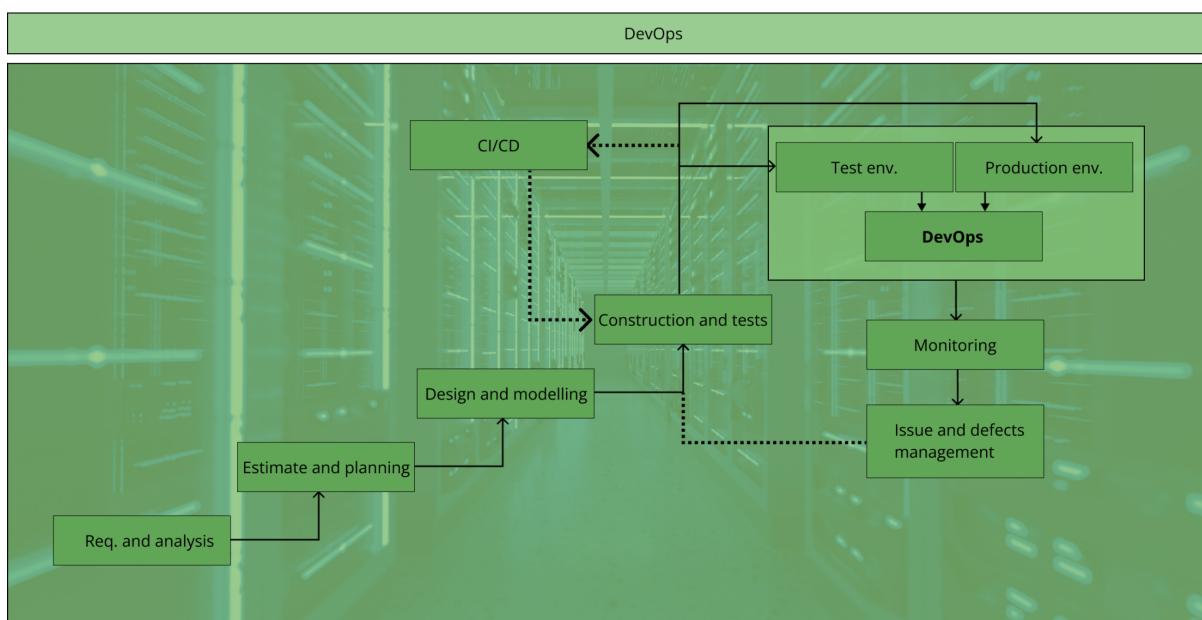
Derudover vil processer også virke mest effektivt når standarder og automatisering er introduceret til de redundante opgaver, der optræder.

Her kan RASCI modellen spille en vigtig rolle.

Eftersom virksomheden består af mange kompetencer og afdelinger, vil de underliggende processer i ALM og SDLC tillægges nogle enkelte medarbejdere.

Og eftersom at projektet udføres agilt, og i organismer, kan disse processer udføres parallelt.

Derfor er der også brug for et DevOps team, der kan opstille og håndtere de testmiljøer, samt produktionsmiljøer, sådan at når hvert team ikke står med dette ansvar og har derfor kun fokus på produktudvikling.



Billede 14. Sammenkobling mellem udvikling og produktion ved hjælp af devops team.

Med udgangspunkt i billede 14, som er en forenklet version af ALM, vil devOps teamet binde udviklingsprocessen sammen med produktion og test miljøerne.

Deres ansvar ligger netop i at vedligeholde og monitorere disse. Dette kan også anses som en del af *Continuous Integration* og *Continuous Deployment* delen af ALM processen. Skulle der opstå defekter eller problemer i monitoreringen, vil devOps teamet videregive disse til udviklerne samt testerne, som vil inddrage det i deres sprint process efter risikovurdering. Sidstnævnte uddybes i *Defekt håndterings* afsnittet i denne rapport.



RASCI							
Process	Developers	Database	Testers	PO's	SM's	OPS'	UX/UI
Requirement and analysis				A/R			
Planning and estimate	R	S	S	I/S	A/C/S		I
Design and modelling (UI/UX)				A/I			R
Design and modelling (system design)	A/R	S	S		S/I		
Construction	R	S	S	A	S/I		S
Test and validation	S	S	A/R		S/I	I	S
Build and test env.	I	I	I	I		A/R	
Issue and defect mgt.	S	S	S	I	C/I	A/R	
Release mgt.						A/R	
Production monitoring						A/R	
Maintenance	R	S	I/S	A	S		C/S
Verification	I	I	R	A/I/S	S		C/S

Billede 15. RASCI model.

Efter konkretisering af RASCI modellen, vil der nu tydeligt være ansvarsfordeling, og fælles forståelse for den overordnede ALM process.

RASCI modellen kan skifte karakter løbende, men det er vigtigt ikke at ændre for meget ansvar, sådan at overbebyrdning af arbejdsopgaver sker, og fokus på målområder bliver mere udvandet.

I design og modellering af systemet, ligger ansvaret hos udviklerne. Dette skyldes at systemarkitekt rollen er underlagt udviklerne (40).

## Gateway, Micro Services og Messaging

Gateway mønstret tilbyder en separation mellem diverse klient platforme, hvorpå at vedligeholdelse og features adskilles i mindre dele.

Dette gør skalering nemmere, og mere effektivt for virksomhedens udviklere (*Interface Segregation principle* fra SOLID principperne).

Dertil vil en opdeling af diverse gateways betyde at muligheden for *single point of failure* (41) mindskes.



For at opnå mest mulig tilgængelighed, skal der implementeres et asynkront message bus system bagved diverse gateways.

Dette formår at fjerne et potentielt performance problem med den synkrone HTTP protokol, hvor at den gældende gateway, skal vente på respons for at frigive ressource til næste forespørgsel.

Dette problem fjerner en message bus, hvor at princippet *fire and forget* (42) ikke har dette problem.

Der findes endnu en tilføjelse til gateway mønstret, hvor at komplekse forespørgsler, der som regel skal aggregeres til en samlet respons tilbage til klienten håndteres separat fra selve gatewayen. Denne implementation er især populær indenfor *Micro Service* mønstret. Her er der tale om en *Aggregator* (43), som netop har til formål at fjerne aggregering af adskillige forespørgsler fra selve gatewayen.

Aggregatoren skal samle den nødvendige data fra diverse forespørgsler, og sende resultatet til den relevante gateway.

## Microservices implementation

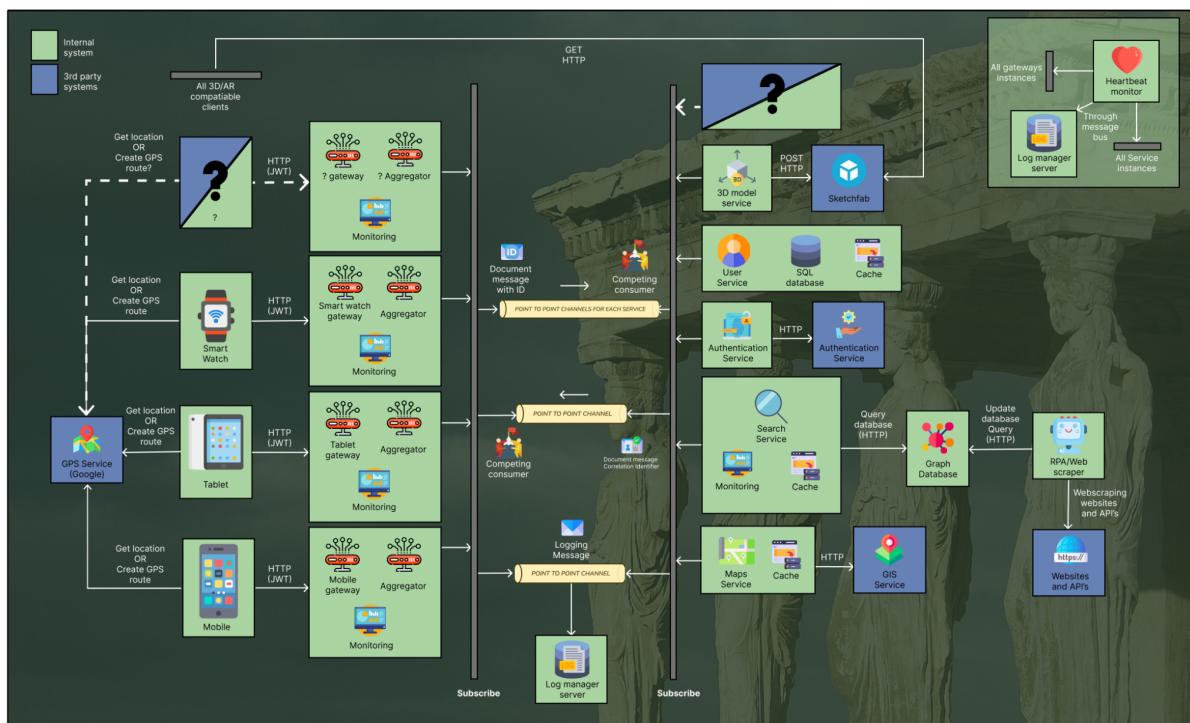
Når det kommer til selve API'en, eller den applikation der tilbyder de ressourcer produktet skal indeholde, er det vigtigt at vælge sit mønster og arkitektur med omhu. Konsekvensen af at vælge et mønster eller arkitektur der ikke passer med systemets fremtidige skalering, kan betyde at implementeringer bliver sværere, mere komplekse og svære at bryde fra hinanden igen.

Skaleringen vil derfor lide, og dermed også potentialet for dette produkt, og virksomheden.

Der findes adskillige arkitekture og mønstre, og kan være svære at navigere, og udvælge.

Eftersom systemet skal udvikle sig over tid, og kravene forventes at ændre sig, vil Micro Service arkitekturen være et godt valg.

Udover dette vil skaleringen på Y og Z aksen også være i fokus, og når produktet skal spredes til Europa, er dette en fordel.



Billede 16. Micro Service arkitektur.

Hver Service kan være implementeret af et hvilket som helst eksisterende programmeringssprog, følge hver sine egne relevante design mønstre, lagre data i cache og gøre brug af et bestemt udvalgt database paradigme.

Dermed er Micro Service arkitekturen teknologi agnostisk, og derudover polyglot persistence (44) også en attraktiv mulighed, grundet fornævnte.

Fornævnte er derfor perfekte til agile udviklingsprocesser, tilføjelser af nye services og ikke mindst tilgængeligheden af systemet kan sikres i lang højere grad end en monolith applikation.

Dertil kan *fault isolation* principippet også være gældende, da systemets individuelle services kan fejle uden at påvirke resten af systemet.

Dette princip kan også gælde for en monolith applikation, men dette vil have langt større effekt på systemets tilgængelighed, og påvirke en samlet instans, fremfor en separeret service.

Efter etableringen af vigtigheden af microservices, er det nødvendigt at se på hvordan kommunikationen vil blive håndteret i systemet.



## Messaging

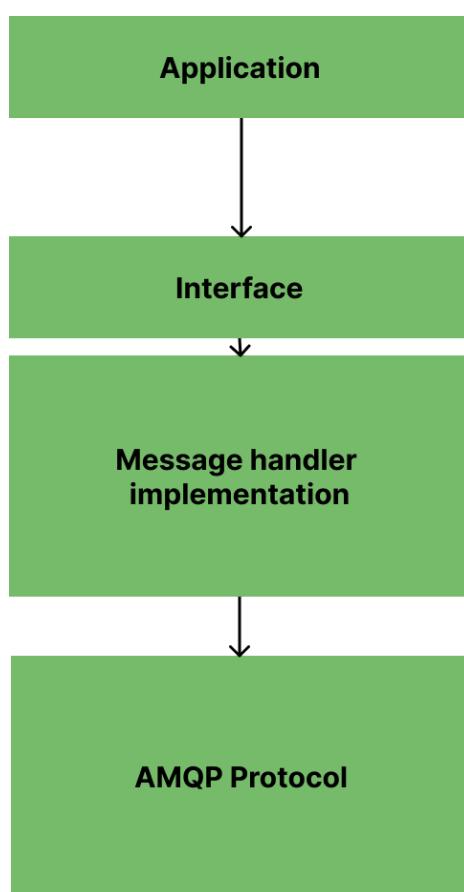
Kommunikationen til services og mellem hver service skal etableres. Der findes selvfølgelig også forskellige muligheder, hvor blandt andet HTTP protokollen kan anvendes. Ellers er der som fornævnt den asynkrone AMQP (45) (Advanced Message Queue Protocol) en mulighed. Den eksisterer også webhooks, SOAP (RPC) eller delte lagringer mfl. Og nogle af disse kan implementeres sammen (46).

Hvad der er vigtigst i denne sammenhæng er produktets fremtidige skalering, og hvad der kan forventes at ændre sig.

Her er det blandt andet vigtigt at overveje hvilke akser produktet skal fokusere på.

Eftersom at ønsket om tilgængelighed, performance, skalerbarhed og vedligeholdelse, vil en AMQP være et godt valg.

Dog har AMQP en kompleksitet der afviger fra sine konkurrenter, nemlig dens asynkrone opførsel.



Billede 17. Typisk implementation af message bus værktøj, ved brug af AMQP protokollen.

Ønsker man en forespørgsel og svar implementation, er man nødt til at identificere de enkelte beskeder, og dirigere dem til de gateways og klienter der har efterspurgt en ressource fra systemet.



Der er værktøjer der er bygget ovenpå AMQP protokollen, og kan hjælpe med at implementere og visualisere message flow i systemet (blandt andet RabbitMQ, Azure Storage Queue og Apache Kafka mfl (47)), og hjælpe med at løse denne kompleksitet. Disse værktøjer har implementeret et interface ovenpå AMQP protokollen, og dermed udnytter protokollen til at videresende beskeder via simple eller komplekse implementeringer (48).

Produktet bruger messaging integration mellem gateways og services. Dette skaber en asynkron sammenkobling, hvor de enkelte gateways ikke skal afvente et synkron respons.

Implementationen har taget inspiration fra Microsofts løsning for Micro Service mønstret (49).

Eftersom man ikke kan forudse hvor mange forespørgsler der vil ankomme til en gateway eller service, kan et message system sørge for at den gældende applikation ikke skal holde styr på transaktioner eller holde stadie på data. Dette kaldes også *fire and forget*.

I dette projekt kan ændringer i form af nye services, krav til antal forespørgsler, eller selve kompleksiteten af forespørgsler, være et scenarie der skal tages hånd om.

Skulle en applikation fejle, vil beskeden der er sendt afsted, stadig være i live, og AMQP protokollen sørger også for at hvis en kanal skulle fejle, at man kan genoprette den givne besked. Dette udgør igen et robust system.

Messaging udgør kun et system integrationsmønster, men yderligere mønstre kan overvejes der har samme fokus, nemlig på systemarkitektur, eller applikations mønstre. Der er igen taget inspiration fra Microsofts forslag til sådanne i en Micro Service arkitektur.

## Yderligere udvalgte design mønstre

Der er blevet diskuteret arkitektur, og mønstre med fokus på systemet, men selve opførsel og opbygning af systemets interne elementer mangler. Her menes der på applikationsniveau.

Der findes en række design mønstre, og nogle kan findes i bogen *Design Patterns* (50). Design mønstre har forskellige formål, men i sidste ende med samme princip, nemlig løse gængse design problemer i software.

Forretningslogik der duplikeres, har større risiko for at fejle, eller misset ved ændringer (hvis der skal foretages ændringer af forretningslogikken, kan dette blive overset i kodebasen, hvis implementeret flere gange), og kompleksiteten samt vedligeholdelsen af kodebasen, inkrementeres derfor.



At kunne udvælge design mønstre hvor det er relevant, kan mindske fornævnte problem.

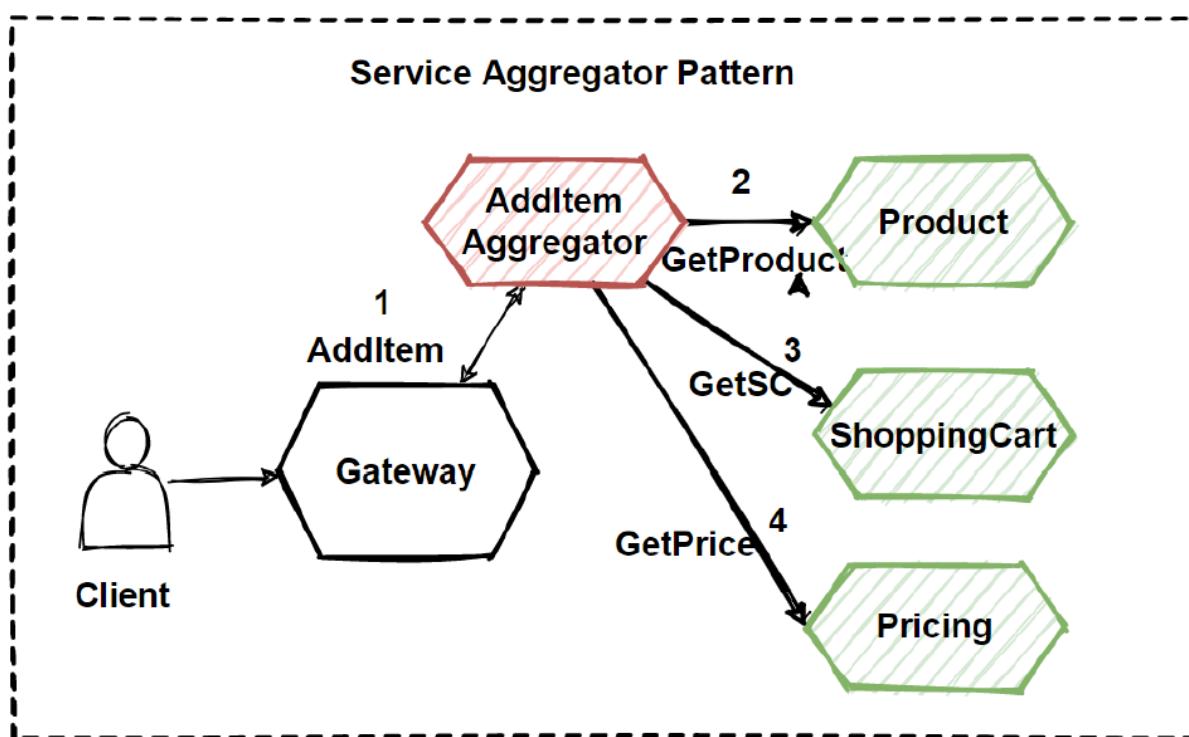
## Aggregator

Aggregator mønstret (43), skal aflaste de foreliggende gateways der i en *Micro Service*, når flere services skal kaldes, og aggregeres til et samlet respons til klienten.

I nuværende fase, hvor det kun er enkelte services der skal kaldes, vil dette mønster være unødvendig.

Dog skal mønstret medtages, når projektet begynder at implementere flere services, såsom et User Service, AR service og eventuelle service til uploading af 3D modeller. I sådanne tilfælde skal systemet bruge flere informationer omkring brugeren, samt eventuelle kalde flere 3. parts services i baggrunden, der kan være medvirkende til at performance ned sætte de pågældende gateways.

Dette vil aggregator mønstret hjælpe med.



Billede 18. Service Aggregator mønstret (43).

Billede 18 viser et eksempel på implementeringen af et scenarie hvorpå aggregator mønstret henter data fra tre services, og samler det til ét respons til gatewayen.

Dette er især gældende i en REST eller SOAP integration, men kan også være gældende i en RPC integration med asynkront message queue, da flere messages skal aggregeres til ét respons.



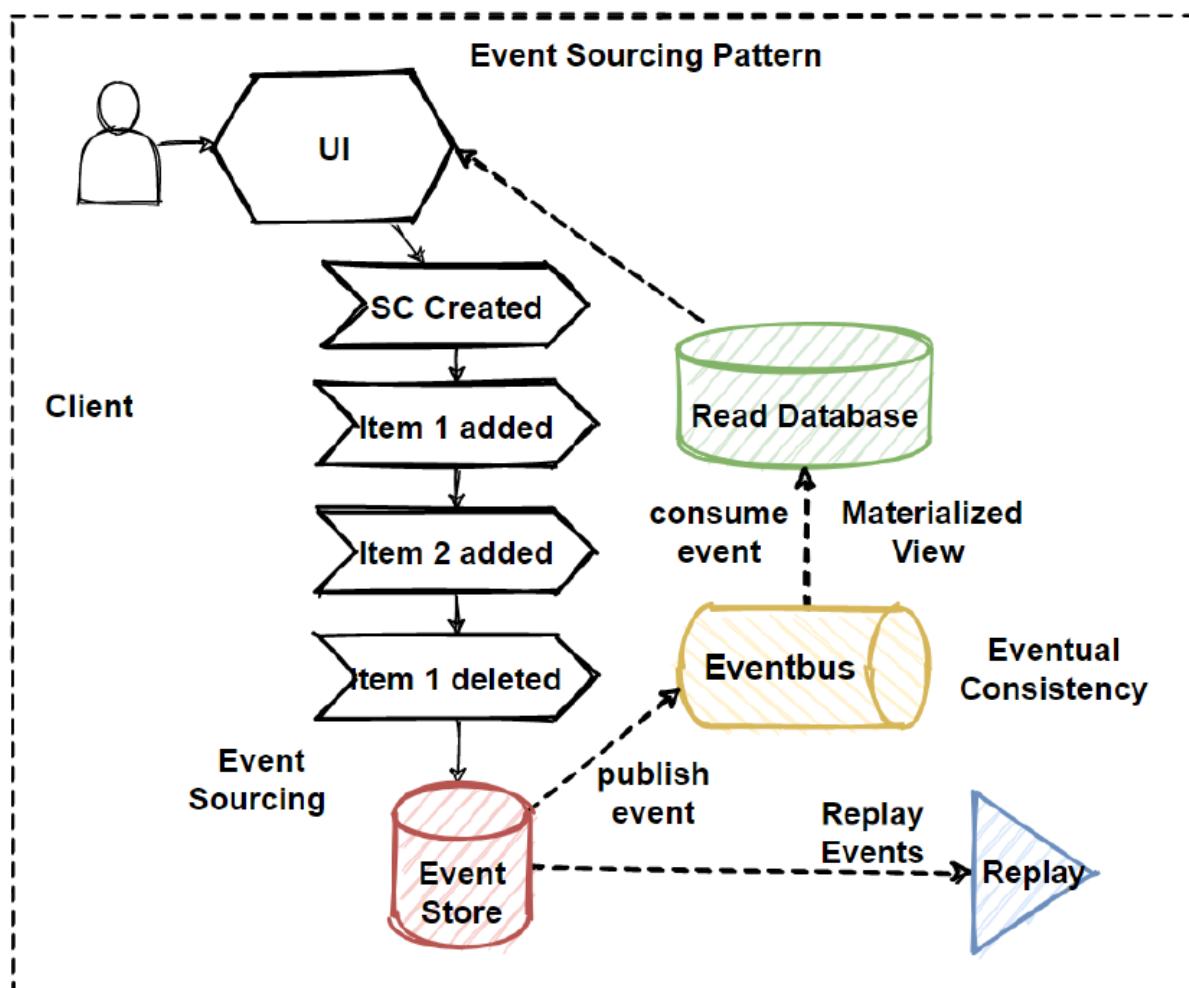
Duplikering af gateways og aggregatoren kan også ske uafhængigt af hinanden, og aggregatoren kan også forekomme i forskellige instanser, hvis systemet får flere services.

Ved at implementere en aggregator vil TTM (time to market) forøges, da kompleksiteten vil blive større, men kan være en fordel for performance og availability for slutbrugeren. *Fault isolation* vil også blive mere konkret med denne løsning, at gatewayen ikke lukkes, men kun en aggregator der i et orkestrering system (se afsnittet containerisering) hurtigt kunne genstartes, og ikke påvirke slutbrugeren.

## Event Sourcing

*Event sourcing* (51) mønstret vil spille en afgørende rolle, hvis ændringer i systemets tilstand foretages. Dette kunne være til databaserne, eller 3. parts systemerne. Her kunne pushede 3D modeller til *Sketchfab* være muligt at kunne opdatere systemet, samt opdatere slutbrugerens applikation automatisk.

Ændringer i lokationsdata, eller tilføjede data til *Search Service*, vil give samme resultat.



Billede 19. Event Sourcing mønstret visualiseret (51).



Med udgangspunkt i billede 19, vil UI (slutbruger applikationen) læse direkte i den bagvedliggende database.

Dette er ikke tilfældet i dette projekts arkitektur, men resultatet er det samme.

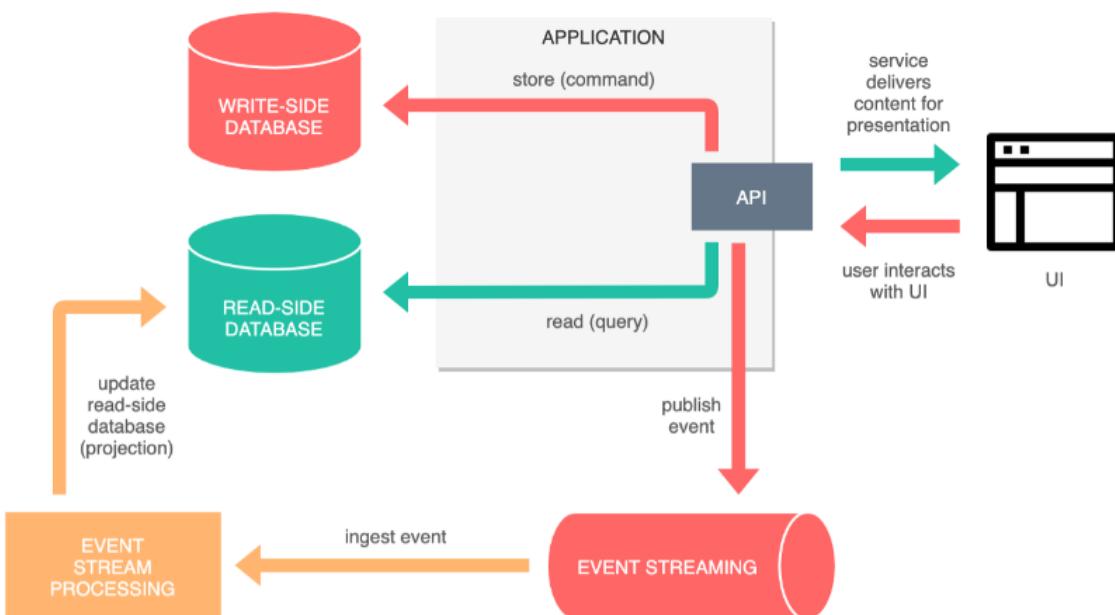
Det opdaterede data sendes i stedet gennem en gateway.

## CQRS

For at opnå yderligere asynkronitet i systemet, vil et applikations mønster som CQRS være relevant.

Læse og skrivning gennem systemet, kan tage forskellige veje, og indeholde forskellige datamængder.

For at undgå at irrelevant data bliver sendt frem og tilbage, og bruge systemets ressourcer vil sådan en implementering være behjælpelig.



Billede 20. CQRS eksempel, med implementeret event sourcing (52).

Derudover hjælper CQRS også med implementering af nye features, da tilføjelser af enten *command* eller *query* model, kun skal implementeres, og der skal ikke foretages refaktorering af eksisterende modeller (52).

Dette betyder også at systemet bliver mere robust, og TTM formindskes.

I forhold til billede 20 er gateways undladt, men principippet er det samme. CQRS fungerer sammen med *Event Sourcing* system mønstret, idet at segrerering af læsning og *command* forespørgsler matcher (52).



## Containerisering og orkestrering

For at mindske kompleksiteten i administrationen af *containers* (53) i uptime, restarts og opdatering, kan *orchestration* (54) systemer hjælpe.

Men først skal der identificeres hvilke problemer der kan opstå i projektet, der indebærer valget.

I en *Cloud Native* udviklingsprocess, skal platformen, som en applikation bliver hostet i, være intetsigende. Dette menes der med at udviklingsprocessen er afhængig af at kodebasen skubbes og testes løbende i det miljø applikationen er hostet i.

For at konkretisere yderligere, kan eksemplet være portabiliteten af en applikation fra udviklerens eget lokale miljø til produktionsmiljøet, hvor eksekvering variere fra miljø til miljø baseret på systemressourcer og OS(Operativ System) versioner.

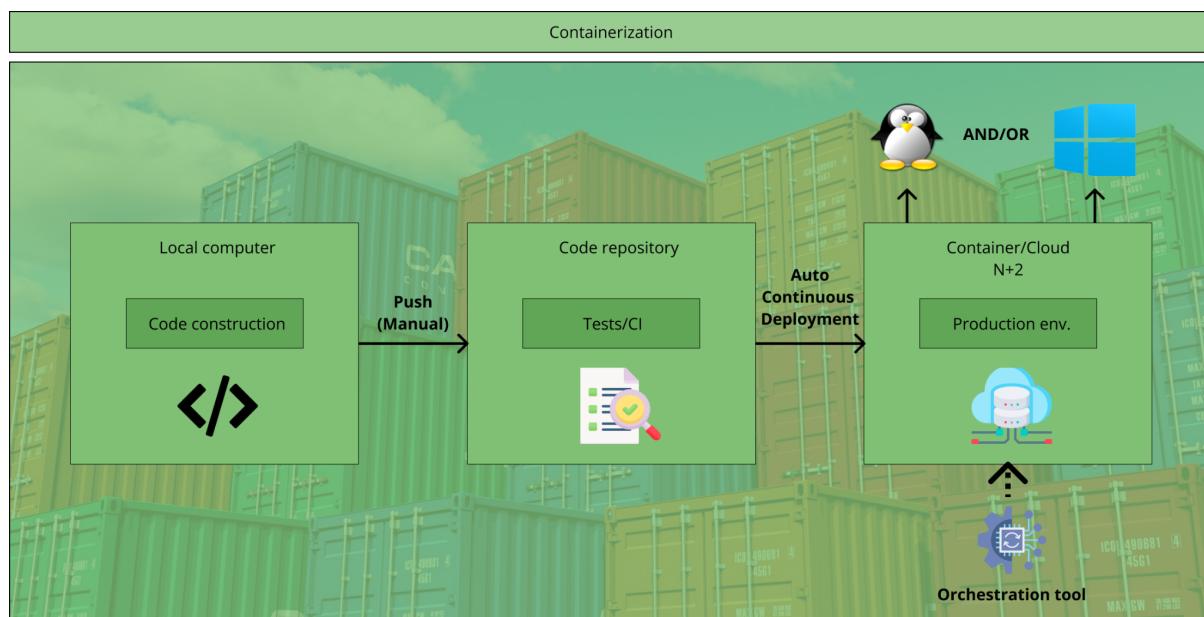
Dette løser containerisering af kodebasen.

I samarbejde med *Continuous Integration* og *Continuous Deployment*, fungerer et *container* miljø i et stærkt samarbejde.

Udviklerne skal ikke bekymrer sig om portabilitet. Administrationen af applikationens levetid afhjælpes også, da værktøjer såsom *Docker* og *Kubernetes*, igennem deres CLI (Command Line Interface), simplificere denne process.

Skulle applikationen ikke svare, eller skulle den fejle vil orkestreringens værktøjer genstarte, og sørge for at applikationen eksekveres igen automatisk.

Dette betyder også at små releases, og builds og failures kan foretages mere smertefrit, og man netop kan automatisere processen af disse.



Billede 21. Workflow for kode produktion til produktionsmiljø i container miljø, som er en vilkårlig platform (Simplificeret) (55).



Ved at redegøre og opklare disse vil sammenkoblingen af udviklingsmiljøet til produktionsmiljøet være automatiseret, og kompetencer og ressourcer der skal bruges på administration formindskes væsentligt.

Derfor vil man i en ALM process og ALM 2.0 (36) process være langt mere fleksibel, og

## Navigation og Augmented Reality (AR)

I et projekt af denne størrelse, og med diverse integrationer, udnyttes der adskillige eksisterende teknologier, og ikke alle har samme relevans eller prioritet i denne opgave. Dermed sagt er der stadig nogle udvalgte der er interessante at undersøge nærmere, og hvordan indgå i projektet for at opnå organisationens mål.

### GPS

Et GPS system kommer til at spille en afgørende rolle i produktet. Dette skal guide brugeren til de fortidsminder, de ønsker at besøge.

I samtal med Kim Callesen var ønsket om dirigering, og vejvisning til fortidsminder et stort plus, da disse kunne være svære at finde, selvom man kendte deres position.

Derudover vil en foreslægt rute til diverse fortidsminder også være et stort plus, for de brugere der ønsker at besøge flere fortidsminder på en rute (56).

Flere GIS systemer(Google maps, bing maps og TomTom mfl.) kommer med understøttelse til navigering.

### GIS og kort data

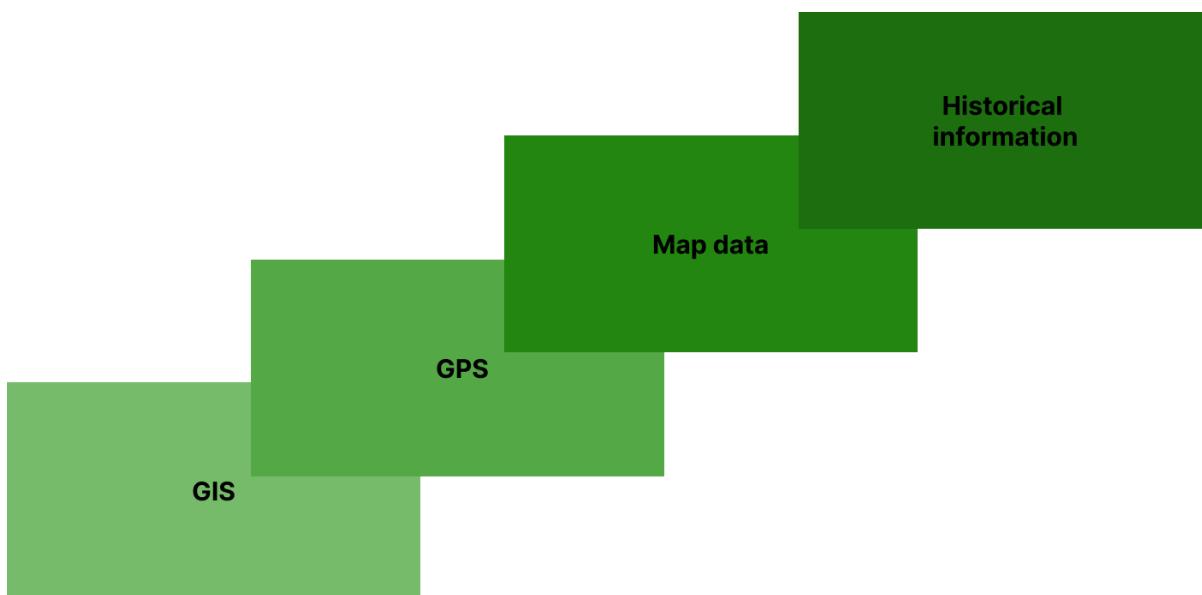
GIS og især kortdata kan sammen med GPS kunne tilbyde en tourguide. Hvor brugeren guides rundt i landskabet, og markere de områder brugeren ønsker at se.

Med et GIS system vil produktet også kunne holde historiske data over et område eller fortidsminde, og give brugeren en større oplevelse heraf (57).

En feature GIS også kan tilbyde er 3D visning af landskabet, samt hvordan landskabet har set ud før i tiden. Derudover kan flere forskellige kortlag skiftes mellem, for at få forskellige informationer på et givent kort frem.

Dette vil man i en senere version af produktet kunne udnytte, og dermed give mere information vedrørende det gældende fortidsminde, og dens sammenhæng med landskabet gennem tiden, og i sammenhæng med andre fortidsminder.

Derudover vil tour guider også kunne give en mere detaljeret, og visuel forklaring af landskabet og fortidsmindets forbindelse (56).



Figur 22. Kortet der vises på klientens applikation kan opdeles i 4 lag.  
Hvert lag tilbyder forskellige features, og information til brugeren.

GIS data kan i forhold til GPS, dermed give langt mere information til brugeren, men også den professionelle i sin formidling.

## AR teknologi og 3D modeller

Augmented Reality er ifølge Kim Callesen også et stort plus hvis dette kan realiseres. Denne del af produktet bliver den mest krævende, da der skal bruges 3D modeller af relevante fortidsminder (1).

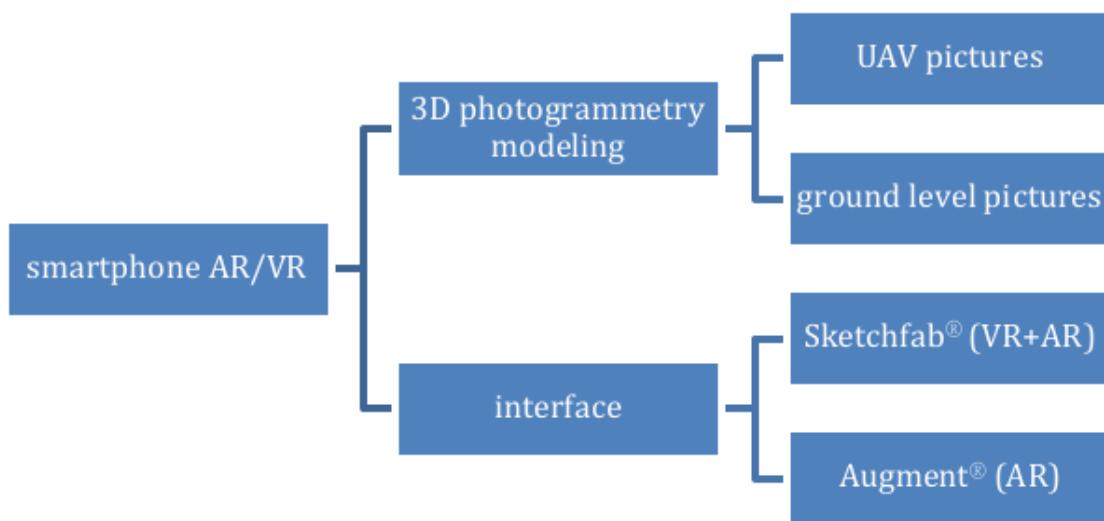
Under en brainstorm runde udsprang ideen om fotogrammetri (18), hvorpå billede taget af et fortidsminde kan rekonstrueres visuelt.

Dog opstår der et andet problem. Dette betyder at fortidsmindet allerede skal være nogenlunde intakt.

Der skal foretages en anden løsning i de tilfælde hvor dele af fortidsmindet er delvist forsvundet og ønskes at vises.

Dette kan løses ved at arkæologer har mulighed for at tilkoble platformen, og dele deres repræsentation af deres 3D model af et fortidsminde de har studeret.

En blanding af disse to løsninger vil i første omgang være en mulighed for at skabe de første 3D modeller til AR visning.



Billede 23. Eksempel for AR (Augmented Reality) implementation til smarttelefoner. Her vises der integration mellem 3D modeller, og eventuelle programmer til visning (58).

Næste problem i denne sammenhæng, er historisk præcision. Her vil der skulle være en tæt kontakt med de relevante arkæologer, og eventuelle andre eksperter, som kan teste modellens placering, kvalitet og realisme (58).

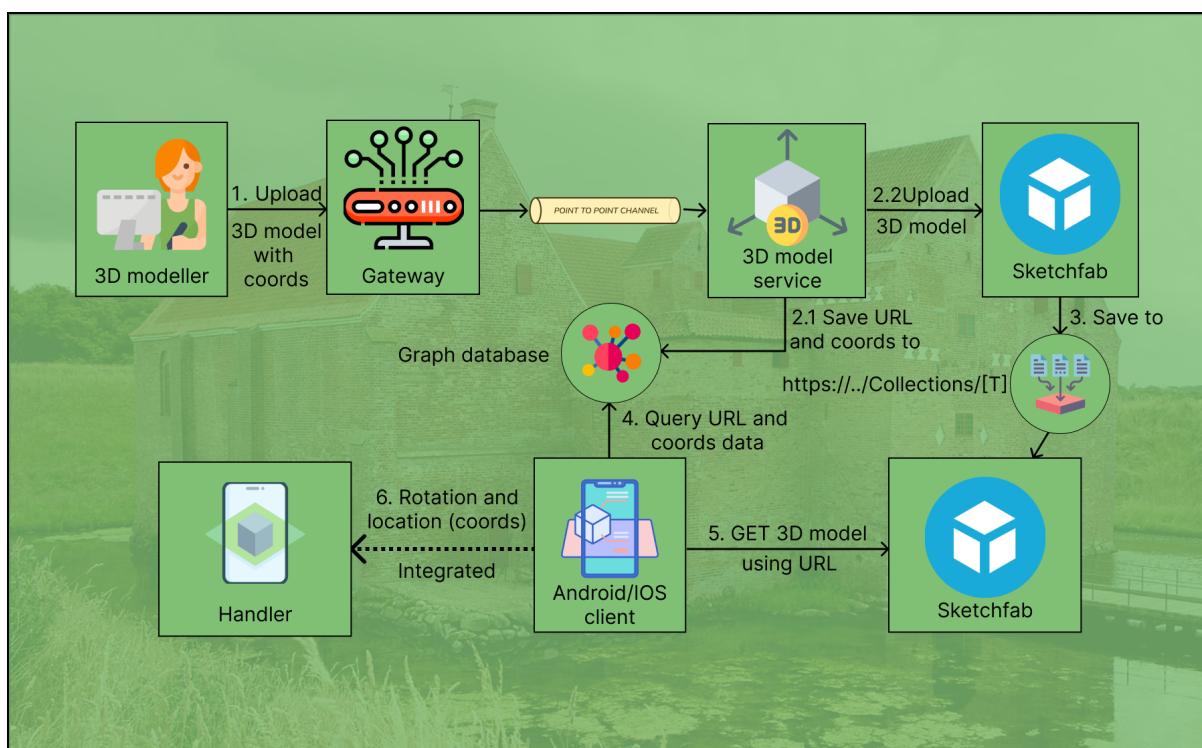
Tilgængeligheden af 3D modeller, (også set på billede 23) kan simplificeres ved at gøre brug af værktøjer såsom *Sketchfab* (59), hvor oprettelse og lagring af 3D modeller sker via en 3. parts løsning.

Dermed kan arkæologer, og fritids hobbyister bidrage til 3D modellering. Dette skal dog igennem en screening process, hvor disse 3D modeller verificeres af en ekspert.

Dette kan gøres ved blot at lade eksperter have adgang til upload funktion, eller notificere eksperter når en 3D model for deres relevante fortidsminde uploades, sådan at den kan verificeres.

Dette skal tages hånd om.

Dette kan gøres ved at oprette en samling i *Sketchfab*, hvorefter at forespørgsler kan forekomme på disse URL'er.



Billede 24. Eksempel for proces for upload af 3D model til visning i AR format hos klienten, ved brug af Sketchfab værktøjet.

Der skal dog være en mulighed for at brugere kan uploadere modeller og billeder til Sketchfab platformen, sådan at disse modeller ender de rigtige steder, og systemet kan varetage disse.

Dette kræver en ny 3D model service der håndterer dette.

Et diskussionsspørgsmål, er placeringen og forarbejdning af diverse 3D modeller og billeder, samt visning af disse via klientens smarttelefon.

Overvejelserne i nuværende fase omhandler en *handler* i klient appen, der tager lokationen af telefonen, og placerer og roterer objektet. Dette kræver at lokationen på objektet er kendt, og at objektet opfører sig statisk, og gøre brug af eventuelle sensorer i telefonen, og lokations dataen via GPS eller GIS.

## Agnostik tilgang

Eftersom ønsket er at skalere til Europa, samt skalere diverse teams i organisationen, og dertil gøre brug af Micro Service mønstret, er det nødvendigt til at designe systemet med en agnostisk tilgang.



Bliver systemet *for bundet af udvalgte teknologier, eller 3. parts services*, vil kravændringer og nye teknologier blive sværere at ændre eller tilføje i fremtiden, hvis dette ønskes.

Finder man en mere effektiv løsning, eller en billigere løsning, kan man blive stoppet i at implementere disse grundet kompleksiteten og ressourcerne det krævet at omskifte.

Men hvordan kan dette løses? Og hvordan undgås en situation som denne?

Sidst spurgte afhænger af systemets kompleksitet, størrelse, tilgængelige ressourcer og levetid, og løses ved effektiv og nøjagtig planlægning, automatiserede processer og nok mest vigtige i denne sammenhæng: systemdesignet.

Men dette er nemmere sagt end gjort. Kompleksiteten af produktet stiger for hver af disse koncepter man dykker dybere ned i.

Tager man udgangspunkt i de nuværende team designs, vil det være muligt for hvert team at kunne udvælge netop de teknologier der passer bedst til deres behov.

Dette skal man dog være opmærksom på, at dette vil kunne skabe nogle eventuelle kommunikationsproblemer, hvis vidensdeling skal ske på tværs af teams.

Dog vil effektiviteten, og fleksibiliteten i både rekruttering af nye teammedlemmer, og implementation af features kunne ske på et højere niveau, eftersom at systemet i sin helhed ikke er bundet i bestemte teknologier.

Eftersom et en Micro Service kun består af *en service*, vil kompleksiteten også bidrage til at kunne konvertere til en anden mere egnet teknologi, uden de store omkostninger (34).

## Caching

Caching spiller også en stor rolle i at dekoble et system, og overholde *swimlane* principippet, samt at øge performance og tilgængelighed i systemet.

Der findes forskellige caching strategier, og disse er værd at undersøge, og dermed konkluderer relevans for systemets krav og design.

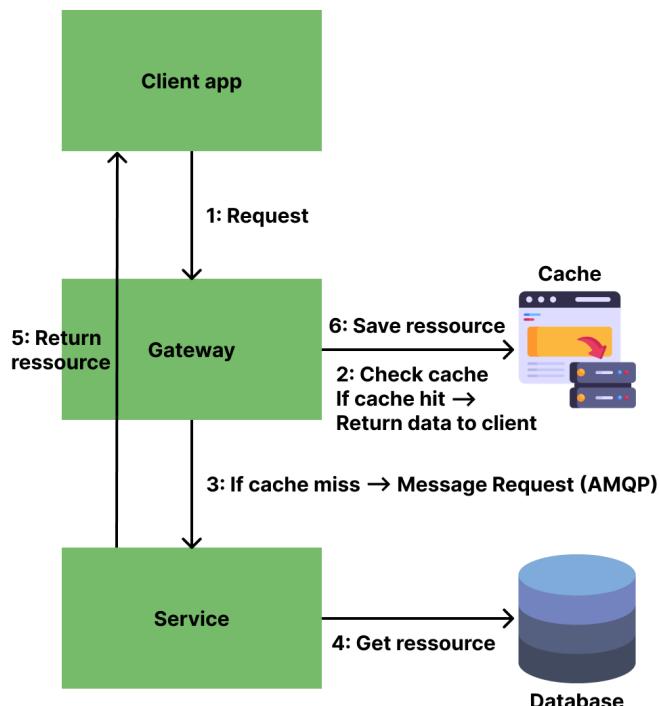
### Caching strategi

Valget af caching strategi, kan betyde performance optimering, og potentielt mindske omkostninger (59).

Der forventes ikke hyppige skrivninger til den bagvedliggende database, og derfor vil en *cache-aside* eller *read-through* cache være gode valg.



Eftersom der kan forekomme skrivninger i form af bruger informationer, og eventuelle profiloplysninger, og siden disse informationer ikke er kritiske, kan cachen behandles, som en write-back cache.



Billede 25. Read-through cache strategy (60).

Valget af en distribueret cache kan også hjælpe med at fjerne flaskehalse ved databasen, og de bagvedliggende services.

Man skal dog ikke cache alt data, da RAM på serveren hurtigt kan blive fyldt op, og derfor skal man vurdere, hvilke data der passer til kravene.

I forhold til *Aggregator*, vil man hurtigt kunne påpege at forespørgsler foretaget gennem denne, gemmes i cachen. Dette kan hurtigt blive uhåndgribeligt, hvis meget data efterspørges, samt flere services kobles på.

Nuværende løsning består i at alle fortidsminder kvalificere sig til at blive gemt i cachen. Derfor skal caching politikken være baseret på at være gemt i et minut, og derefter variere det eftersom at cachen bliver populært med data.

Derfor kræver det også monitorering på cachen, hvorefter at DevOps teamet kan administrere denne.

Der ønskes også et loft på hvor mange entries der skal kunne gemmes i cachen, sådan at den ikke bliver over styrtet af data, og dermed påvirke performance og levetiden for appen.

I fremtiden vil andre politiker kunne vurderes eftersom at den gemte data kan gennemses, og eventuelt caching kan lokalitetsbaseret på den mængde data der gemmes i henhold til.



For eksempel kunne storbyer eller turist tætte områder caches yderligere frem for yderområder der sjældent besøges eller forespørges (25).

## Database

Valg af database skal imødekomme de krav der sættes til projektet, samt det data der skal lagres.

Eftersom at projektet følger et Micro Service mønster, med *Swimlane* principippet, kan valget af database være forskellig mellem diverse services.

Dette gør det muligt at følge et *Polyglot* design, hvor flere databasetyper kan være på spil, og kommunikationen og datamodellering sker gennem services.

Første skridt vil derfor være at undersøge og vælge det database paradigm til de enkelte services.

## Valgt database paradigm

Når det kommer til valget af databasen er der nogle forskellige faktorer omkring dataen, og hvordan det skal anvendes. Til dette ønskes der en database, der kan håndtere store mængder af data. Databasen vil skulle håndtere data og queries hvor at data har adskillige relationer til andre. Dette kan være relationer i forhold til lokation, tidsalder, events og billeder. Dette data kan også variere meget i dets form, alt efter hvilken form for entitet, der er tale om, samt hvad der er tilgængeligt. Det er fx begrænset, hvilke ting der har 3D modeller tilgængelige.

Til at håndtere disse krav vil der blive set videre på en grafdatabase (61).

## Grafdatabase

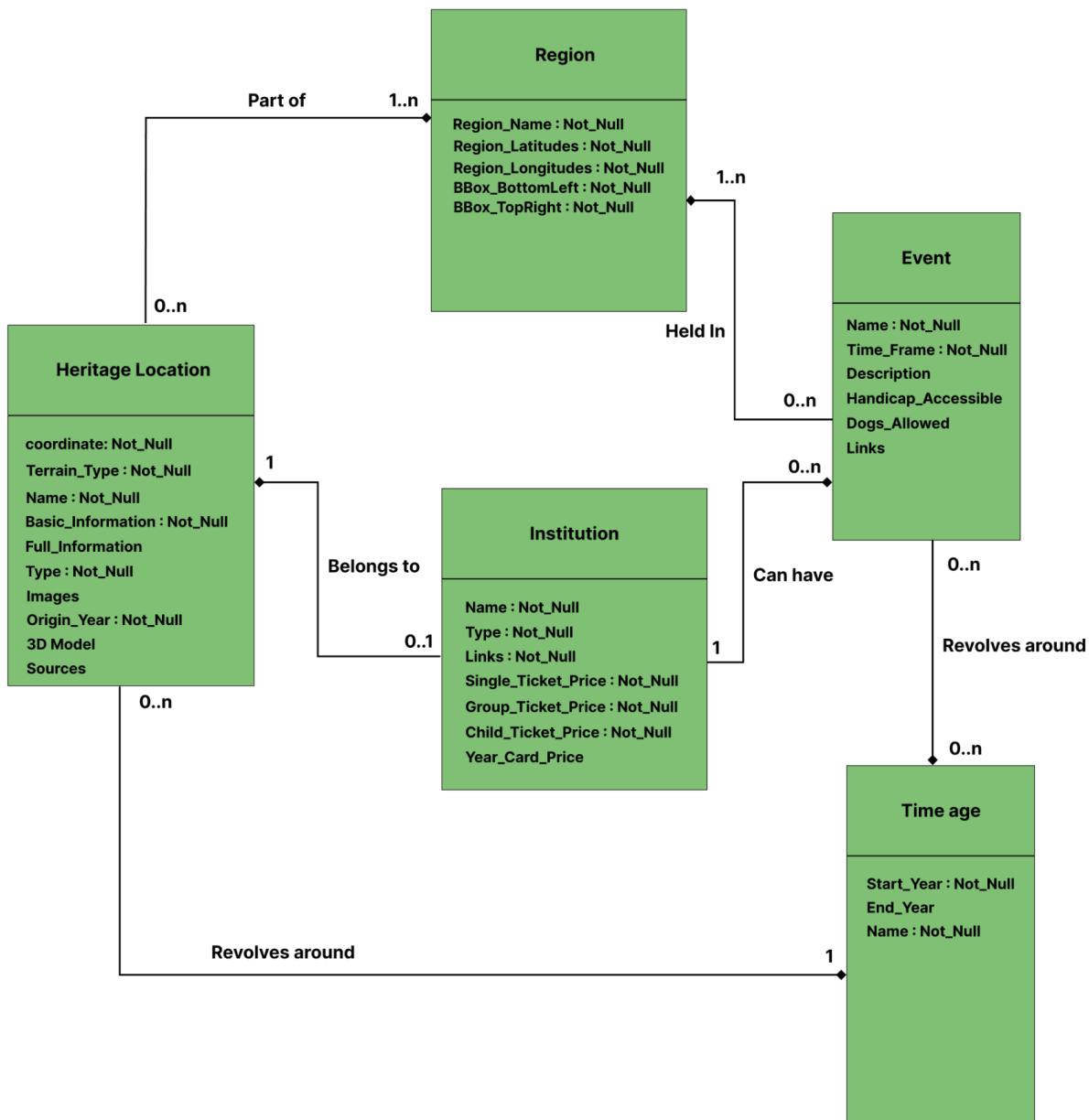
En grafdatabase har som primære fokus og mål at anvende relationen(Edge) mellem entiteter(Nodes). Den primære brug af graf databaser er at anvende disse relationer. Grundet at graf databaser er så fokuseret på relationer, er det meget hurtigt at bevæge sig på tværs af disse relationer. Dette er muligt siden at relationen ikke skal beregnes under querying, den er i stedet gemt på databasen selv.

Graf databasen kommer med flere funktionaliteter, der kan hjælpe med de behov der kommer med dette projekt. Siden at graf databaser hører under NoSQL så er der ikke et krav til et *schema*, hvilket gør håndteringen af varierende data nemmere (62).



## Design af database

Efter at valget vægter på at gå videre med grafdatabase, er det nødvendigt at komme med et design af databasen for hvordan det ønskes at den skal sættes sammen. Baseret på de ønsker til systemet opstilles et generelt design for hvordan at databasen forestilles at skulle designes.



Billede 26. Nuværende grafdatabase design



Billede 26 viser nuværende database design. Til modsætning af en relationel database er dette ikke en opstilling af tabeller. Dette er en opsætning af noder, der er sammenkoblet med diverse relationships. Siden det er en grafdatabase, så er der også meget der er unikt ved disse relationer. Det første er at hvis man ønskede man kan tilføje properties på selve relationerne, da relationerne selv kan hentes, og ikke bare noder. Det andet er at relationerne i grafdatabaser kan have en form for retning. Hvilket vil sige at de kan enten være *directed* eller *undirected*. Under oprettelsen af databasen, har det været nødvendigt at tilføje en retning til disse relationer, men de udvalgte queries ignorere denne retning, da den ikke er relevant for projektet. For at designe databasen korrekt blev der valgt at der skal tages udgangspunkt i de valgte queries, som ville være mest prævalente og hyppigt anvendte. Denne måde at designe på, bliver også videre anbefalet af Neo4j (63). Disse queries inkluderer blandt andet "Top 10 fortidsminder nær dig". Hvor at der gennem sammenligninger af distancen mellem din lokation, og koordinaterne af fortidsminder, inden for hvis radius, bliver valgt de 10 nærmeste. De resterende queries der bliver lagt fokus på til designet, er fortidsminder baseret på ønsket tidsalder, og fortidsminder baseret på tilstedeværelsen af tilkoblede events. Disse queries lagde baggrund til designet af databasen, og er årsagen til at der blandt andet er en separat node til tidsalder, så at den er nemmere at lave søgninger på. Lignende er det blevet besluttet at have Region separat fra de andre noder, da nogle af queries bliver begrænset baseret på en regions bounding box, som er rektangel der omringer den (64).

Til dette system anvendes der også koordinater, da det ønskes at mappe disse lokationer til et kort. Grundet dette ønske er der tilføjet et koordinat til Heritage\_Location. Grundet mulighederne for at have dette som en property, da der er mulighed for at anvende *spatial values* og *functions* til dele af systemets logik (65).

## Delkonklusion

Dette design af databasen er taget med udgangspunkt i et forsøg at få den bedste performance, for de queries, der blev antaget for at have den største værdi samt frekvens. Det blev bestemt at lade være med at flytte koordinatet fra Heritage\_Location til en ny node, på trods af at der var interesse i at bruge det til queries, og sætte et index på det. Dette var dog noget der var oppe til diskussion flere gange om hvorvidt det ville forbedre performance, men det blev nedstemt.

## Queries

For at have noget at tage udgangspunkt i rapporten, når der senere skal ses på optimeringer og indeksering. Så vil der blive opstillet 5 queries, som er baseret på, hvad systemet ofte vil blive brugt til.



### 1) Top 10 fortidsminder nær dig.

Denne query vil være den mest anvendte og vil være en query, der vil blive udført ofte. Denne query vil tage udgangspunkt i givet koordinater. Herefter vil den se efter fortidsminder med den korteste distance til ens nuværende position. Den vil returnere en liste over de 10 nærmeste fortidsminder, med information omkring fortidsmindet, basis information, ekstra information og eventuelt andre midler.

```
With point({longitude: 9, latitude:56}) as yourLocation
MATCH (h:Heritage_Location)
WHERE point.distance(yourLocation, h.Coordinate) < 50000
RETURN h.Name, h.Coordinate, h.Terrain_Type, h.Basic_Information, h.Origin_Year, h.Type
ORDER BY point.distance(yourLocation, h.Coordinate) ASC
limit 10
```

Billede 27. Top 10 fortidsminder nær dig.

Hvis man ser på denne query vil det give mening at se på effekten af et sekundært indeks på Heritage\_Location.Coordinate. Da det er et koordinat der vil blive sat et sekundært indeks på, vil det være nødvendigt at anvende et B-Tree indeks. Siden at B-Tree indeks er *deprecated*, vil det skulle erstattes på sigt med et af de nye indekser der bliver præsenteret i næste afsnit.

### 2) Søg efter alle fortidsminder baseret på tidsalder.

Den næste query der fokuseres på er muligheden for at se efter fortidsminder baseret på tidsalder, siden at dette kan blive en query der returnerer mange resultater, specificeres en region.

```
match(t:Time_Age)-[r:REVOLVES_AROUND]-(h:Heritage_Location)-[r1:PART_OF]-(reg:Region)
where t.Name = 'Bronze age' and reg.Region_Name = 'Denmark'
return
t.Name, h.Name, h.Coordinate, h.Terrain_Type,
h.Basic_Information, h.Origin_Year, h.Type
```

Billede 28. Fortidsminder indenfor en bestemt tidsalder.

Når det kommer til sekundære indekser ved denne query er der både Time\_Age.Name og Region.Region\_Name. Indekset der vil give mest mening her er tekst indeksering. Da den bliver brugt til strings.

### 3) Søg efter alle fortidsminder indenfor en region.



Denne query vil også være meget anvendt, det vil være den default søgefunktion hvis man bare søger efter en region.

```
match(h:Heritage_Location)-[r1:PART_OF]-(reg:Region)
where reg.Region_Name = 'Denmark'
return
h.Name, h.Coordinate, h.Terrain_Type,
h.Basic_Information, h.Origin_Year, h.Type
```

Billede 29. Fortidsminder indenfor en bestemt region.

Den vil udelukkende tage udgangspunkt i det givne regions navn, og returnerer det der har en relation til denne. Lignende forrige queries, kan det have fordel ved at have et indeks på Region\_Name.

- 4) Søg efter alle fortidsminder med en specifik type.

Denne query tager udgangspunkt i en region, samt en type af fortidsminde. Det vil være et forholdsvis typisk scenerie, hvis man er interesseret i en specifik form for fortidsminde.

```
match(h:Heritage_Location)-[r:PART_OF]-(reg:Region)
where h.Type = 'Kirke' and reg.Region_Name = 'Denmark'
return h.Name, h.Coordinate, h.Terrain_Type,
h.Basic_Information, h.Origin_Year, h.Type
```

Billede 30. Fortidsminder indenfor en bestemt type og region.

Udover det vil være relevant at se på et indeks til Region\_Name som før er blevet nævnt, men her vil også kunne hjælpe med et tekstbaseret sekundært indeks på Heritage\_Location.Type.

- 5) Søg efter fortidsminder med en forbundet event.

Den sidste query, der er blevet valgt, fokuserer på dem der er interesserede i events. Disse events kan fx highlighte forskellige fortidsminder, og kan gøre hele oplevelsen mere social. Dette vil være events der afholdes af institutioner, hvor det enten er forbundet med fortidsminder hos dem, eller der afholdes noget vedrørende et fortidsminde, hvor transport er involveret. Den sikre sig kun at medtagne fortidsminder, har en oprettet event.



```
match (reg:Region)
where reg.Region_Name = 'Denmark'
call{
    with reg
    match (h:Heritage_Location)-[r:BELONGS_TO]-(i:Institution)-[r2:CAN_HAVE]-(e:Event)
    where exists((i)-[r2]-(e)) and point.withinBBox(h.Coordinate, reg.Bbox_Bottom_Left, reg.Bbox_Top_Right)
    return h
}
return h.Name, h.Coordinate, h.Terrain_Type,
h.Basic_Information, h.Origin_Year, h.Type
```

Billede 32. Fortidsminder indenfor en bestemt bestemt region med events tilkoblet.

Denne query tilbyder muligheden for at lave endnu et sekundært indeks på koordinaterne der udgør dens *bounding box*. Dette bliver dog valgt at undgåes da effekten af dette vil være tvivlsom. Det kan overvejes yderligere når at de indekser der skal erstatte B-Tree er implementeret, men selv i denne situation er det tvivlsomt.

## Optimeriseringer og indeksering

Som en vigtig del af beskrivelsen og overvejelserne omkring databaserne giver det mening at se på hvilke muligheder, der eksisterer for at optimere nuværende queries. Modsat de relationelle databaser, hvor at det er muligt at anvende relationel algebra til at beskrive en query og dens forventede kompleksitet, så virker denne mulighed ikke til at eksistere ved graf databaser. Baserede på hvad der er blevet undersøgt af denne gruppe, eksisterer der ikke en ækvivalent form når det kommer til graf databaser. På trods af dette er der stadig rigelig mulighed for at tune og optimere queries. Dette kan blandt andet formåes ved at se på eksekverings planer.

Ved at undersøge en queries eksekveringsplan, er det muligt at finde ud af hvordan den har tænkt sig at udføre den query du har valgt at eksekvere. Dette inkluderer blandt andet, hvordan den får fat i resultater og alle de handlinger der er involveret.

## Indeksering

Når det kommer til indeksering i Neo4j, så er der nogen forskellige former for sekundære indekser der kan bruges. Den første af disse er *B-tree* indeksering. Denne form for indeksering har været anvendt til de fleste tilfælde af indeksering i Neo4j. Den anden form for indeksering der har været anvendt meget før, er tekstbaseret indeksering. Denne form for indeksering kan kun blive udført på strings, og bruges i forbindelse med keywords såsom, CONTAINS og ENDS WITH. Tekst baseret indeksering kan både være dele af en streng eller hele strengen. Disse to former bliver defineret separat med hver sit keyword. Den sidste indekserings form der eksisterer på nuværende tidspunkt, er token baseret indeksering. Denne form er brugt til at indeksere *labels* og *relationstyper*.



I forbindelse med dette system, vil det give fin mening at anvende en form for sekundær indeks, der kan hjælpe med performance og formindske nødvendigheden af at søge gennem hele samlingen af noder under et specifikt Label, der ledes igennem.

De indekser der vil blive valgt til dette projekt vil være en kombination af B-tree og tekst baseret indekser. Dette vil kun være en midlertidig løsning, da at B-tree indeksering er *deprecated*. B-tree indeksering vil i en senere version blive erstattet af et par andre former for indeksering. Disse indekser vil være et point og range indeks, der sammen med tekst indeks vil erstatte B-tree indeksets funktionalitet.

Denne nye form for indeks vil være specielt interessant for dette projekt, grundet brugen af koordinater, distance funktionen og bounding boxes (66).

## Queries

Med udgangspunkt i de tidligere præsenterede queries, vil de nu blive gennemgået sekventielt for at bestemme om hvilke, hvis nogen, sekundære indekser kan forbedre dets performance. Disse queries bliver udført på grafdatabasen, hvor der er indsat dummy data, for bedre at kunne se en eventuel forbedring i performance grundet de sekundære indekser.

De sekundære indekser der vil blive implementeret er som følger. Først vil der forsøges at implementere B-Tree indeks på Heritage\_Location.Coordinate. Dette skyldes at labellet Heritage\_Location, vil med alt sandsynlighed blive den med flest noder. I forbindelse med det, ønskes det at udnytte muligheden for at anvende de spatial values der tilbydes af Neo4j. Her tænkes det også at siden der bliver lavet beregninger baseret på, hvor tæt et fortidsminde er på personen der søger. Så vil en indeksering på dette koordinat forbedre den nødvendige beregning, da der ønskes at finde de tætteste fortidsminder baseret på distance.

Udover dette vil der også forsøges at anvende et sekundært indeks på Region.Region\_Name. Dette skyldes at dette er et ofte brugt parameter i de præsenterede queries, da den hjælper med at begrænse mængden af mulige resultater. Forventningen med dette indeks, er mere en forbedring af performance på sigt. Dette skyldes at mængden af regioner, med de nuværende dummy data er begrænset. Dette vil sige at selv hvis der var en ændring ved at tilføje dette indeks, vil det først give mening når at mængden af noder i Region er steget nok. Grundet dette forventes den kommende test af eksekveringsplaner, ikke at have en betydelig effekt. Til sidst vil der også tilføjes et tekst indeks på Heritage\_Location.Type. Den anvendes til at filtrere resultater så der kun returneres den type man er interesseret i. Dette indeks skyldes at Heritage\_Location igen er det Label med flest forventede noder. Grundet dette vil et



indeks kunne have stor effekt. Derudover vil typerne gå igen adskillige gange hvilket betyder, at der forventes at de resterende med den type findes nemmere.

Baseret på de queries og indekser, der er blevet introduceret nu, vil det næste skridt være at se på eksekveringsplanen før og efter at indekserne er blevet tilføjet. Den første der vil blive set nærmere på, er den der tjekker top 10 nærmeste fortidsminder. Når man anvender neo4j browser til at se eksekverings planen for denne får man følgende flow.



Billede 33. 10 nærmeste fortidsminder eksekveringsplan før indeksering.

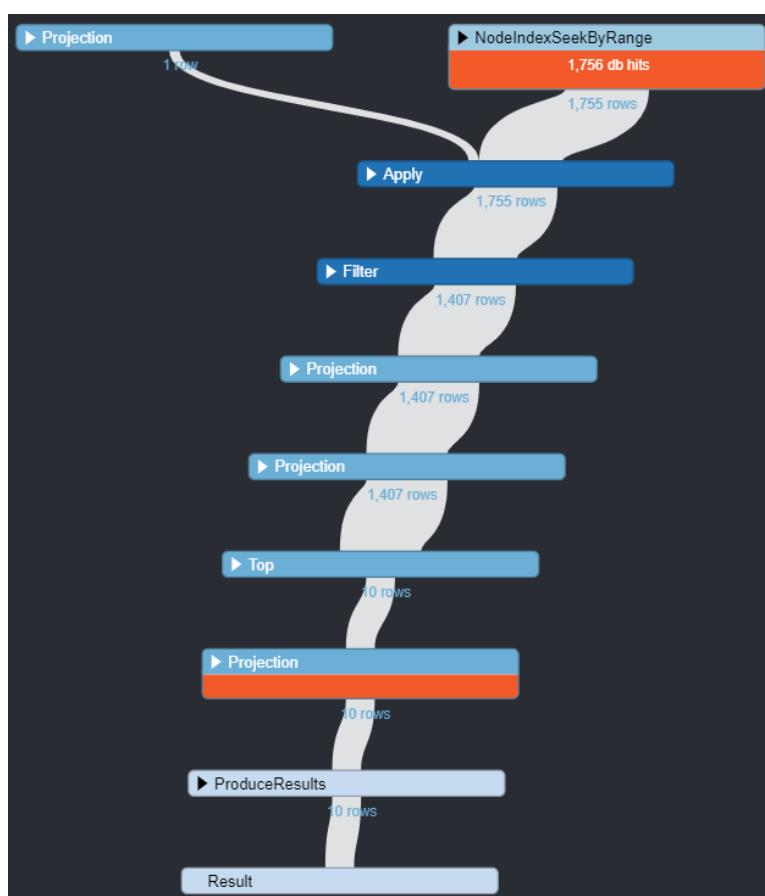
Denne eksekveringsplan viser de forskellige steps der er involveret i at udføre den ønskede query. Denne query resulterede i 45101 database hits og tog 248 ms.<sup>1</sup> Database hits i denne sammenhæng er en abstrakt enhed, der anvendes til at beskrive, når der hentes eller opdateres mm. Detaljerne for hvad der kan beskrives som et database hit, kan ses her (67). Hvis man gennemgår hvordan eksekveringsplanen for queryen opfører sig. Så kan man se at den starter ud med en Scan baseret på labellet Heritage\_Location. Hvilket betyder at den går igennem hver eneste node i dette label. Ved at sætte indexet ind på Heritage\_Location.Coordinate, forventes der at den ikke

<sup>1</sup> Projection i denne sammenhæng vil være når man navngiver et variabel til senere brug.



længere behøves at gennemgå alle noder. Herefter går den igennem hver eneste node, og ser hvor vidt den passer med den "Where" klausul der er sat op, da der er sat en grænse på en maks distance af 50 km før at den bliver overvejet<sup>2</sup>. Denne del af eksekveringsplanen er også hvor mængden af database hits eskalerer drastisk, og ved brugen af indexet forventes det også at denne operation vil være formindsket. Dette skyldes at mængden den vil skulle igennem også ville være markant mindre. Efter denne operation laver den nogle projekteringer hvorefter den finder de 10 nærmeste lokationer.

Efter at der er tilføjet indeksering vil eksekverings planen for samme query se ud som følger.



Billede 34. 10 nærmeste fortidsminder eksekveringsplan efter indeksering.

Ved sammenligning af billederne for de 10 nærmeste fortidsminder før og efter indeksering, kan der ses en drastisk ændring. Dette kan blandt andet ses at i mængden af database hits før indeksering lå på 45101 sammenlagt, hvor at det tog 248 millisekunder. Hvor efter indeksering tog det 1856 database hits som tog 155 millisekunder. Dette er muligt grundet at eksekveringsplanen ikke længere anvender en NodeByLabelScan, men i stedet bruger NodeIndexSeekByRange. Dette blev muliggjort

<sup>2</sup> Filter i denne sammenhæng er hvor håndteringen af where klausulen gennemgås.



gennem indexet, der blev tilføjet, hvilket resulterede i at kun visse noder under Heritage\_Location labellet behøvede at blive gennemgået.

Den næste eksekveringsplan der bliver set på er fortidsminder indenfor en specificeret region.



Billede 35: fortidsminder baseret på region eksekveringsplan før indeksering.

Denne eksekveringsplan blev udført med 26279 database hits, som tog 48 millisekunder.

Som det kan ses fra den tidligere eksekveringsplan bruges der også den samme form for label scan. Forskellen med denne er at denne eksekveringsplan forsøges der at lave et indeks på Region.Region\_Name. Dette indeks forsøges, da det forventes at ville give mere mening på sigt, når mængden af regioner stiger. Grundet dette forventes der ikke den store forandring, når at indekset er tilføjet.

Efter at indekset er blevet tilføjet ser den nye eksekveringsplan ud som følgende.



Billedet viser den eksekveringsplan, der vil blive set på fortidsminder baseret på type og region.

Denne eksekveringsplan viser sig ikke at have ændret sig meget efter at have fået et indeks tilføjet. Den eneste der er sket er at den er gået over til NodeIndexSeek i stedet for en skanning af hele labellet. Som forventet så er denne ændring ikke særlig markant og fjerne kun få resultater grundet det begrænsede dataset af regioner. Siden dette er tilfælde er det ikke for overraskende at mængden af database hits ikke har ændret sig med mere end 20. Tværtimod så er tiden det tager querying at blive udført steget til 100 millisekunder. Dette kan skyldes påvirkningen af caching der foregår i neo4j. En minimal stigning i tid, kan forklares med det ekstra der kræves ved at tilføje indekseringen. Men grundet at omfanget af denne indeksering er så minimal, virker det usandsynligt. Grundet den begrænsede mængde af regioner i det nuværende dataset vil et indeks i denne sammenhæng ikke give mening. Dog som systemet skaleres, vil det give mening at genoverveje, hvorvidt et indeks i denne situation ville virke.

Den sidste eksekveringsplan der vil bliver set på fortidsminder baseret på type og region.



Billede 37: Fortidsminder baseret på type og region før indeksering.

Denne eksekveringsplan viser sig til at være meget lignende den forrige, der udelukkende var baseret på Region. Denne query har et startpunkt med et total af 6576 database hits før indeksering, som tog 48 millisekunder. Denne query anvender to indekser, hvor den ene af dem er Region\_Name som blev anvendt tidligere, som viste ingen betydelig effekt på hvordan eksekveringsplanen blev udformet. Dog grundet muligheden for at kunne udføre en anden form for scanning lignende den første viste eksekveringsplanen, så forventes det, at der vil være en ændring af eksekveringsplanen efter indeksering.



Billede 38: Fortidsminder baseret på type og region efter indeksering.

Denne eksekveringsplan viser resultatet efter tilføjelsen af indeksering. På denne eksekveringsplan kan det ses at indekseringen ikke har udført en større ændring i fx database hits. Det interessante her er dog at den virker til at udføres på samme måde som hvis at indekseringen på Heritage\_Location.Type ikke eksisterede. Det viser sig faktisk at querying nu tager 100 millisekunder ekstra. Dette skyldes højst sandsynligt det ekstra at indekseringen kræver. Dog noget af dette kan også skyldes det caching der er til stede på neo4j. Denne ændring viser at det umiddelbart ikke giver nogen som helst stigning i performance at tilføje indeksering på Type.

## Del diskussion

I denne sektion har der været meget fokus på valget af database og det design der er valgt at gå videre med. Med det i mente blev der fundet flere overraskelser under forsøget på at optimere de valgte queries. En af de største overraskelser her var blandt andet indekset der var sat på Heritage\_Location. Type ikke virkede til at have nogen effekt på querying omkring fortidsminder baseret på type og region. Dette kan skyldes måden en grafdatabase får fat i de resterende noder efter start noden. Grundet dette vil det ikke give mening at fortsætte med et indeks på type. Dog kan det diskuteres hvorvidt et indeks på en kombination af Region.Region\_Name og



Heritage\_Location.Type, kan give mening. Dog på nuværende tidspunkt vil dette ikke undersøges videre.

## Sikkerhed og GDPR

I dette projekt vil det være nødvendigt at overveje nogle af de forskellige former for sikkerhed, der er nødvendige. Dette skyldes at systemet indsamler personfølsomme data. Dette forekommer blandt andet i det ønske at have noget *authentication* til stede i systemet (Kan bruges til yderligere udvikling af butik, betaling, favoritliste mfl). Dette vil kræve at brugerne angiver noget information, man kan identificere dem på, når de bruger systemet. Der vil også være nødt til at være fokus på GDPR, da der anvendes lokationsdata, hvis at brugeren tillader dette. På den måde kan der justeres anbefalinger, og til at kunne på sigt anvende GPS funktionaliteter. Som en del af designet af app'en vil der være en meddelse man skal reagere på, om hvorvidt brugeren tillader at bruge enhedens lokation. Hvis dette afvises vil der blive lavet en generel visning ved indgang til appen, hvor at systemet ikke tager højde for en start position. Siden systemet på sigt vil komme til at have services der involverer køb af vare, så vil der være nødvendigt at være forsigtige med håndtering af dette. Det er specielt gældende i forhold til hvilket information der opbevares i databasen.

Dertil vil det også være en mulighed at bruge en anerkendt 3. part, hvorpå ansvaret for håndtering af kreditkort data, og håndtering af betaling ikke håndteres af det interne system.

På denne måde imødekommes eventuelle sårbarheder i denne henseende, men også at der kan fokuseres på kerneproduktet i stedet.

## JSON Web Tokens

JSON Web Tokens (JWT) er et alternativ til den klassiske *cookie* (68), hvorpå ansvaret fjernes fra blandt andet diverse gateways i at holde styr på bruger sessioner.

Tokens bliver placeret på brugerens side, og derfor skal man være observante på de sikkerhedsrisici dette indebærer.

OWASP (69) har som en guide, informeret omkring de sårbarheder, diverse virksomheder verden over står overfor, netop ved brug af JWT'er.

De nævnte sårbarheder OWASP fremlægger, kan løses ved at blackliste tilbagekaldte JWT'er, indtil disse udløber, undgå vigtig information at blive gemt sammen med (For eksempel rettigheder, simple ID'er, indkøbskurv information, kodeord mfl). Derudover skal der også sikres at man hasher hver token, og sikre at denne valideres. Dertil skal der også sikres at det bibliotek, der gøres brug af gemmer hver token i *Token Storage* på



brugerens side, da adgang til disse vil blive udeladt, ved blandt andet *Cross Site Scripting* (XSS) (70) angreb.

Derfor når der udvælges et bibliotek der skal være systemets JWT generator og validator, skal dette sikre systemet kan tage hånd om disse.

Der findes adskillige biblioteker der kan bruges til at implementere JWT'er i sit program. JWT.io har opsat en række open source muligheder via deres hjemmeside (71), og her kan udviklerne uanset programmeringssprog/paradigme få et overblik over features og sikkerhedsmuligheder der står til rådighed i hvert bibliotek.

Valget af bibliotek er irrelevant eftersom systemet skal være agnostisk, men at det vigtigst af alt at det valgte bibliotek overholder de sikkerhedsregler, som før nævnt.

## Test og produktkvalitet

Som en del af at have kvalitetssikring vil det selvfølgelig være nødvendigt at bedre uddybe, hvordan at test vil håndteres i systemet. Testning vil selvfølgelig være en vigtig del i hver eneste iteration og fase af systemet. Dette skyldes den nuværende opdeling og videreudvikling af systemet. Som er baseret på de strategiske mål, samt tidlige emner nævnt i denne rapporten. Grundet dette vil formatet af systemets måde og håndtere test tage en mere generel fremgangsmåde, både i starten og senere som organisationen skaleres. Med dette menes der, at der vil fokuseres på at sætte en generel template op for håndteringen af test. Dette skyldes at som så mange andre emner i denne rapport, så er det yderst vigtigt, at der tilbydes fleksibilitet. Med dette i mente vil der sættes generelle retningslinjer op for håndteringen af test, der kan lettet rettes til det enkelte holds og organisationens krav på det tidspunkt.

## Teststrategi

Valget af teststrategi variere eftersom projektet og organisationen ændrer sig, og behovet skifter.

Dermed sagt er det vigtigt at vælge sin teststrategi(er), baseret på nuværende fase af projektet.

Der er et krav til systemet skal være *maintainable*, høj tilgængelighed, men samtidig også sigte efter TTM (time to market).

Derfor skal der være nogle kompenseringer, da man med høj vedligeholdelse er på bekostning af TTM.

Derfor kan man tage udgangspunkt i de KPI'er der er blevet opstillet, og de nuværende strategiske mål.

Valget ligger på en *Model-Based, Regression-Averse* (72) strategi, eftersom fornævnt



Udover dette er der også lagt vægt på en *Reactive* (sammen med en *Exploratory* tilgang) testing fremgangsmåde er foretrukket, da tilgængeligheden kan blive påvirket af defekter.

I testfasen af ALM, vil man med en *Reactive* og *Exploratory* fremgangsmåde give testerne mulighed for at minimere dokumentation, og fokusere mere på logging tests, og samtidig give mulighed for at udforske softwaren fremfor forudsætninger.

Dette kræver også at testerne i hvert team har god forståelse for denne fremgangsmåde, men kan i løbet af projektets levetid gøre testerne klogere og mere indsigtfulde i produktets stærke og svage sider.

Og med en *Model-Based* fremgangsmåde, samt *Regression-Averse* vil kvaliteten af produktet stadig følge nogle krav baserede test cases, sådan at kernekvaliteten bevares. Som fornævnt kan strategierne ændre sig, eftersom at projektet og organisationen ændrer sig, men i nuværende fase ses de nuværende udvalgte strategier som matcher de strategiske mål og KPI'er.

## Test logging

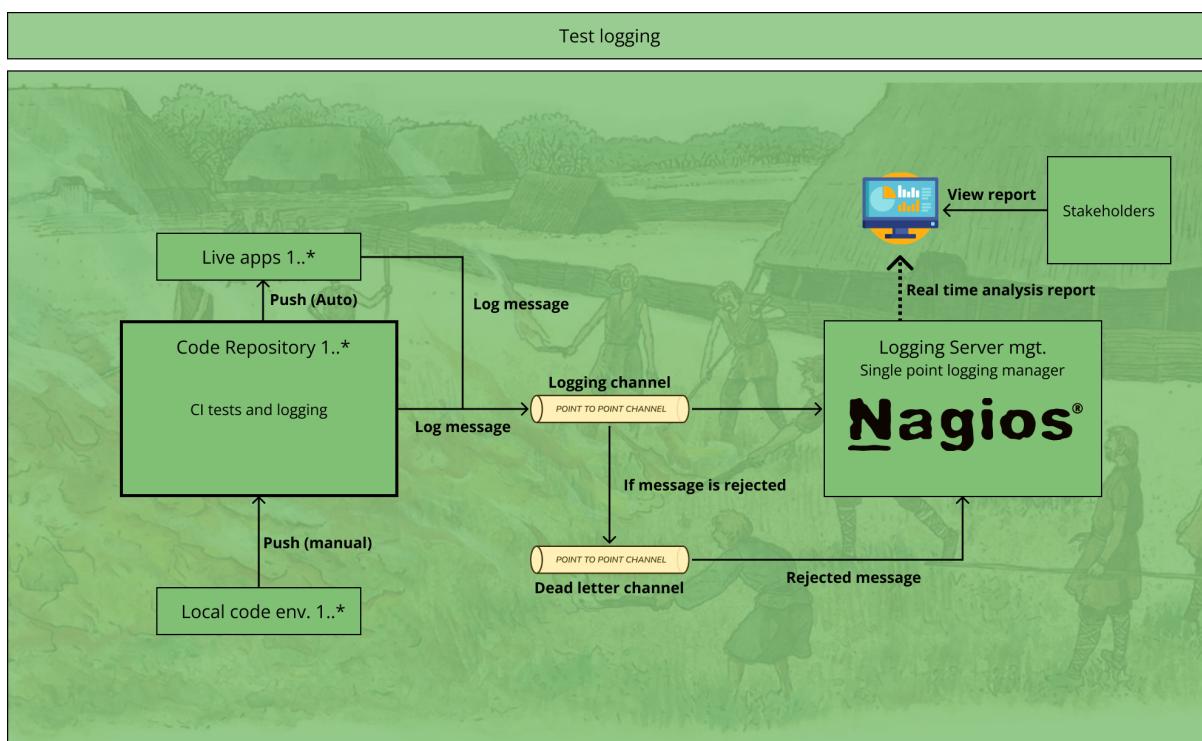
Eftersom at dokumentation ikke er i fokus i den *Reactive* og *Exploratory* strategi fremgangsmåde, skal fokus derfor være i logging.

Logging skal selvfølgelig også indtræde i de mere formelle tests, da logging kan give et mere klart billede af produktets opførsel.

Og dette falder især også i god jord med *Regression-Averse* strategien, da logging af allerede eksisterende tests kan se ændringer i værdier og udfald af nuværende tests.

Dermed sagt skal man, ligesom med monitorering af systemet, være ørvågen over for for meget logging data, sådan at disse bliver uoverskuelige at overvære.

Derfor vil logging af fejlene tests kun være nødvendige (73).



Billede 39. Centraliseret logging manager, til håndtering af test logging og live/production miljø logging, ved brug af Nagios værktøjet (73).

Jævnfør billede 40 er test logging en del af hele processen, fra udvikling til live-miljø. For at simplificere og centralisere logging fra testing resultater samt live miljøet, vil en samlet logging server blive opstillet.

Her vil Nagios være et værktøj der kan tages i brug, som netop har en service til rådighed for at håndtere dette, hvor real-time analyse af logging resultater kan vises. Valget af denne løsning ligger i *Buy-When-Non-Core* principippet, sådan at TTM kan opnås hurtigere, samt et gennemtestet værktøj der kan løse logging informationer for projektet er derfor oplagt.

Ved at have et visualiseringsværktøj, kan dette påvises rundt i virksomheden (internt og eksternt), og dermed give interessenter mulighed i at følge hele udviklingsprocessen, samt helbredet for hostede applikationerne.

Efter *Continuous Deployment* er en del af projektet, er det også vigtigt at samle logging informationer (monitorering), sådan at godkendt kode er skubbet til live-miljøet, kan følge påvirkninger, hvis disse skulle forekomme.

Dette hjælper også på *maintainability* og *traceability*, sådan at defekter kan spottes og løses hurtigere.



## Defekt håndtering

I forlængelse af logging, skal man kunne håndtere de fejl der nu opstår når disse optræder.

Derudover skal det også være muligt at kunne indrapportere og løse bugs og defekter, som ikke spottes af tests, men af eventuelle udviklere eller testere.

Det skal også konkretiseres hvad log-beskederne skal indeholder, sådan at rapportering og håndtering foregår formelt, og gnidningsfrit.

Hvis bugs eller defekter spottes uden for løbende tests skal disse indrapporteres via projektstyringsværktøjer.

Dette gælder også med logging-beskeder hvor fejl opstår.

Dette kan integreres i via værktøjer som *Jira* (74), og teamet kan dermed inkludere bugs i backloggen, og blive fikset.

Vigtige punkter at have med i en fejlrappor er:

- Tidspunkt for opstået fejl.
- Unik ID for Bug Story (User Story ækvivalent til bugs).
- ID for log eller rapport i forlængelse af user story (*traceability*).
- Navn på person der har opdaget fejlen.
- Prioriteret i forhold til risiko (Kritikalitet).
- Beskrivelse af defekten, og hvor og hvorfor den opstod.
- Tidspunkt for fikset. (Sker normalt automatisk gennem et projektstyringsværktøj).
- Navn på person der har fikset fejlen (Sker normalt automatisk gennem et projektstyringsværktøj).

Normalt i et CI værktøj kan udviklerne i real-tid følge med i de gennemførte tests, og fikse fejlen inden det skubbes til et live-miljø.

Men skulle der opstå bugs, fejl eller defekter i et live miljø er ovenstående relevant.

Men hvordan skal en bug, defekt eller fejl prioriteres, defineres og videregives til udviklerne?

Hvad skal der ske hvis DevOps teamet finder defekter eller fejl i systemet?

Dette løses ved at indkalde (jævnfør billede 6 i *Organisationsopstilling* afsnittet), sammen med den eller de personer der har fundet defekten eller fejlen.

Her skal der vurderes hvad der årsag og tidspunkt for defekten eller fejlen, og der udføres en beskrivelse.

Derefter prioriteres den i forhold til risiko og kritikalitet for projektet samt indføres i backloggen efter sigende.



Dermed vil den blive løst efter dens kritikalitet. Er kritikaliteten høj, skal udvikler teamet løse denne med det samme.

Er den lav, kan den fikses i næste sprint, eller nuværende sprint hvis teamet har tid og ressourcer til dette.

På denne måde vil støj i udvikler teamet forblive på et minimum, da kun én repræsentant er med i processen, og processen for videregivelse af en bug story sker forholdsvis automatisk ind i udvikler processen (75).

## “Definition of Ready” og “Definition of Done”

For at bedst kunne identificere userstories, der er klar til implementering og hvornår de kan blive sendt videre til verificering. Til dette vil der blive fastlagt en overordnet definition af hvornår noget er ”Ready” og ”Done”. Denne definition skal ses som en template for hvad organisationen finder vigtige ved alle hold. De individuelle hold vil være i stand til at kunne modifcere denne definition. Som udgangspunkt så vil den generelle definition af hvornår noget er Ready lyde som følger.

### Definition of Ready

No code may be written until we first define the acceptance criterias/tests and test cases

A story can be deemed to be independent from other stories

A story should be small enough for one developer to complete

A story is estimated with a Fibonacci number under 13

As a <role> I want <feature> so that <benefit> title template for User Stories

Denne definition er lavet så at en user story er blevet defineret nok til at kunne blive udført indenfor en periode på højst et par dage. Dette sikre også at kompleksiteten ikke bliver stor ved at sikre at den defineres til noget der er håndgribeligt. Denne restriktion på estimeringen er også noget der kan tilpasses teamets behov.

Den anden definition er hvornår et team kan vurdere at en *User Story* er ”Done”.

Denne definition lyder som følger



## Definition of Done

80% statement coverage

100% decision coverage

Low Cyclomatic complexity between 1-15

Minimum 95% of tests have passed

Scrum board is updated

No known defects

No build failures

Documentation updated

Denne definition af "Done" beskriver tidspunktet, hvor en user storyen er klar til at sendes til validering, med en PO. Det er valgt at prioritere decision coverage i test casene fremfor statement coverage. Dette skyldes at der prioriteres meget højere at alle beslutninger der foretages bliver eksekveret og testet, frem for nødvendigheden at teste hver eneste linje kode. Højere prioritering af testning af hver eneste linje kode anses i dette projekt som at gå imod de strategiske mål om time to market. Derudover prioriteres der også at den *Cyclomatic* kompleksitet skal holdes nede for at undgå metoder, der er for kompliceret til at teste tilstrækkeligt (76).

Disse definition vil blive anvendt i alle teams, for bedst at kunne opretholde en vis kvalitetskontrol. Med det sagt så er disse definitioner anset, som en template, som hvert team skal hjælpe med at forme til deres omstændigheder.

## Test discipliner

For at hjælpe med processen i at udføre test og hvordan de bliver gjort så fyldestgørende som muligt, er der valgt diverse værktøjer og teknikker.

### Moq

Der vil blive anvendt Moq i udviklingsprocessen. Dette er et værktøj der kan skabe Mock objekter i forbindelse med testfasen. Dette værktøj bruges blandt andet til at facilitere TDD, så der kan udføres test på områder, der ikke er implementeret endnu (77).



## Blackbox og white box testing

Der vil blive anvendt både black og white box testing discipliner i dette system. White box testing vil blive anvendt til at teste systemets indre struktur, hvor at organisationen kontrollere de services der bliver anvendt. Dette muliggør den baggrundsviden, der er nødvendig for at facilitere white box testing. Hvorimod at black box testing vil blive brugt i forbindelse med kontakt med tredjeparts systemer, der kobles på. Da der ikke er kendskab til det bagvedliggende logik (78) (79).

## DAMP

I forbindelse med dette projekt, og for at sikre bedre læsbarhed af kodden, så vil der anvendes DAMP for at promovere bedre forståelse (80).

Dette er en standard der skal hjælpe med at gennemlæse tests hurtigere, mindske forvirring af kodebasen.

## TDD

Test driven development anvendes i dette system som et grundlag for, hvordan de diverse teams sikre sig høj coverage med deres testning. Denne disciplin er specielt vigtig siden, der ønskes høj decision coverage og statement coverage (81).

I forlængelse af *Definition of Ready* skrives hver testcase før aktuel kode skrives.

Dette sikre også at man tester det konkrete mere præcist.

Og i et *Continuous Integration* miljø, vil hver User Story også have et sæt af test cases, der vil blive eksekveret hver gang noget skubbes til et code repository.

## Test Automations

Som en del af testprocessen, vil der blandt andet være brug for muligheden, at bruge automatiserede test, specielt siden der ønskes at være "Regression averse". Siden det allerede er uddybet, at der er planlagt at anvende CI/CD, så vil det give mening at anvende et værktøj til håndteringen af dette. Her er der flere mulige værktøjer og vælge imellem blandt andet GitLab, CircleCI og Jenkins (37). Dette system vil dog fortsætte med Jenkins, da den er open source, simpelt og nemt at sætte op. Dette er specielt relevant hvis det ønskes at hjælpe på Time to market.

## Boundary Testing

For at opnå en mere effektiv form for decision coverage, vil der anvendes boundary testing, for at reducere mængden af nødvendige test cases (82).

I stedet for at teste adskillige cases hvor man forventer et godkendt resultat, kan man spare tid ved at boundary teste. Udover dette sikre man også at værdier udenfor en ramme, ikke godkendes.



## Maze

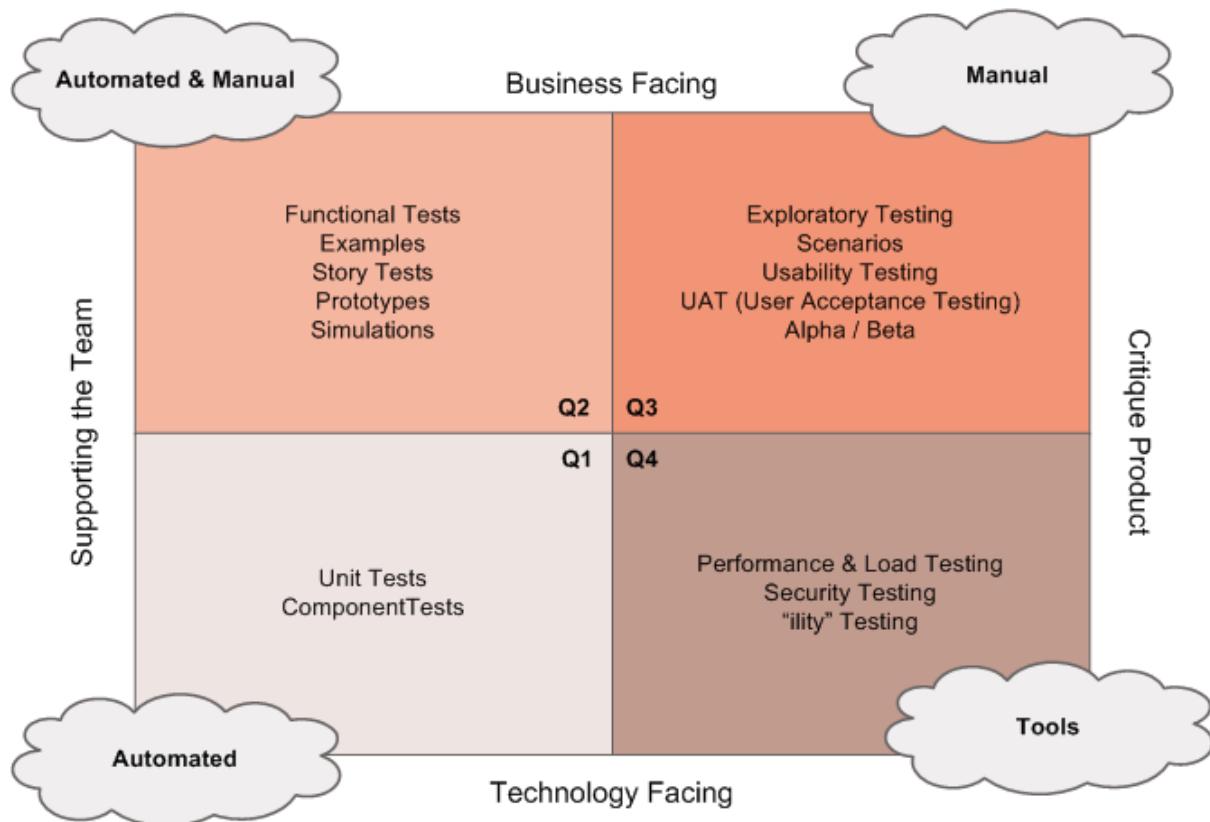
Til at bedre håndtere det strategiske mål omkring usability, vil der anvendes værktøjet Maze. Maze muliggør bedre analyse af brugerinteraktion med applikationen og hjælper med at visualisere dette data. Dette kan give bedre mulighed for at foretage løbende ændringer til designet (83).

## Den agile kvadrant

For at skabe en fælles og simplificeret forståelse for diverse tests på tværs af organisationens medarbejdere, er der blevet gjort brug af den *Agile Kvadrant* (84). Projektet er som før nævnt, med sine strategiske mål, vægtet både mod automatisering for at opnå en højere *Maintainability*, samt en mere manuel testing i henhold til den *Reactive* og *Exploratory* testing strategier, for at nå TTM (*Time to Market*). *Maintainability* stiller krav til at systemet er gennemtestet, og derfor vil Q1 og Q2 i den Agile Kvadrant være i fokus i en 40/20% sammensætning. Og de sidste 40% fokus ligger i den 3 kvadrant. Eftersom der også er fokus på princippet *Build Small, Release fast, Fail Small*, vil testerne ikke bruge for lang tid på den manuelle fremgangsmåde i henhold til *Reactive* og *Exploratory* strategi. Dette betyder at mængden af kode, testerne skal gennemgå ikke er overvældende for stadig at opnå kvalitet og TTM.



Agile Testing Quadrants



Billede 40. Agile kvadrant (85).

Grundet at systemet stadig befinder sig i de initierende faser vil der forekomme ændringer når det kommer til hvilke typer test der bliver fokuseret på. Grundet dette vil der senere i systemets levetid nødt til at blive lagt mere vægt på performance og load testing, som at systemet og kundebasen gror. Dette vil tage fokus over tiden væk fra usability testing (Q3) og mere over imod Q4 (86). Denne overgang skyldes blandt det strategiske mål omkring skalerbarhed og evnen til at håndtere 500 beskeder i minuttet.

## Diskussion

I forhold til web scraping, findes der 3. parts løsninger, man kan gøre brug af. Ingen vil dette være en mulig løsning, da man kan spare et udviklerteam, som ikke skal stå for implementeringen, vedligeholdelse og administration af nuværende løsning.

Dermed kan man have en administrator ansat i stedet, eller lade den indgå i DevOps teamet.

En web scraper såsom [webscraper.io](https://www.webscraper.io) (87), indeholder også automatiserede processer, hvor blandt andet notifikation via webhooks, kan foretages, sådan at man hele tiden er opdateret på hvornår en forespørgsel er færdig, eller hvis en forespørgsel skulle fejle.



Dertil kan den integreres med adskillige fil cloud storages, der kan samle alt information et sted.

Dette kan hjælpe med TTM.

*Derudover har Webscraper.io også en indbygget normalizer, som kan tage forskellige filtyper, og omdanne disse til en fælles format til systemet, hvilket også bidraget til scalability.*

Grundet relevansen af usability, som der er blevet refereret til i de strategiske mål. Så vil det være gavnligt for systemet og projektet på sigt at få en måde at indsamle mere bruger feedback uddover anvendelsen af MAZE. Dette kunne blandt andet være ekstra interviews med andre kulturinstitutioner. Her ville spørgsmålet om fokusområder være relevant, samt et kontinuerligt samarbejde i forhold til deres perspektiv omkring inklusionen af relevante teknologier. Det kunne også være meget relevant gennem monitorering at se hvordan brugerne anvender klienterne på sigt. Både for at sikre at fremgangsmåden giver mening, men også for at se hvilke funktionaliteter der er mest hyppigt anvendt.

Som en del af processen med at overveje arkitekturprincipper, var principippet om design to be monitored oppe og blive diskuteret. Det blev valgt at fjerne dette princip under udformningen af arkitektur principperne, da det ikke virkede muligt at prioritere den højere end en af de allerede valgte principper. Det blev heller ikke set som en mulighed at tilføje den som et ekstra princip, da det blev vurderet, som for mange principper, at håndtere på samme tidspunkt.

Dog senere i systemets levetid kan det overvejes at tage design to be monitored ind som et princip. Dette skyldes at efter systemet er blevet ordentligt etableret, vil det være muligt at fjerne fokusset lidt fra Asynchronous design, da den allerede er blevet en meget integreret del i systemet på det tidspunkt. Ved at gøre dette kan *Designed to be Monitored* inkluderes, i takt med at systemet går mere over på Q4 af den *agile kvadrant*.

Der skal overvejes om alle de udvalgte 3. parts løsninger, kan betale sig at gøre brug af versus, lade dem indgå som egne implementeringer i projektet.

Her skal det igen nævnes at nuværende design, hvor flere 3. parts løsninger (Og derfor fokusere på *Buy When None Core*) indgår i system designet, skyldes at nå TTM, så tidligt som muligt.

I ethvert projekt, er det ikke sikkert der er kapital nok til at gøre brug af alle disse løsninger, men det forventes at være billigere versus at have et udviklerteam der står for egen implementering, eller lade det indgå i et nuværende team.

Dertil er der også et vigtigt kriterie hvert udvikler team, skal kunne leve op til.



At være cloud udvikler kræver nogle flere kompetencer, da udvikling i den klassiske facon ikke skal tænke på den mere komplekse Micro Service mønster, eller *containers/orchestration* samt cloud konfigurering.

Derfor kræver det at hver udvikler team, har mindst en udvikler med disse kompetencer.

Der er også blevet lavet en revurdering af nuværende system design, hvorpå diverse gateways forventes at skulle gøre brug af en cache.

Grundet fravalget i systemdesignet, skyldes den komplekse, og meget specifikke lokationsdata, der bliver videresendt sammen med kort dataen. Dette blev dog senere fundet til kun at påvirke en begrænset mængde af kaldene, hvilket medgjorde at en cache på gateway, blev en sandsynlighed.

## Konklusion

I udarbejdelse af dette projekt, er diverse teknologier og design mønstre blevet undersøgt.

Nyere forskning viser at AR teknologi bliver mere populært i museumshusene, og er også en feature der bliver set på med stor entusiasme fra Kim Callesen.

Dette betyder også at 3D modeller, skal inddrages i projektet for at kunne gennemføre en AR applikation.

Her er der også udtænkt løsningen, hvorpå en service bliver udarbejdet, der skal tage imod 3D modeller der gemmes i en fælles database (3. parts, Sketchfab), og disse skal bruges til Augmented Reality (AR) platformen.

Der er også blevet introduceret flere pitfalls, man skal være opmærksomme på. Her er blandt andet verifikationen af hver 3D model, til at indgå i et formidlings miljø, og ikke vise ukorrekte data.

Dertil er kompleksiteten og viljen til at deltage, samt tiden det vil tage at få 3D modeller til landets fortidsminder, en længere process.

Udover dette er der blevet gjort brug af et nyere systemarkitektur, nemlig Cloud Native, der skal hjælpe med at skalere mere effektivt på Y og Z aksen.

I forløbet af dette projekt er der anvendt flere teknologier og design mønstre til at sikre at fremtidig skalering, bliver nemmere at håndtere. Til dette blev der dykket ned i blandt andet en Cloud løsning. Denne teknologi blev anvendt med diverse design mønstre og arkitekture, for at sikre nem skalering, uden stor påvirkning af regression. Her blev der blandt andet anvendt microservices, hvilket gjorde skaleringen på diverse akser nemmere. Derudover blev der også brugt Fault isolation, for at sikre at nye features kunne implementeres, uden effekt på det resterende program. Samt muliggøre høj tilgængelighed ved at sikre en fejl, ikke påvirker hele systemet. Herudover anvendes der



også et Normalizer mønster, som sikre at tilføjelsen af nye filtyper ikke vil være besværligt at implementere.

For at løse problemet, hvorpå at de ønskede data kan blive vist til brugerne, er der blevet gjort brug af eksterne databaser og API'er.

Disse databaser og API'er er enten statsejet, eller dele af et EU projekts, og standarden samt kvaliteten er derfor høj.

Staten og EU håndterer også de standarder der forventes at blive overholdt, for at opnå præcision og korrekthed af historiske data.

Derfor er det ikke en opgave dette projekt skal have ansvar for, men blot hente dataen, og videreforsmidle den.

Der er også blevet gjort brug af en web scraper for netop at simplificere og automatisere hentning af disse data, og organisationens udviklere skal derfor ikke stå med opgaven at implementere en lignende løsning.

For brugervenlighed er designere tænkt ind i de udvikler teams, der skal stå for klient applikationerne.

Og test af disse klient applikationer foretages med et automatiserings værktøj kaldet *Maze*, hvor kvaliteten af klient appen forbliver høj.

Formidling til en gængs bruger bliver foretaget gennem kort data, og information der er tillagt hver fortidsminde.

I fremtiden er der tiltænkt flere features, der skal forbedre formidlingsprocessen for slutbrugeren, nemlig oplæsning samt virtuel guide.

Til sidst skal det nævnes at den fornævnte web scraper, også kan hjælpe med inkludering af adskillige filformater, og dermed kunne informationssøgningen blive mere udspredt.

Der er også undersøgt nuværende teknologier, til at hjælpe med navigering og hvordan det er muligt at assistere med formidlingen. Hertil er der anvendelse af GIS teknologier, til bedst at identificere og hjælpe med at guide brugerne til de ønskede fortidsminder.

Herudover er der også dykket ned i muligheden for brugen af AR teknologi, til at fremme oplevelsen af fortidsminder minder, der er blevet medtaget gennem årene.



## Referencer

- (1) Malcolm Champion E. Virtual Heritage . Whitechapel Technology Centre 75 Whitechapel Road London E1 1DU: Ubiquity Press Ltd.; 2021.
- (2) Signe Lykke Littrup. Museet som hybridt medieret oplevelsesrum: Undersøgelse af udstillingers måde at medskabe museers relationer til gæster. Socio-materiel og teknologisk mediering i rumRoskilde Universitet; 2019.
- (3) Kulturarv. Available at: <https://slks.dk/omraader/kulturarv>. Accessed 15/04/22.
- (4) Slots- og kulturarvsstyrelsen. Fund og Fortidsminder . N/A; Available at: <https://www.kulturarv.dk/fundogfortidsminder/>. Accessed 15/04/22.
- (5) esri.com. What is GIS? . N/A; Available at: <https://www.esri.com/en-us/what-is-gis/overview>. Accessed 15/04/22.
- (6) Kulturministeriet Kulturarvsstyrelsen. Vejledning om pleje af fredede fortidsminder. Kulturarvsstyrelsen 2009 Juni.
- (7) PTC. What Is Augmented Reality. N/A; Available at: <https://www.ptc.com/en/technologies/augmented-reality>. Accessed 15/04/22.
- (8) Slots og Kulturstyrelsen. Slots- og kulturstyrelsen. Available at: <https://slks.dk/om-styrelsen>. Accessed 15-04-, 2022.
- (9) europeana. Our API's<br>. Available at: <https://pro.europeana.eu/page/apis>. Accessed 15/06/, 2022.
- (10) Kaur H. What is Web Scraping and How to Use It? 2021; Available at: <https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/>. Accessed 15/06/, 2022.
- (11) Suciu D. Create your own web scraper in c in just a few minutes. Available at: <https://medium.com/c-sharp-progarmming/create-your-own-web-scraper-in-c-in-just-a-few-minutes-c42649adda8>. Accessed 14-06-, 2022.
- (12) HTML Agillity Pack. HTML Agillity Pack. Available at: <https://html-agility-pack.net/>. Accessed 14-06-, 2022.
- (13) Gregor H, Woolf B. Message Transformation. Enterprise Integration Patterns 75 Arlington Street, Suite 300 Boston, MA 02116: Pearson Education; 2015. p. 291-316.
- (14) Vesthimmerlands. Medarbejdere. Available at: <https://www.vesthimmerlandsmuseum.dk/museum/aabningstid-og-kontakt/medarbejdere>. Accessed 15/06/, 2022.
- (15) L. Abbott M, T. Fisher M. The Art of Scaleability<br>. Pearson Education, Inc. Rights and Contracts Department 501 Boylston Street, Suite 900 Boston, MA 02116: Addison-Wesley; 2010.
- (16) Guldbrandsen M. MoSCoW metoden. Available at: <https://blivprojektleder.dk/moscow-metoden/>. Accessed 14-06-, 2022.
- (17) Dyson J. Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements. 2019; Available at: <https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson/>. Accessed 15/06/, 2022.
- (18) Gartstki K. Digital Innovations in European Archaeology. First ed. University Printing House, Cambridge CB2 8BS, United Kingdom: Cambridge University Press; 2020.



- (19) cidoc-crm. What is the CIDOC CRM? Available at: <https://www.cidoc-crm.org/>. Accessed 15/04/22.
- (20) intarch. The LEAP projects . Available at: <https://intarch.ac.uk/leap/>. Accessed 15/04/22.
- (21) pro.europeana. INTRODUCTION. Available at: <https://pro.europeana.eu/page/linked-open-data>. Accessed 15/04/22.
- (22) Microsoft. Introduction to cloud-native applications. 2022; Available at: <https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/introduction>. Accessed 15/06/, 2022.
- (23) Vettor R. Architecting Cloud-Native .NET Apps for Azure<br>. 1st ed. Redmond, Washington 98052-6399: Microsoft Developer Division, .NET, and Visual Studio product teams A division of Microsoft Corporation One Microsoft Way; 2022.
- (24) Vettor R. Microservices. 1st ed. Redmond, Washington 98052-6399: Microsoft Developer Division, .NET, and Visual Studio product teams A division of Microsoft Corporation One Microsoft Way; 2022.
- (25) Vettor R. Architecting Cloud-Native .NET Apps for azure<br>. 1st ed. Redmond, Washington 98052-6399: Microsoft Developer Division, .NET, and Visual Studio product teams A division of Microsoft Corporation One Microsoft Way; 2020.
- (26) Microsoft. Create a CI/CD pipeline for .NET with Azure DevOps Starter. 2021; Available at: <https://docs.microsoft.com/en-us/azure/devops-project/azure-devops-project-aspnet-core>. Accessed 15/06/, 2022.
- (27) Evans E. Domain-Driven Design<br>. Pearson Education, Inc. Rights and Contracts Department 75 Arlington Street, Suite 300: Pearson Education; 2004.
- (28) Kemal Erinç Y. The SOLID Principles of Object-Oriented Programming Explained in Plain English. 2020; Available at: <https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/>. Accessed 15/06/, 2022.
- (29) Vettor R. API Gateways<br>. Architecting Cloud Native NET Apps for Azure Redmond, Washington 98052-6399: Microsoft Developer Division, .NET, and Visual Studio product teams A division of Microsoft Corporation One Microsoft Way; 2022. p. 27-28.
- (30) Ferguson P. Architectural principle fault isolation and swim lanes. Available at: <https://akfpartners.com/growth-blog/fault-isolation-swim-lane>. Accessed 19-12-, 2021.
- (31) L. Abbott M, T. Fisher M. Chapter 12  
Exploring Architectural  
Principles. The Art of Scalability Pearson Education, Inc. Rights and Contracts Department 501 Boylston Street, Suite 900: Addison-Wesley; 2010. p. 195-209.
- (32) Abbott L. Martin, Fisher T. Michael. Monitoring Applications. The Art of Scaleability: Pearson Technology Group. p. 469-482.
- (33) Microsoft. What is monitoring<br>. 2021; Available at: <https://docs.microsoft.com/en-us/devops/operate/what-is-monitoring>. Accessed 15/06/, 2022.
- (34) L. Abbott M, T. Fisher M. The Art of Scalability<br>. ; 2010.
- (35) tech.gsa.gov. Conducting a Scrum of Scrums. Available at: [https://tech.gsa.gov/guides/conducting\\_scrum\\_of\\_scrums/](https://tech.gsa.gov/guides/conducting_scrum_of_scrums/). Accessed 21-10-, 2021.



- (36) Redbridge. What is Application Lifecycle Management <br>.
- (37) Katalon. Best 14 CI/CD tools you must know<br>. Available at:  
<https://katalon.com/resources-center/blog/ci-cd-tools>. Accessed 15-06-, 2022.
- (38) Nagios. Nagios. Available at: <https://www.nagios.com/>. Accessed 19-12-, 2021.
- (39) tutorialspoint.com. SDLC - Overview. Available at:  
[https://www.tutorialspoint.com/sdlc/sdlc\\_overview.htm](https://www.tutorialspoint.com/sdlc/sdlc_overview.htm). Accessed 15/06/, 2022.
- (40) Reeves S. A comprehensive guide to RACI/RASCI model. Available at:  
<https://www.goodcore.co.uk/blog/a-guide-to-the-raci-rasci-model/>. Accessed 19-12-, 2021.
- (41) avinetworks.com. Single Point of Failure Definition. Available at:  
<https://avinetworks.com/glossary/single-point-of-failure/>. Accessed 15/06/, 2022.
- (42) enterpriseintegrationpatterns. Fire-and-Forget. Available at:  
<https://www.enterpriseintegrationpatterns.com/patterns/conversation/FireAndForget.html>. Accessed 16/06/, 2022.
- (43) Ozkaya M. Service Aggregator Pattern<br>. Available at:  
<https://medium.com/design-microservices-architecture-with-patterns/service-aggregator-pattern-e87561a47ac6>. Accessed 15-06-, 2022.
- (44) Fowler M. Polyglot Persistence<br>. Available at:  
<https://martinfowler.com/bliki/PolyglotPersistence.html>. Accessed 15-06-, 2022.
- (45) Johansson L. what-is-amqp-and-why-is-it-used-in-rabbitmq. Available at:  
<https://www.cloudamqp.com/blog/what-is-amqp-and-why-is-it-used-in-rabbitmq.html>. Accessed 15-06-, 2022.
- (46) Hohpe G, Woolf B. 2. Integration Styles. Enterprise Integration Patterns 75 Arlington Street, Suite 300 Boston, MA 02116: Pearson Education; 2004. p. 63-74.
- (47) Best Message Queue (MQ) Software. Available at:  
<https://www.g2.com/categories/message-queue-mq>. Accessed 15/06/, 2022.
- (48) Gregor H, Woolf B. Messaging . Enterprise Integration Patterns 75 Arlington Street, Suite 300 Boston, MA 02116: Pearson Education; 2004. p. 72-74.
- (49) Watson M. Message Queues & You - 12 Reasons to use message queuing<br>. Available at: <https://stackify.com/message-queues-12-reasons/>. Accessed 15-06-, 2022.
- (50) Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns - Elements of Reusable Object-Oriented Software<br>. Pearson Education Corporate Sales Division 201 W. 103rd Street Indianapolis, IN 46290 (800) 428-5331: Addison-Wesley; 2009.
- (51) Ozkaya M. Event Sourcing patterns in microservice architectures<br>. Available at:  
<https://medium.com/design-microservices-architecture-with-patterns/event-sourcing-pattern-in-microservices-architectures-e72bf0fc9274>. Accessed 15-06-, 2022.
- (52) Bennett A. Introduction to CQRS<br>. Available at:  
<https://medium.com/microservicegeeks/introduction-to-cqrs-64f609544f4a>. Accessed 15-06-, 2022.
- (53) Gaitonde A. introduction to containers: the basics of containerization<br>. Available at:  
<https://medium.com/geekculture/introduction-to-containers-basics-of-containerization-bb60503df931>. Accessed 15-06-, 2022.
- (54) <https://kubernetes.io/>. Production-Grade Container Orchestration . Available at:  
<https://kubernetes.io/>. Accessed 19/12/21.
- (55) Katalon. Continous Delivery vs Continous Deployment: Where to draw the



line?<br>. Available at:

<https://medium.com/katalon-studio/continuous-delivery-vs-continuous-deployment-where-to-draw-the-line-3d220033dfb6>. Accessed 15-06-, 2022.

(56) learnz.org.nz. GPS and GIS Technology. Available at:

<https://learnz.org.nz/highcountry152/bg-standard-f/GPS-and-GIS-Technology>. Accessed 15/06/, 2022.

(57) Chan A. GIS and Heritage<br>. Available at:

<https://medium.com/digital-heritage/gis-and-heritage-aa5e9a009cc>. Accessed 15-06-, 2022.

(58) Chen H, Shih N. The Application of VR and AR in the Digital Preservation of Old City Cultural Elements. 2020.

(59) Mansoor U. Caching strategies and how to choose the right one<br>. Available at: <https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/>. Accessed 15-06-, 2022.

(60) Amazon. Caching patterns<br>. Available at:

<https://docs.aws.amazon.com/whitepapers/latest/database-caching-strategies-using-redis/caching-patterns.html>. Accessed 15-06-, 2022.

(61) Despot I. Graph Database vs Relational Database. Available at:

<https://memgraph.com/blog/graph-database-vs-relational-database>. Accessed 11-04-, 2022.

(62) aws.amazon. What is a graph database. Available at:

<https://aws.amazon.com/nosql/graph/>. Accessed 14-06-, 2022.

(63) neo4j. Modelling tips. Available at: <https://neo4j.com/developer/modeling-tips/>. Accessed 14-06-, 2022.

(64) Subramanyam VS. Basics of bounding boxes. Available at:

<https://medium.com/analytics-vidhya/basics-of-bounding-boxes-94e583b5e16c>. Accessed 14-06-, 2022.

(65) neo4j. Spatial. Available at:

<https://neo4j.com/docs/cypher-manual/current/syntax/spatial/>. Accessed 14-06-, 2022.

(66) Neo4j. Indexes for search performance<br>. Available at:

<https://neo4j.com/docs/cypher-manual/current/indexes-for-search-performance/>. Accessed 15-06-, 2022.

(67) neo4j. DB-hits. Available at:

<https://neo4j.com/docs/cypher-manual/current/execution-plans/db-hits/>. Accessed 15-06-, 2022.

(68) Usercentrics. What are cookies? 2019; Available at:

<https://usercentrics.com/knowledge-hub/what-are-cookies/>. Accessed 15/06/, 2022.

(69) owasp. JSON Web Token Cheat Sheet for Java. 2021; Available at:

[https://cheatsheetseries.owasp.org/cheatsheets/JSON\\_Web\\_Token\\_for\\_Java\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html). Accessed 22/04/22.

(70) owasp. Cross Site Scripting Prevention Cheat Sheet. 2021; Available at:

[https://cheatsheetseries.owasp.org/cheatsheets/Cross\\_Site\\_Scripting\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html). Accessed 22/04/22.

(71) jwt.io. Libraries for Token Signing/Verification . N/A; Available at:

<https://jwt.io/libraries>. Accessed 22/04/22.

(72) Dorothy G, Black R, van Veenendaal E. Test strategy and test approach. Foundations



of Software Testing Cheriton House, North Way, Andover, Hampshire, SP10 5BE, United Kingdom: Cengage Learning, EMEA; 2020. p. 164-167.

(73) Nagios. Nagios Log Server. Available at:

<https://www.nagios.com/products/nagios-log-server/>. Accessed 14-06-, 2022.

(74) Jira Software<br>. Available at:

[https://www.atlassian.com/software/jira?&aceid=&adposition=&adgroup=95003656689&campaign=9124878870&creative=513481357090&device=c&keyword=jira&matchtype=e&network=g&placement=&ds\\_kids=p51242194730&ds\\_e=GOOGLE&ds\\_eid=700000001558501&ds\\_e1=GOOGLE&gclid=CjwKCAjwxZqSBhAHEiwASr9n9Cx9xFpQpPSZXtHsvNHg7T62s-Jb4DfGEXhzqkET\\_dj\\_oAdVUI-7MxoCPZQQAvD\\_BwE&gclsrc=aw.ds](https://www.atlassian.com/software/jira?&aceid=&adposition=&adgroup=95003656689&campaign=9124878870&creative=513481357090&device=c&keyword=jira&matchtype=e&network=g&placement=&ds_kids=p51242194730&ds_e=GOOGLE&ds_eid=700000001558501&ds_e1=GOOGLE&gclid=CjwKCAjwxZqSBhAHEiwASr9n9Cx9xFpQpPSZXtHsvNHg7T62s-Jb4DfGEXhzqkET_dj_oAdVUI-7MxoCPZQQAvD_BwE&gclsrc=aw.ds). Accessed 25/3.

(75) Nanda V. Defect management process in software. Available at:

<https://www.tutorialspoint.com/defect-management-process-in-software-testing>.

Accessed 14-06-, 2022.

(76) Barjis J. Definition of Ready & Definition of Done PART 1: Concepts and Foundation Definition.

(77) Kanjilal J. MOQ development in C#. Available at:

<https://www.infoworld.com/article/3264438/how-to-use-moq-to-ease-unit-testing-in-c.html>. Accessed 24/05/21.

(78) Graham D, Black R, van Veenendaal E. Black-box test techniques. Foundations of Software Testing Cheriton House, North Way, Andover, Hampshire, SP10 5BE, United Kingdom: Cengage Learning, EMEA; 2020. p. 112-132.

(79) Dorothy G, Rex B, Erik van Veenendaal. White-box test techniques. Foundations of Software Testing. Fourth ed. Cheriton House, North Way, Andover, Hampshire, SP10 5BE, United Kingdom: Cengage Learning, EMEA; 2020. p. 132-140.

(80) Khorikov V. DRY and DAMP. 2020; Available at:

<https://enterprisecraftsmanship.com/posts/dry-damp-unit-tests/>. Accessed 23/05.

(81) Hamilton T. Test Driven Development. Available at:

<https://www.guru99.com/test-driven-development.html#3>.

(82) Hamilton T. Equivalence partitioning & boundary value analysis. Available at:

<https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html#:~:text=Boundary%20testing%20is%20the%20process,is%20called%20%E2%80%9Cboundary%20testing%E2%80%9D>. Accessed 14-06-, 2022.

(83) Maze. Why Maze<br>. Available at: <https://maze.co/why-maze/>. Accessed 15-06-, 2022.

(84) Cogniti. Test automation og Agile test quadrants<br>. Available at:

<https://www.cogniti.com/blog/agile-test-automation-and-agile-quadrants/>. Accessed 15-06-, 2022.

(85) Crispin L. Using the agile quadrants<br>. Available at:

<https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/>. Accessed 15-06-, 2022.

(86) Tutorialspoint. Agile testing quadrants<br>. Available at:

[https://www.tutorialspoint.com/agile\\_testing/agile\\_testing\\_quadrants.htm](https://www.tutorialspoint.com/agile_testing/agile_testing_quadrants.htm). Accessed 15-06-, 2022.

(87) webscraper.io. Making web data extraction easy and accessible for everyone.

Available at: <https://webscraper.io/>. Accessed 16/06/, 2022.