MUSIT Ny IT-arkitektur, planleggingsfase: Utredninger av teknologivalg
Møte i MUSIT styre 19. februar 2016

Innledning

Prosjekt MUSIT ny IT-arkitektur har jobbet med utredning av teknologivalg (plattform, språk, database, søkeindeks) og lisensiering av prosjektet (åpen eller lukket kodebase) siden oppstart av planleggingsfasen i november 2015. I henhold til mandat for planleggingsfasen skal teknologivalgene inngå som del av rapport som inneholder en oppdatert beskrivelse for IT-arkitekturprosjektet, prosjektmandat og prosjektplan for hovedprosjektet.

For å komme videre i arbeidet med planlegging og estimering av pilot og hovedprosjekt, har prosjektet behov for delbeslutninger angående valg av plattform, språk, database, søkeindeks og lisensiering av prosjektet. Styringsgruppen ber derfor styret om tilslutning til anbefalingene gitt av denne utredningen.

Teknologiutredningene er behandlet i styringsgruppemøte 25. januar 2016 og via e-post.

Sammendrag og anbefalinger

Vurdering av språkvalg

Plattform (operativsystem)

Det har blitt vurderte to plattformer; Windows og Linux. USIT drifter begge plattformene og begge vil fungere veldig bra for MUSIT. USIT benytter Windowsplattformen i hovedsak til drift av hyllevare og klientnettverket. Ved valg av Windows må MUSITs automasjonsbehov bygges opp fra grunnen av for å kunne understøtte sikker drift og microservices arkitektur.

Linux som plattform har meget gode verktøy for automasjon, noe som gjør det enkelt for MUSIT å få det som trengs for å kunne understøtte microservice arkitektur. Videre er Linux som plattform ønskelig siden USIT allerede har mye automasjon på plass for denne plattformen til webløsninger.

Økosystem

Med økosystemer refereres det til grunnplattformen til språkene og ikke selve språkene. Det finnes flere økosystemer som brukes i IT-bransjen. Av disse støttes både .Net og Java godt av USITs driftsorganisasjon.

På Linux plattformen er Java økosystemet både godt utprøvd og det klart sterkeste av de vurderte økosystemer, både på USIT og i andre driftsmiljøer i Norge og utenlands.

På Windows er .Net økosystemet det mest utprøvde, og det som anbefales av leverandøren (Microsoft).

Språk

Anbefalingen av et hybrid funksjonelt språk kommer ut fra å kunne bruke styrkene til både funksjonelle og objektorienterte språk for MUSIT.

I Java økosystemet er det Scala som har best utbredelse av de hybride funksjonelle språkene. Det har høy score på RedMonk skalaen, og webseksjonen ved USIT har kompetanse og erfaring med språket og det er god driftsstøtte og support fra Oracle og Java miljøet. Språket har en meget god tredjeparts støtte og bredt valg av serverteknologier og leverandører rundt økosystemet. USITs utviklere ved DS har liten eller ingen kompetanse på Java økosystemet og verktøyene, som betyr at det vil ta lengre tid å komme i gang med utviklingsarbeidet.

I .Net økosystemet er det F# som er mest utbredt av de hybride funksjonelle språkene. USITs utviklere ved DS har noe kompetanse og erfaring med dette språket og .Net har god support fra Microsoft og Microsoft miljøet. Men USIT mangler erfaring på webdrift, spesielt ved eksponering av tjenester ut av UiO. I tillegg mangler utprøvd infrastruktur rundt USITs skallsikring og sikkerhetssystemer for web.

Konklusjon/anbefaling

Styringsgruppen anbefaler at det velges et Java økosystem på Linux plattform (operativsystem) med Scala som språk for mellomvare.

Vurdering av databasevalg

I dagens løsning brukes relasjonsdatabase som lagringsplattform for MUSIT. I forbindelse med utredningen av teknologivalg som pågår har triplestore blitt lansert som et alternativ til relasjonsdatabse. Relasjonsdatabase og triplestore har to meget forskjellige strukturer og krever forskjellige prosesser for implementasjon.

Alternativ 1 videreføring av Oracle relasjonsdatabase, men med ny felles datamodell og migrering av data fra dagens databaser:

- USIT har god kompetanse på drift av Oracle relasjonsdatabase.
- Oracle har meget gode verktøy for utvikling og feilsøking som forenkler hverdagen for utviklere.
- Mulighet for gode integritetssjekker på dataene som ligger i databasen, gitt at dette modelleres inn i relasjonsmodellene ved implementering.
- Vil kunne komme raskt i gang med utviklingen.
- Meget komplekse datamodeller for å støtte opp rundt de avanserte datakrav museene har. Kan være tungt å jobbe med endringer på datastrukturen.
- Kompliserte og omfattende spørringer for å hente ut data på grunn av avanserte datastrukturer og datakrav.

Alternativ 2 bytte ut dagens relasjonsdatabase med en semantisk database (triplestore):

- Mindre kompleks datastruktur enn relasjonsdatabase, siden koblingskravene kan beskrives som en del av støtten som er innebygd i databasen.

- Medfører mindre kompleksitet i spørringer som skal gjøres mot databasen, enn relasjonsdatabase.
- Prosjektet vil ha raskere hastighet på utviklingen enn alternativ 1, siden spørringene er mindre komplekse
- USIT har ingen erfaring med drift av denne typen database.
- Noe begrenset tilgang på kompetanse i markedet, dette er en teknologi som nylig har begynt å bli utstrakt brukt.
- Enklere og mindre utviklede utviklingsverktøy enn relasjonsdatabasene.
- Datamodellering og oppstart av utvikling vil ta lengre tid enn ved relasjonsdatabase.

Generelt vil begge alternativene her ha omtrentlig samme kostnad for utviklingen. Men i forvaltning antas det at triplestore vil være billigere enn relasjonsdatabase.

Utfra prosjektets rammer og antatt risiko har prosjektet valgt å ikke anbefale triplestore for gjennomføringen av prosjektet.

Konklusjon/anbefaling

Styringsgruppen anbefaler alternativ 1 som er en relasjonsdatabase med Oracle i bunn, med en ny felles datamodell og migrering av data fra dagens databaser.

Vurdering av søkeindeks

Søkeindeks er viktig for å kunne lage en god søkefunksjon som kan brukes på tvers av fag og eksterne datakilder. Søkeindeksen vil være med på å sikre at eventuelle svakheter i aggregeringene, som kan ligge i en database, ikke vil påvirke fagsiden i sitt daglige bruk av MUSIT.

Det er i hovedsak to produkter som peker seg ut i dagens marked. Dette er:

- Elastic Search
- Solr

Solr er på mange måter et likeverdig produkt som Elastic Search, men det mangler aggregeringsstøtte.

Konklusjon/anbefaling

Elastic Search anbefales som søkeindeks fordi den har full støtte for aggregering. Elastic Search vil sørge for at MUSIT får enda bedre aggregeringsmuligheter enn det som finnes i dagens database.

Vurdering av lisensiering av prosjektet

Fordelen for MUSIT, med å gå for en åpen modell, er at eventuell samhandling med andre blir veldig enkelt, og interessenter fra andre museer rundt om i verdenen kan bidra med felles utviklingsinnsats.

Lukket kildekode er tradisjonelt brukt mye for produkter og skreddersøm som ikke er ment å deles med andre

Konklusjon/anbefaling

Det anbefales åpen kildekode, og at det i samarbeid med USITs jurist velges lisens som er mest mulig åpen.

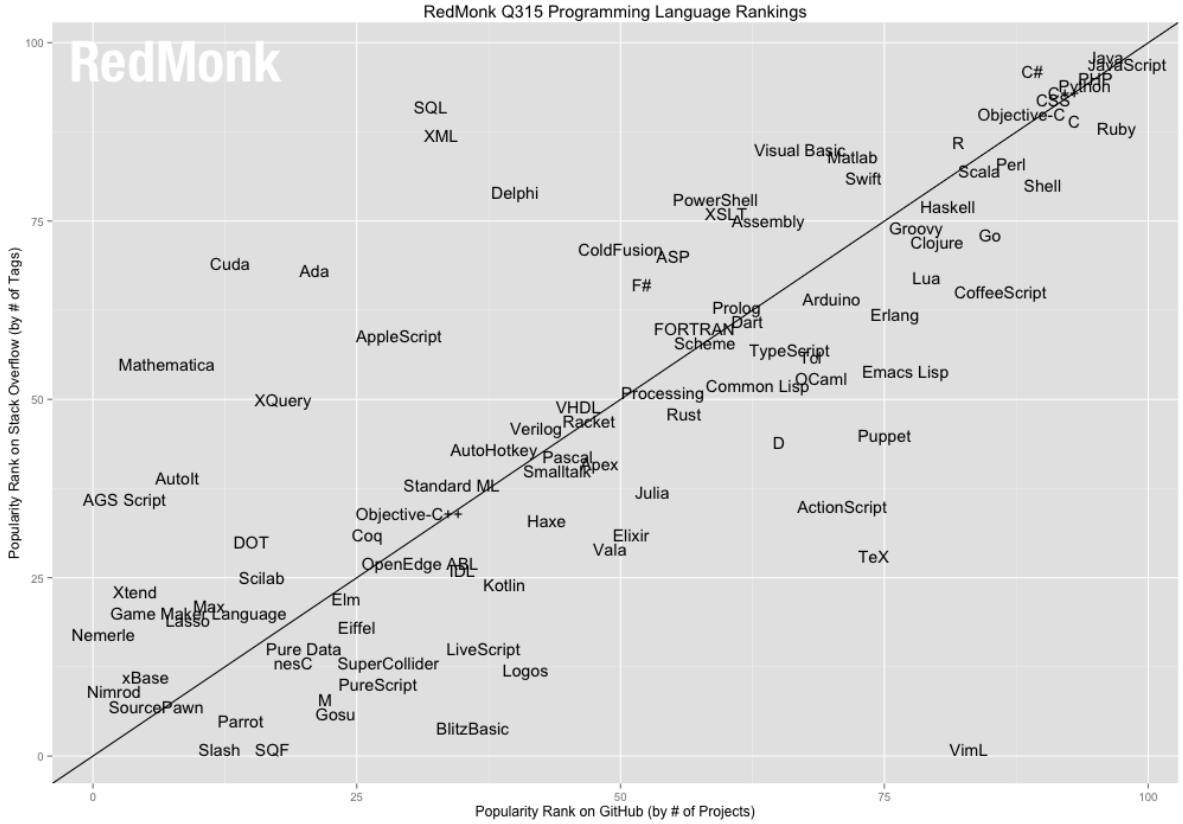
Innholdsfortegnelse

Språkutredning

Popularitet / Utbredelse

Det finnes flere måter å måle popularitet og utbredelse av et programmeringsspråk. Den som er benyttet her baserer seg på RedMonk sin årlige analyse av det mest utbredte programmeringsdiskusjonsforumet i verden (Stack Overflow) og den mest populære cloud tjenesten av prosjekt hosting i verden (GitHub). Denne analysen tar statistikk fra begge tjenestene og plotter språk på to akser for å vise analysereferanser over tid.

Slike analyser kan bare sees på som en indikasjon. Følgende link viser analyse for 2015 i detalj: http://redmonk.com/sogrady/2015/07/01/language-rankings-6-15/



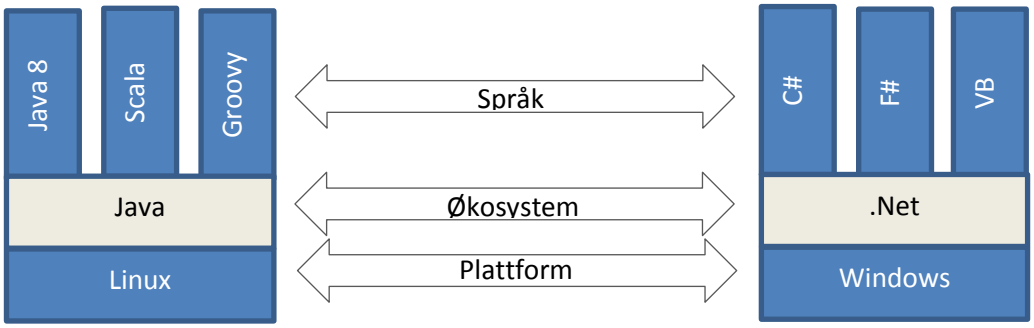
RedMonk har laget et rangeringssystem hvor de lister det 20 mest populære språk med tanke på kombinasjonen mellom communitystøtte og tilgjengelige aktive prosjekter. Denne kombinasjonen gir en indikasjon på hvor lett det er å få innspill for noen som jobber med, eller starter med, et språk. Den gir også indikasjon på tilgjengelig kompetanse. Rangeringen går fra 1 (best) til 20 (dårligst).

Økosystemer (språkplattform)

Det å velge et programmeringsspråk er mer omfattende enn bare å se på selve språket. Språkene er bare regelsett som kjøres på toppen av et økosystem, som veldig ofte støtter mange språk.

Med økosystemer refereres det til grunnplattformen til språkene og ikke selve språkene. Det finnes flere økosystemer som brukes i IT-bransjen. Av disse er det to som støttes godt av USITs driftsorganisasjon, .Net og Java.

Under er leverandørenes (Oracle og Microsoft) anbefalte modeller av Java og .Net :



Legg merke til at det er mange forskjellige språk (dette er bare et lite utvalg av hva som finnes i begge økosystemene). De fleste språkene i et økosystem kan bruke produkter og ofte biblioteker på kryss av språk. Dette kan også være noe forvirrende ettersom Java både er navn på et økosystem og ett av språkene (noe figuren over viser).

Økosystemet må ha god driftsstøtte og automatiseringsstøtte. Derfor bør valget av språk gjøres i etterkant av valget av økosystem. Språket bør velges ut fra hva det skal brukes til, slik at det gir de nødvendige verktøy utviklerne trenger for å løse oppgaven.

.Net økosystemet

.Net kjører i dag best på Windows plattformen, men det finnes alternativer for å kjøre det på Linux som vil bli forbedret ytterligere i løpet av 2016. Microsoft lanserer offisiell .Net støtte for Linux i 2016.

Fordeler:

- God driftsstøtte på USIT.
- God support fra Microsoft og Microsoft miljøet.
- Gode verktøy, og har begynt å få gode tredje parts alternativer.
- God tilgang på open-source biblioteker for gjenbrukbar funksjonalitet.
- USITs utviklere ved DS kjenner godt til økosystemet fra tidligere applikasjoner.
- Vil komme raskt i gang med utviklingen.
- Kan være en mulighet for gjenbruk av kode fra dagens system.

Ulemper:

- Mangler erfaring på webdrift i USIT (gruppen som har drift av applikasjoner på Windows, drifter ikke webløsninger i dag), spesielt ved eksponering av tjenester ut av UiO.
- Mangler mye automatisering og forutsetter utredninger for hvordan disse skal driftes og provisjoneres ved web eksponering.
- Mangler utprøvd infrastruktur med mer rundt USITs skallsikring og sikkerhetssystemer for web. Blir mest brukt til server installasjoner og tykke klienter i USIT i dag.
- Liten eller ingen mulighet for kompetansedeling med resten av USIT.
- Gjenbruk av kode fra dagens system kan bli tungt og kostbart å forvalte på grunn av kompleks kode (jamfør dagens situasjon) og ny datastruktur i ny arkitektur.

Java økosystemet

Java kjører i dag best på Linux plattformen, men det finnes godt støttede alternativer for å kjøre det på Windows.

Fordeler:

- God driftsstøtte på USIT.
- God support fra Oracle og Java miljøet.
- Gode verktøy og mange gode tredjeparts valgmuligheter.
- Mye erfaring i USIT webdrift. (Webseksjonen ved USIT bruker Java og Scala som språk og har gjort mye av det MUSIT IT-prosjektet planlegger å gjøre innenfor automasjon. Webdrift har også lagt til rette for noen av MUSITs behov innen automatisering rundt dette økosystemet fra før).
- Meget god tredjeparts støtte og bredt valg av serverteknologier og leverandører rundt økosystemet, unngår at man blir sterkt knyttet til en leverandør.
- Meget god open-source tilgang på biblioteker for gjenbrukbar funksjonalitet.
- Tilgang på internressurser, kunnskapsutveksling og gjenbrukbare komponenter fra webseksjonen på USIT.
- En av to prefererte serverside teknologier for webløsninger på USIT (jamfør forprosjektrapporten).
- God kompetansetilgang fra øvrige utviklingsmiljøer ved USIT og NTNU.

Ulemper:

- USITs utviklere ved DS har liten eller ingen kompetanse på Java økosystemet og verktøyene, som betyr at det vil ta lengre tid å komme i gang med utviklingsarbeidet.
- Stor valgfrihet i gratis biblioteker, dette kan skape forvirring for utviklere som ikke kjenner økosystemet (dette blir en fordel etter hvert som man får mer erfaring).
- Lite mulighet for gjenbruk av moduler fra dagens system.
- Avhengig av ekstern techlead, noe som vil medføre høyere kostnad.

Aktuelle språk

Når det gjelder språkvalg, kan en i utgangspunktet skille mellom objektorienterte eller funksjonelle språk.

Objektorienterte språk er meget gode til å håndtere livssykluser i kildekoden, og prosjektet vil ha forskjellige livssykluser de må håndtere for det som skal implementeres. Eksempel på elementer som har livssykluser: microservicer, database, webtjenester, med mer. Både Java 8 (Java-plattformen) og C# (.net-plattformen) er trygge alternativer når det gjelder objektorienterte språk. Begge har stor utbredelse i markedet, og er stabiliserte språk (delvis det vil ikke komme større endringer som vil få konsekvenser for en fremtidig MUSIT-portefølje). Utfordringen er at begge språkene er objektorienterte, og dermed «utdaterte» med hensyn til trendene mot funksjonell programmering.

Funksjonelle språk er gode til å implementere tunge algoritmer og er mindre «kodekrevende» enn objektorienterte språk. Trendene i dag går mot funksjonelle språk, bl.a. fordi dette gir mindre kode for samme problemstilling enn det objektorienterte språk gjør. Dette minsker igjen risikoen for feil i koden.

«Hybride» funksjonelle språk med objektorientert støtte for interoperabilitet med valgt økosystem, gjør at utviklerne har verktøyene til å bruke det beste fra begge verdenene ved utvikling av mellomvaren, som igjen vil spare implementasjonstid og gi mindre kompleksitet i kildekoden.

Under listes de to beste representantene for objektorienterte og hybride funksjonelle språk som støttes av de to vurderte økosystemene[1].

C#

Type språk: Moderne objektorientert
Økosystem: .Net
RedMonk rangering: 5
Anbefalt operativsystem: Windows Server
Veldig trygt språk å satse på, full backing fra leverandøren av økosystemet og enorm adopsjon i markedet. Eneste utfordring er at dette er et objektorientert språk som begynner å bli "utdatert" sammenlignet med trendene mot funksjonell programmering. Dette språket er stabilisert så det vil ikke skje store endringer som vil få implikasjoner for MUSITs fremtidige portefølje.

Java 8

Type språk: Moderne objektorientert
Økosystem: Java
RedMonk rangering: 2
Anbefalt operativsystem: Linux
Veldig trygt språk å satse på, full backing fra leverandøren av økosystemet og enorm adopsjon i markedet. Eneste utfordring er at dette er et objektorientert språk som begynner å bli "utdatert" sammenlignet med trendene mot funksjonell programmering. Dette språket er stabilisert så det vil ikke skje store endringer som vil få implikasjoner for MUSITs fremtidige portefølje.

F#

Type språk: Funksjonelt språk med objektorientert støtte for interoperabilitet med økosystemet.
Økosystem: .Net
RedMonk rangering: litt over 30
Anbefalt operativsystem: Windows Server
Trygt språk å satse på, full backing fra leverandøren av økosystemet, noe mindre utbredelse i markedet enn andre språk, så det kan være noe mer utfordrende å finne kompetanse ved behov. Språket begynner å bli modent og mye av stabiliseringen er gjort slik at en del av de gamle problemstillingene rundt full kjernebibliotekstøtte i økosystemet er helt borte. Det finnes fortsatt noen utfordringer, i spesielle tilfeller, med å bruke biblioteker fra andre språk på kryss av av økosystemet. Noen av de mindre brukte rammeverkene som er tilrettelagt C# er ikke ved nåværende tidspunkt klargjort for F#. Dette forventes å bli løst på sikt når språket oppnår større utbredelse i markedet.

[1] Python er ikke tatt med i vurderingen (jamfør forprosjektrapporten) grunnet mindre moden utviklingsverktøystøtte for server sentrerte teknologier, som dermed kan være til hinder for forvaltningsprosessen (spesielt ved refactoring), noe som er meget viktig for langtidslevende systemer.

Scala

Type språk: Funksjonelt språk med objektorientert støtte for interoperabilitet med økosystemet.
Økosystem: Java
RedMonk rangering: 14
Anbefalt operativsystem: Linux
Trygt språk å satse på. Firmaet Typesafe styrer mye av utviklingen til språket og de satser hardt på sin portefølje med blant annet Scala, Akka og Playframework. De har klart å skape seg en sentral posisjon i «Internet of Things» verdenen for å løse særdeles utfordrende problemstillinger rundt store datasett, og er ledende innenfor global standardisering på teknologier rundt dette. Språket har også stabilisert seg så gamle problemstillinger med språkjusteringer og manglende dokumentasjon er helt borte. Dette språket har dermed blitt trygt å bruke på systemer som skal ha lang varighet og forvaltes videre i etterkant.

Vurdering av språkene

Ettersom hybride funksjonelle språk gir tilgang til å bruke det beste fra både den objektorienterte og funksjonelle verdenen, vil utviklingen av mellomvaren bli mindre kompleks, noe som vil spare implementasjonstid. I henhold til dette anbefales det å velge hybride, funksjonelle språk i den fremtidige IT-infrastrukturen i MUSIT.

Plattform (operativsystem)

USIT drifter to plattformer, Linux og Windows. Begge plattformene vil fungere veldig bra for MUSIT. Det som ofte er drivende for om man velger det ene eller det andre er hva man skal kjøre på plattformen. Eksempelvis anbefaler Microsoft å kjøre .Net på Windows mens Oracle anbefaler å kjøre Java på Linux-plattformen.

USIT har skilt bruken av disse plattformene, hvor Linux hovedsakelig har blitt brukt for drift av webtjenester. Webseksjonen har jobbet aktivt med webdrift for å utforme en god modell for en del av prosessautomasjonen MUSIT vil ha behov for, samt eksponering av slike tjenester på vegne av UiO. Windows benyttes i hovedsak til drift av hyllevare og klientnettverket. Ved valg av Windowsplattformen må MUSITs automasjonsbehov bygges opp fra grunnen av.

Utfra uttalelser fra både sikkerhetssjefen og webdrift ved USIT, og i et felles møte mellom MUSIT IT-prosjektet og ulike aktører ved USIT, vil Linux og Java være et tryggere valg enn Windows med tanke på automatisering og sikkerhet for eksponering av komplekse webtjenester. Hovedgrunnen er at driftsavdelingen har god kompetanse på drift av webapplikasjoner på Java og vet hvordan disse skal forvaltes, driftes og automatiseres. Dette betyr at USIT vil ha større kompetansespenn til å levere de tjenestene MUSIT har behov for enn i dag.

Risikovurdering

Det er ikke foretatt risikovurdering av objektorienterte språk, da anbefalingen uansett peker på hybride språk. Dette baseres på at fordelene med mindre og mere rasjonell kode innenfor hybride språk oppveier for både utbredelse og stabilisering innen tradisjonelle, objektorienterte språk.

Videre er det med hensyn til plattformvalg identifisert at USITs utviklere ved DS har eksisterende kompetanse på .Net-plattformen, men at Java-plattformen har større utbredelse ved USIT, og således gir en bedre kontaktflate og kompetansedeling.

De to hovedalternativene som da utkrystalliseres er F# på .Net-plattform og Scala på Java-plattformen.

Under er en tabell som viser usikkerhetene med en vurdering av sannsynlighet og konsekvens. Sannsynlighet multiplisert med konsekvens vil gi en risikofaktor som brukes som en del av vurderingen av alternativene.

Sannsynlighet rageres fra «veldig lav», «lav», «moderat», «høy» og «veldig høy». Konsekvenser rangeres fra «ubetydelig», «liten», «moderat», alvorlig» og «veldig alvorlig».

Valg av Java-plattform med Scala	**Sann-synlighet**	**Kon-sekvens**	**Risiko**	**Kommentar/tiltak**
Liten eller ingen kompetanse i DS-gruppa, vil føre til at oppstart og utvikling tar lengre tid.	Høy 4	Moderat 3	12	Prosjektet kan leie inn ekstern techlead og systemarkitekt.
Mye nytt å lære seg for DS-gruppa, kan følges tungt og gi lavere motivasjon i starten.	Moderat 3	Liten 2	6	Benytte parprogrammering, techlead og systemarkitekt kan hjelpe til.
Lite mulighet for gjenbruk, medfører at mer funksjonalitet må implementeres på nytt.	Lav 2	Liten 2	4	
Enorm valgfrihet, dette kan skape forvirring for utviklere som ikke kjenner økosystemet (dette blir en fordel etter hvert som man får mer erfaring).	Lav 2	Alvorlig 4	8	Støtte fra techlead og systemarkitekt, benytte dokumenterte best practice.
Avhengig av ekstern techlead, noe som vil medføre høyere kostnad.	Høy 4	Moderat 3	12	Prosjektet kan forsøke å få leid techlead fra USIT.

Valg av .Net-plattform med F#	**Sann-synlighet**	**Kon-sekvens**	**Risiko**	**Kommentar/tiltak**
Liten eller ingen kompetansedeling med resten av USIT, DS-gruppa vil fortsatt være isolert og MUSIT vil ikke ha stordriftsfordel med støtte fra andre ressurser ved USIT.	Veldig høy 5	Alvorlig 4	20	
Lite erfaring med webdrift hos USIT på denne plattformen, medfører at noe kompetanse må bygges opp.	Høy 4	Moderat 3	12	
Gjenbruk av moduler fra dagens system kan bli tungt å forvalte på grunn av kompleks kode (jamfør dagens situasjon).	Moderat 3	Moderat 3	9	Usikkert om det vil være mulig med gjenbruk, jamfør overgang fra tolags- til trelagsarkitektur.
Lite automasjon tilgjengelig for Windows plattformen, gjør at all	Veldig høy	Liten 2	10	Prosjektet må være drivende for automatiseringsprosesser

automasjon må bygges fra starten for å kunne understøtte sikker drift og microservices.	5			på Windowsplattformen.
Avhengig av ekstern techlead, noe som vil medføre høyere kostnad.	Moderat 3	Moderat 3	9	

Oppsummering og anbefaling

Det anbefales følgende valg av plattform/operativsystem, økosystem/språkplattform og utviklingsspråk:

- **Linux** anbefales som plattform, siden Linux har meget gode verktøy for automasjon, noe som gjør det enkelt for MUSIT å få det som trengs for å kunne understøtte microservice arkitektur. Videre har USIT allerede mye automasjon på plass for denne plattformen til webløsninger.
- **Java** anbefales som økosystem, da det er godt utprøvde og klart det mest utbredte, både i USIT og i andre driftsmiljøer i Norge og utenlands.
- **Scala** anbefales som utviklingsspråk, fordi dette er det hybride, funksjonelle språket med størst utbredelse, med en høy score på RedMonk-skalaen. I tillegg har blant annet webseksjonen ved USIT kompetanse og erfaring med språket, noe som øker muligheten for kompetansedeling.

Konsekvensen av disse valgene vil blant annet være at en bruker lengre tid i oppstartsfasen av prosjektet, fordi en skal ta i bruk et annet økosystem og programmeringsspråk enn det som har vært benyttet av USITs utviklere ved DS tidligere. På sikt vil imidlertid dette bety at en får tilgang på et bredere kompetansemiljø, og en bedre forankring av både utviklingspraksis og teknologivalg.

Dersom det skulle oppstå spesielle behov eller utfordringer hvor det ikke er hensiktsmessig å bruke valgt utviklingsspråk, må valg av alternative språk/scriptspråk begrunnes spesielt, og godkjennes av MUSIT via styringsgruppe eller styret. Dette for å sikre en entydig infrastruktur som er enkel å vedlikeholde/forvalte i etterkant. I tillegg må eventuelt alternative språk/scriptspråk kunne kjøre på valgt økosystem, slik at biblioteker og kildekode skal kunne være gjenbrukbare i det gjeldende utviklingsspråket.

Ved bruk av scriptspråk bør det ha full JSR223 støtte, dermed sikres det at man har muligheten til å bruke all programkode fritt på kryss av språkene, og har full tilgang fra hovedspråket.
En god referanse for dette er: https://en.wikipedia.org/wiki/List_of_JVM_languages.

Databaseutredning

Innledning

I dagens løsning brukes relasjonsdatabase som lagringsplattform for MUSIT. I forbindelse med utredningen av teknologivalg som pågår har triplestore blitt lansert som et alternativ til relasjonsdatabse. Ettersom triplestorebegrepet er ganske nytt i MUSIT gis det en utvidet beskrivelse av hva dette er og hva det vil si for MUSIT i vedlagt ordliste.

Relasjonsdatabase og triplestore har to meget forskjellige strukturer og krever forskjellige prosesser for implementasjon. Grunnstrukturen på dataene, teknikkene som brukes i mellomvaren og måten man gjør uttrekk på dataene vil ha store variasjoner for de to lagringsteknologiene. Dette betyr at det i praksis kun vil være brukergrensesnittet (frontend) som kan gjenbrukes direkte ved et eventuelt skifte av databaseteknologi underveis. Det anbefales derfor at databaseteknologi og metode som blir valgt for piloten videreføres for resten av prosjektet.

Arkitekturdrivere

Overgangen fra dagens server-klient arkitektur til trelagsarkitektur med web klient, gi store endringer i arkitektur og infrastruktur i den nye MUSIT-porteføljen.

Prosjektet har valgt å bruke microservices for mellomvaren. Denne arkitekturen er godt tilrettelagt for å skape en helhetlig infrastruktur, og som er enkel å oppdatere modul for modul i forvaltning. Arkitekturen støtter også godt opp rundt arkitekturprinsippene til DIFI.

For å kunne forvalte alle museer og fagområder enhetlig må det bygges en applikasjon som har felles funksjonalitet på kryss av alle fagområder, og som behandler data likt for hver type data. Den nye arkitekturen drives av sentralisering og sammenstilling. Så mye som mulig av database og forretningslogikk må være felles for alle fagsidene for å gjøre forvaltning og utvikling entydig og felles.

Beskrivelse av alternativene

Det er i utredningsarbeidet tatt forutsetninger om at noen valg tas underveis. Dette for å kunne begrense utredingen til to alternativer. Disse forutsetningene er listet under:

- Det forutsettes videreføring av Oracle som database dersom alternativet for relasjonsdatabase alternativet blir valgt. Dette for å redusere parallelldriftskostnadene og kompleksiteten i prosjektperioden.
- Det forutsettes at triplestore teknologi som eventuelt skal benyttes blir valgt senere.
- Det forutsettes at søkeindeks for bruk i arkitekturen er valgt. Dette for å tilrettelegge for valg av en enklere database etter alt er migrert til ny arkitektur, slik at MUSIT kan redusere driftskostnader på sikt.
- Det forutsettes at det anbefalte konseptet fra forprosjektet legges til grunn. Dette medfører at en utvikler en ny løsningsarkitektur basert på felles informasjonsarkitektur og arbeidsprosessene i virksomhetsanalysen, uavhengig av databaseplattform.

I Forprosjektets rapport ble en videreføring av MUSITs relasjonsdatabaser i sin nåværende form (konsept 1), vurdert og besluttet som ikke aktuelt. Dette er derfor ikke vurdert som alternativ her (se http://www.musit.uio.no/musit/informasjon/Rapport_Forprosjekt_MUSIT_ITArkitektur_2015.pdf).

Alternativ 1 – Relasjonsdatabase

Alternativ 1 er en relasjonsdatabase med Oracle i bunn, med restrukturering og migrering av dagens database for å håndtere dagens krav i en felles datamodell. Dette må gjøres for å kunne ha en mellomvareserver som kan håndterer datasettene så likt som mulig.

Dette alternativet vil gi stor kompleksitet i datastrukturen for å kunne gi ønsket dynamikk med tanke på å legge til og endre felter og avanserte avhengigheter mellom dataene.

Alternativet vil også medføre høy kompleksitet i integrasjonsmoduler som må implementeres for å kunne snakke med dagens database, og ny database parallelt. Denne kompleksiteten vil avskjermes i integrasjonsmodulene som skal fjernes etter at ny arkitektur er komplett. Brukergrensesnittet vil ikke eksponeres for denne kompleksiteten siden denne blir abstrahert gjennom en felles mellomvare. Den største risikoen er knyttet til aktivitetene for integrasjonsmodulene for å understøtte dagens datamodell, samt ved migrering av data. I tillegg er det vesentlige utfordringer knyttet til å samordne behovene innen de enkelte fagområdene i en felles modell.

Alternativet vil kunne støtte opp rundt microservice arkitektur. Alternativet vil skape en del utfordringer rundt modellering siden modelleringsteknikkene som må brukes vil kunne få inngrep på kryss av microservicene. Dette skaper risiko og kompleksitet i forvaltning og drift. De aller fleste microservicene vil være selvstendige, og man vil ofte kunne forvalte og oppdatere en og en service.

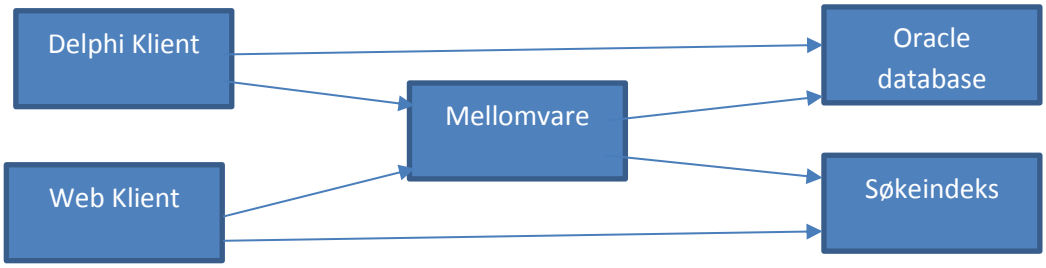
Alternativet er arbeidskrevende i utviklingsfasen, men vil gi en enklere mellomvare å forvalte i etterkant, ettersom databasens struktur forenkles i forhold til nåværende situasjon.

Valget av relasjonsdatabase vil gi noen begrensninger på hvordan dataene som skal lagres kan beskrives, spesielt med tanke på indeksering av søkeindeksen, som krever en flat og enkel struktur. Fleksibiliteten ved utvidelser vil være noe begrenset, og endringer kan ta lang tid. Videre vil det ved endringer i relasjonsdatabase alltid være risiko for uforutsette ringvirkninger. Eksempelvis problemer med at en tabell bryter fysiske begrensninger i oppsettet av datalageret, som resulterer i store utfordringer med ytelse, eller man bryter databasens fysiske begrensninger på hvor mange attributter den kan håndtere for en tabell m.m.

Inntil systemet er ferdig utviklet som helhet vil det også kreves noen endringer på gammel klient for å kunne bruke ny funksjonalitet mot gamle moduler. Funksjonaliteten som blir utviklet vil gradvis bli tilgjengeliggjort brukerne i ny drakt gjennom nettleser. Dette vil være tilsvarende som for alternativ 2.

Prosjektet bør benytte et ORM bibliotek for abstraksjon av Oracle databasen slik at man ikke blir sterkt bundet av databaseproduktet, dette fører til at man kan bytte ut databasen uten endring i kildekoden i etterkant.

De datamodellene som jobbes frem sammen med fagsiden, må transformeres til relasjonsmodeller for å kunne implementere dette i en relasjonsdatabase.



Fordeler:

- Databasedriften vil være uendret (kjent og stabil drift, etablerte backuprutiner).
- USIT har god kompetanse på drift av Oracle relasjonsdatabase.
- USITs utviklere ved DS har god kjennskap til Oracle relasjonsdatabase (gir trygghet).
- Enkel og god tilgang på kompetanse på relasjonsdatabaser i markedet.
- Oracle har meget gode verktøy for utvikling og feilsøking som forenkler hverdagen for utviklere.
- Mulighet for gode integritetssjekker på dataene som ligger i databasen, gitt at dette modelleres inn i relasjonsmodellene ved implementering.
- Museene har kompetanse på relasjonsdatabaser, noe som forenkler dokumentasjon, bearbeiding og migrering av datasett.
- God ytelse.
- Vil kunne komme raskt i gang med utviklingen.

Ulemper:

- Meget komplekse datamodeller for å støtte opp rundt de avanserte datakrav museene har.
- Kan være tungt å jobbe med endringer på datastrukturen.
- Kan ikke bruke Darwin Core og lignende internasjonale standarder direkte mot databasen, men må produsere oversettere til relasjonsdataene.
- Kompliserte og omfattende spørringer for å hente ut data på grunn av avanserte datastrukturer og datakrav.
- Oracle sine verktøy legger opp til Oracle spesifikt spørrespråk istedenfor å bruke standard spørrespråk. Dette kan gjøre det vanskelig å holde databasen flyttbar, siden utviklerne må styre dette selv.
- De datastrukturer MUSIT har, med mange relasjoner som går i sirkel, er motstridende med hvordan relasjonsdatabasen optimalt ønsker data strukturert. Noe som vil få ytelseskonsekvenser.

Alternativ 2 – Triplestore

Alternativ 2 er å bytte ut dagens relasjonsdatabase med en semantisk database (triplestore). For å kunne ha en mellomvareserver som kan håndtere MUSITs data så likt som mulig på kryss av fagområdene, må det både foretas en restrukturering av dagens data, og dagens datafelter må mappes riktig mot feltene i de internasjonale standarder (som Darwin Core og lignende).

Valg av triplestore vil skape høy kompleksitet i integrasjonsmoduler som må implementeres for at mellomvaren skal kunne snakke med dagens database og ny database parallelt. Denne

kompleksiteten vil avskjermes i integrasjonsmodulene som skal fjernes etter at ny arkitektur er komplett. Brukergrensesnittet vil ikke eksponeres for denne kompleksiteten siden denne blir abstrahert gjennom en felles mellomvare.

Dette alternativet vil ha høyest risiko rundt integrasjonsmodulene og migrering av data. Konvertering av datastrukturen (modelleringen) vil for en del områder være mer arbeidskrevende enn for alternativ 1, men selve migreringen vil ikke ha større risiko.

De datamodellene som bygges med fagsiden og samkjøres med de internasjonale standardene vil kunne brukes direkte i databasen. Datamodellene vil være enklere å utvikle, og vil være strukturert på en måte som vil være enklere å forstå for fagsiden enn en relasjonsmodell. De internasjonale standardene inneholder langt flere attributter enn hva MUSIT har tatt i bruk i dag, og vil kunne benyttes som et hjelpemiddel dersom det oppstår behov for å lagre mer informasjon om en forekomst i databasen enn man har i dag.

Kraften triplestoren har til å beskrive virkeligheten vil gi en risiko for å skape modeller som ikke lar seg indeksere godt og effektivt i en søkeindeks. Dette er viktig å være oppmerksom på ved endringer både underveis i prosjektet og forvaltningen i etterkant.

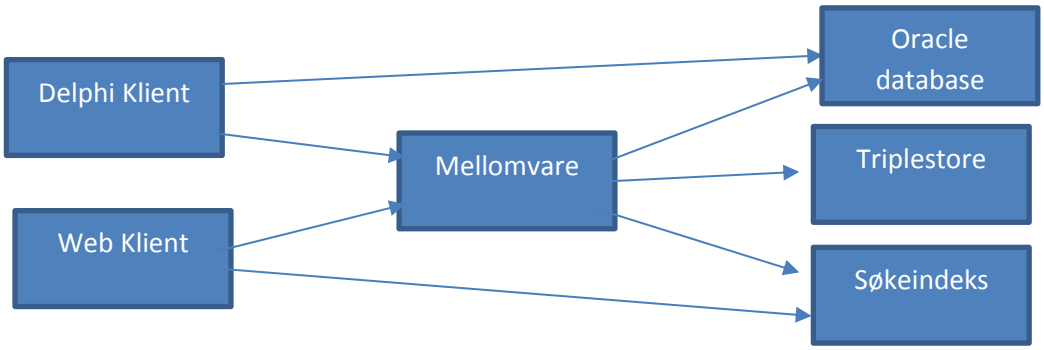
Triplestore har begrenset utvalg på verktøy for spørring og feilsøking sammenlignet med en relasjonsdatabase. Spørrespråket, som er en utvidelse av relasjonsdatabasens spørrespråk, bruker noen andre konsepter som kan bli utfordrende for USITs utviklere ved DS i gjennomføringen av prosjektet. Egenskapene ved spørrespråket til triplestore gjør at spørringene og datauttrekkene blir mindre omfattende og enklere å bygge når utvikleren får kontroll på dette.

Dette alternativet vil gi en datastruktur som er enkel å dele med andre interessenter, eksempelvis GBIF og Europeana, fordi disse bygger på standarder som er beskrevet i det samme filformatet som triplestore bruker for å utveksle datastrukturer og dataelementer (RDF).

Triplestore vil gjøre det enklere for MUSIT å implementere de internasjonale natur- og kulturhistoriske datastandardene (Darwin Core, CIDOC CRM m.fl), og forenkle utveksling av informasjon.

Definering av felles domenespråk må gjøres slik at en så stor del av dataene som mulig kan håndteres likt i MUSIT. Dette for å kunne håndtere størsteparten av dataene felles og legge til rette for enklere samarbeid med andre initiativer, og lette uttrekk fra databasen til eksterne institusjoner som GBIF.

Microservice arkitekturen vil være enklere å implementere ved å ha triplestores siden den ikke krever eierskap til datastrukturen, men eierskap til data. Datastrukturen blir definert av hvilken standard og versjon MUSIT velger som utgangspunkt for å lagre dataene (eksempelvis Darwin Core).



Fordeler:

- USITs utviklere ved DS har kjennskap til semantisk databaseteori og har implementert hybridløsninger i relasjonsdatabasen for å understøtte fagsidens krav på datastrukturer.
- Medfører mindre kompleks datastruktur enn relasjonsdatabase, siden koblingskravene kan beskrives som en del av støtten som er innebygd i databasen.
- Enkelt å utvide eller endre datamodellen.
- Vil forenkle uthenting betraktelig i forhold til relasjonsdatabase.
- Medfører mindre kompleksitet i spørringer som skal gjøres mot databasen, enn relasjonsdatabase.
- Standardene innenfor natur- og kulturhistorie kan gjenbrukes direkte for å utforme grunndatamodellen i nye MUSIT, og disse er utformet og blir oppdatert av representanter fra museumsfagsiden i flere land.
- UiB (gjennom universitetsbiblioteket) og NTNU har erfaring med drift av tilsvarende databaser (grafdatabase).

Ulemper:

- USIT har ingen erfaring med drift av denne typen database. Databasedriften må utredes med driftsorganisasjonen og utviklere (USIT-DS) må involveres i driftsprosessen. Antas å gi økt kostnad for prosjektet.
- Ekstern drift av triplestore ved UiB eller NTNU vil gi økt kostnad, med parallell drift, og øke kompleksitet for infrastrukturen.
- Bruk av triplestore som masterdatabase er nytt og mindre utprøvd enn relasjonsdatabaser.
- Noe begrenset tilgang på kompetanse i markedet, dette er en teknologi som nylig har begynt å bli utstrakt brukt.
- Enklere og mindre utviklede utviklingsverktøy enn relasjonsdatabasene.
- USITs utviklere ved DS kjenner ikke til fullblods triplestore (gir utrygghet).
- Datamodellering og oppstart av utvikling vil ta lengre tid enn ved relasjonsdatabase.
- Kompliserende arkitektur i overgangsfasen (må beholde eksisterende Oracle-base i lang tid).
- Dårligere ytelse enn relasjonsdatabaser, avhengig av søkeindeks.

Risikovurdering

Prosjektet har vurdert risiko ved valg av alternative 1 og 2. Under er en tabell som viser usikkerhetene med vurdering av sannsynlighet og konsekvens. Sannsynlighet multiplisert

med konsekvens vil gi en risikofaktor som brukes som en del av vurderingen av alternativene.

Sannsynlighet rageres fra «veldig lav», «lav», «moderat», «høy» og «veldig høy».
Konsekvenser rangeres fra «ubetydelig», «liten», «moderat», «alvorlig» og «veldig alvorlig».

Valg av relasjonsdatabase	Sann-synlighet	Kon-sekvens	Risiko	Kommentar/tiltak
Kompleks datamodell gjør det vanskelig og tidkrevende å sette seg inn i datastrukturen, både for eksisterende og nye utviklere.	Moderat 3	Moderat 3	9	
Kompleks datamodell gjør det tungt å jobbe med endringer av datastrukturen.	Moderat 3	Alvorlig 4	12	
Kompliserte og omfattende spørringer, gjør det vanskelig og tidkrevende å sette seg inn i, både for eksisterende og nye utviklere.	Høy 4	Moderat 3	12	
Kompliserte og omfattende spørringer, dette gir risiko for feil i spørringer/datauttrekk.	Høy 4	Veldig alvorlig 5	20	
Kompliserte og omfattende spørringer, dette gjør spørringer/datauttrekk tungt å vedlikeholde.	Moderat 3	Moderat 3	9	
Oracle benytter ikke standard SQL, gjør det mer komplisert å bytte database i ettertid.	Lav 2	Liten 2	4	Benytte ORM-bibliotek for abstraksjon av databasen.

Valg av triplestore	Sann-synlighet	Kon-sekvens	Risiko	Kommentar/tiltak
Manglende driftskompetanse ved USIT, gir risiko for at drift må kjøpes eksternt, noe som vil gi økte kostnader og kan gi redusert sikkerhet.	Lav 2	Veldig alvorlig 5	10	Drift kan eventuelt kjøpes av UiB og/eller NTNU.
Manglende driftskompetanse ved USIT, gir behov for å bygge opp kompetanse, noe som kan medføre kostnader for MUSIT.	Veldig høy 5	Moderat 3	15	Applikasjonsdrift har erfaring med drift av ulike servertyper. Kombinasjon av applikasjonsdrift og USITs utviklere ved DS vil kunne redusere risikoen. Prosjektet lager detaljert drifts- og installasjonsveiledning.
Begrenset tilgang på kompetanse i markedet.	Høy 4	Moderat 3	12	Bevegelser i markedet tilsier at tilgangen på kompetanse vil øke de nærmeste årene.

Begrenset utbredelse i markedet (lite utprøvd), dette gjør det vanskelig å finne hjelp på nett til problemløsning, samt gir en usikkerhet ved teknologien.	Høy 4	Moderat 3	12	Velge databaseprodukt med mest mulig utbredelse og god supportavtale.
Ukjent teknologi for USIT ved DS-teamet, noe som medfører at det vil ta lengre tid å modellere og komme i gang med utviklingen.	Moderat 3	Liten 3	9	Dagens modellering av hendelser er i tråd med modellering av triplestore.
Enkle og lite utviklede utviklingsverktøy, dette gjør utvikling og feilsøking mer kompleks og dermed tidkrevende.	Moderat 3	Moderat 3	9	
Museene har liten eksisterende kompetanse på teknologien, dette gjør fagsiden usikker på om de fortsatt vil ha kontroll på dataene.	Svært lav 1	Alvorlig 5	5	Tilgjengeliggjøre god dokumentasjon på felles datamodell.
Enkelt å utvide eller endre datamodellen, kan gjøre det for enkelt å få til endringer, noe som på sikt kan gjøre systemet mindre ensartet.	Svært lav 1	Alvorlig 5	5	Prosess styrt av produkteier skal sikre felles og ensartede endringer.

Oppsummering

Begge alternativene er omfattende, men gir forskjellige fordeler under utvikling og forvaltning. Både alternativ 1 og 2 vil medføre endret datastruktur, dermed vil det ikke være noen fordel for noen av disse alternativene at data i dag finnes i Oracle databasen. Begge alternativene vil kreve parallell drift av dagens MUSIT database og den nye som gradvis bygges opp. Både relasjonsdatabaser og triplestore skalerer like godt (det vil si dårlig), og har like gode backupmuligheter. Begge teknologiene er stabile og trygge databaser.

Alternativ 1 relasjonsdatabase:
Valg av relasjonsdatabase vil gi noe høyere kompleksitet for mellomvaren siden spørringene blir mye mer komplekse enn alternativ 2. Resultatet vil være en løsning med høyere kompleksitet som vil være mer rigid og tyngre å endre i etterkant enn ved bruk av triplestore. Løsningen vil kunne støtte bra opp rundt microservice arkitektur.

Alternativ 2 triplestore:
Triplestore vil skape lavere kompleksitet for mellomvaren siden forretningslogikken og spørringene blir mindre komplekse enn alternativ 1. Prosjektet vil ha raskere hastighet på utviklingen enn alternativ 1, siden spørringene er mindre komplekse. Men ved oppstart av utviklingen vil det kunne oppstå usikkerhet med ukjent lagringsteknologi. Resultatet vil være en løsning med lavere kompleksitet i forvaltning, og datastrukturen vil være dynamisk og enkel å justere i etterkant. Alternativet gjør microservice arkitekturen enkel å implementere.

Generelt vil begge alternativene her ha omtrentlig samme kostnad for utviklingen. Men i forvaltning antas det at triplestore vil være billigere enn relasjonsdatabase.

Anbefaling

Triplestore er bedre egnet for å håndtere kompleksiteten i MUSIT enn en relasjonsdatabase, grunnet sine datastrukturelle krav. Men utfra prosjektets rammer, tilgjengelige ressurser og risiko knyttet til teknologiens utbredelse og tilgjengelig kompetanse har prosjektet valgt å ikke anbefale triplestore for gjennomføringen av prosjektet. En har lite erfaring med teknologien fra før. Dette sammen med en eventuell endring av utviklingsplattform og språk, vil gjøre at en ny databaseteknologi medfører en for stor risiko for prosjektet.

Det anbefales derfor å gå for alternativ 1 som er videreføring av dagens relasjonsdatabase, men med ny felles datamodell og migrering av dagens databaser.

Alternativ 1 sørger for at man får:

- bygd opp en ny database som ikke inneholder mange sterke bindinger til leverandøren av Oracle databasen.
- samler så mange elementer som mulig mellom fagsidene, og man får enkel forvaltning etter at prosjektet er ferdig.
- åpnet for muligheten til å migrere over til en billigere databaselisens når MUSITs data og forretningslogikk er fullstendig migrert til ny arkitektur, noe som vil senke driftskostnadene på sikt.
- støtte for microservice arkitektur i mellomvaren.

Momenter til forvaltningsprosessen i etterkant av migreringsprosjektet

Når MUSITs data og forretningslogikk er ferdig migrert til ny arkitektur kan MUSIT vurdere andre alternativer for relasjonsdatabase, som eksempelvis PostgreSQL. MUSIT vil da ha muligheten til å kunne velge billigere lisensalternativer for datalagringen på sikt.

Dersom MUSIT ønsker å bytte fra relasjonsdatabase til triplestore senere, vil dette kreve en større innsats mht. både datamodell og omskriving av mellomvaren for å støtte opp om ny datastruktur. Det er derfor heller ikke å anbefale at en endrer databaseteknologi i løpet av prosjektperioden.

Søkeindeks utredning

Innledning

En søkeindeks vil fungere som et mellomlag hvor dataene i MUSIT og andre tjenester det ønskes å gjøre raske oppslag på, katalogiseres. Dette gjør at fagsiden vil være i stand til å raskt aggregere og søke på all informasjon på kryss av datakildene. En søkeindeks vil også være med på å sikre at eventuelle svakheter i aggregeringene som kan ligge i en database, ikke vil påvirke fagsiden i sitt daglige bruk av MUSIT. Den vil være langt raskere enn å bruke databasen direkte, uansett databaseprodukt. Å gjennomføre MUSITs IT-prosjekt uten en søkeindeks vil medføre til at det vil være vanskelig å lage en god søkefunksjon som brukes på tvers av fag og eksterne datakilder.

Det er i hovedsak to produkter som peker seg ut i dagens marked. Dette er:

- Elastic Search
- Solr

Disse produktene er mye brukt og omtalt, og er veldig like. Funksjonaliteten er helt lik bortsett fra aggregeringsstøtten på toppen av søkeindeksene. Tabellen viser en overordnet sammenligning:

	Solr	Elastic Search
Kan ta mot JSON for å fylle indeks	Ja	Ja
Kan aggregere på indeks	Lite til ingenting	Ja, utvidet støtte
Kan clustres på flere små maskiner.	Ja	Ja
Kan kjøre på en stor maskin	Ja	Ja
Clustringmekanisme	Zookeeper	Zookepper
Lisens	Apache	Apache
Web API for søk og kontroll	Ja	Ja

Som tabellen viser er Solr på mange måter et likeverdig produkt som Elastic Search, men det mangler aggregeringsstøtte. Det kan være verdt å merke seg at begge produkter er åpen kildekode (open source) og lisenstypene som ligger i bunn er like.

Produktene gir ingen lisenskostnader med mindre man ønsker å ha en supportlisens som gir tilgang på eksperthjelp fra produsent og leverandør av produktet.

Det finnes en detaljert teknisk sammenligning av produktene tilgjengelig for de som ønsker å se på detaljene: http://solr-vs-elasticsearch.com.

Anbefaling

I styringsgruppemøte den 25. januar besluttet styringsgruppen å anbefale Elastic Search som søkeindeks fordi den har full støtte for aggregering. Elastic Search vil sørge for at MUSIT får enda bedre aggregeringsmuligheter enn det som finnes i dagens database.

Lisensiering av prosjektet (åpen eller lukket kodebase)

Innledning

Museene må ta stilling til hvor vidt de ønsker å utvikle nye MUSIT som et åpent eller lukket kildekodeprosjekt. Dersom man velger en åpen kildekode modell må det også velges hvilken lisenstype man ønsker å ha i bunn. Dette er viktig med tanke på hvordan bruken av kodebasen skal være i fremtiden.

Åpen kildekode (open source)

Åpen kildekode modeller har blitt brukt mer og mer for å kunne dele både kildekode og arbeidsmengde. Slike modeller baserer seg ofte på copyright, og lisenser som krever tilbakeføring av 3dje parts innsats på kildekoden.

Fordelen med å gå for en åpen modell er at eventuell samhandling med andre blir veldig enkelt, og interessenter fra andre museer rundt om i verdenen kan bidra med felles utviklingsinnsats. Denne typen samarbeid må styres ved hjelp av planer og regler for å sikre MUSITs funksjonalitet.

Det er ingen garanti for at noen henger seg på, men det er enkelt å samarbeide når man har tilgang på kildekode med andre prosjekter.

Mye brukte lisenser og "executive overview" over hva disse impliserer:

MIT og BSD
Disse to lisensene gir i utgangspunktet fri bruk til 3dje part, uten å sette krav til tilbakeføring av utvidelser og feilrettinger av kildekode.

Apache og Eclipse
Disse to lisensene gir full frihet på bruk av koden slik at eventuelle feilrettinger skal føres tilbake men utvidelser kan gjøres uten at dette impliserer krav om tilbakeføringer.

GPL serien
Denne lisenstypen gir fritt innsyn som de andre lisenstypene, men krever at all videreutvikling og feilretting må tilbakeføres gjennom publisering eller patch.

Lukket kildekode (closed source)
Lukket kildekode er tradisjonelt brukt mye i privat sektor for å hindre innsyn og eie kodebasen alene. Dette blir veldig ofte brukt for produkter og skreddersøm som ikke er ment å deles med andre.

Anbefaling
I styringsgruppemøte den 25. januar besluttet styringsgruppen å anbefale åpen kildekode, og at det skal velges lisens som er mest mulig åpen. Valg av lisens skal gjøres i samarbeid med USITs jurist for å sikre at MUSITs behov ivaretas på best mulig måte.

Ordliste
Aggregering
Å gå gjennom en mengde data og avleder en verdi fra denne. Brukes typisk når man teller antall forekomster, summerer tallene i en kolonne i et regneark, tar gjennomsnittet av tallene i en kolonne i et regneark og lignende operasjoner.

Backup
Å ta en digital kopi av master data og lagrer dette på et trygt sted.

Bibliotek
I språksammenheng henviser et bibliotek til en samling kildekode som er gruppert sammen for å kunne gjenbrukes av flere prosjekter. Disse bibliotekene er ofte gjort om til maskinkode som er spesifikk til ett økosystem.

Brukergrensesnitt
Brukergrensesnittet er beskrivende for de websider som lar sluttbrukere jobbe mot systemet. Dette kan beskrives ved hjelp av mange synonymer som brukerflate, GUI, Frontend og lignende.

Cluster
Når man syr sammen flere like datamaskiner, som skal gjøre identisk jobb, for å samarbeide. Dette kan brukes til å øke regnekapasiteten til installasjonen eller man kan bruke dette for å øke driftssikkerheten til systemet, slik at hvis en server faller ned vil den andre svare istedenfor.

Database

En datamaskin/server som kjører en applikasjon som er spesialisert på å lagre og hente ut data.

Datalager
Et sted hvor den fysiske lagringen av digital informasjon skjer.

DevOps
Dette er en moderne prosessbetegnelse for samarbeid mellom utvikling, drift og test. Ofte har automatisering av prosesser som går på kryss av disse funksjonene fokus, for å lette samarbeid og øke kvaliteten.

Executive overview
Et enkelt og kort sammendrag som skal være forståelig for ikke faglig kyndige, for området det omhandler.

Forretningslogikk
Implementerte krav for hvordan data skal behandles og valideres. Dette kan være hvordan en person ser ut, hvilke navne regler gjelder for person, hvordan personnummer er bygd opp, hvordan man snakker med databasen og så videre.

Forvaltning
Når et prosjekt er ferdig utviklet og leverer sitt produkt til drift vil applikasjonen gå over i en vedlikeholds fase, denne kalles forvaltning.

Frontend
Synonym med brukergrensesnitt.

Funksjonalitet
Implementerte krav.

Funksjonelle språk
En programmeringsteknikk hvor kildekoden blir samlet i funksjoner. Funksjoner kan også motta referanser til andre funksjoner, dette gjør meget ofte at man kan definere funksjonalitet med å skrive mindre kildekode. Denne type programmeringsteknikk er spesielt godt egnet til å støtte opp rundt matematiske teorier og algoritmer, og øker testbarhet betraktelig sammenlignet med Objektorienterte språk. Eksempelvis implementasjonen av koordinat konvertering for kart.

Grafdatabase
En database hvor oppbyggingen av forekomstene i datalagret bruker grafteori. Dette gjør at man enkelt kan beskrive data mere virkelighetsnært og gjøre meget komplekse utredninger som eksempelvis dijkstra (shortest path), slektskapsutredning av forekomster og lignende.

Hybride språk
Dette indikerer at et språk støtter både objektorientering og funksjonell programmering. Slike språk inneholder fordelene til begge verdenene.

Kompilere
Tolke et språk og gjøre det om til maskinkode.

Maskinkode
Maskinkode er små enkle arbeidsinstruksjoner som er spesialiserte på å beskrive hva et operativsystem eller maskinvare skal gjøre. Dette er meget lite leselig og egner seg ikke for mennesker å jobbe direkte med. Det er spesielt tilrettelagt for å leses av en maskin.

Master data

Dette er hoved forekomsten av en digital dataforekomst. Kan sidestilles med et type objekt i naturhistorie og kulturhistorie.

Mellomvare
I en tre-lags arkitektur er ett av lagene mellomvaren, her bruker man å legge forretningslogikken til applikasjonen slik at denne er sentral og felles for alle klientene. Dette gjør at eksterne applikasjoner kan integrere mot mellomvaren og MUSIT bevarer tryggheten for sine data og datastrukturer.

Microservice
Microservices er en arkitektur som indikerer at man designer mellomvaren ved hjelp av mange små tjenester som kan stå alene og ikke er avhengig av andre tjenester for å gjøre jobben sin. Dette er en gammel arkitektur som i senere tid har blitt populær, etter samarbeidet mellom utvikling og drift har blitt bedre og man har fått utviklet gode automatiserings prosesser for å oppdatere og drifte en slik arkitektur.

Objektorienterte språk
En programmeringsteknikk som samler funksjoner og informasjon i enheter som kalles objekter, disse kan muteres gjennom arv og enkelt bygge gjenbrukbare hierarkier med kildekode. Denne type programmeringsteknikk er spesielt godt egnet til å styre livssykluser i digitale systemer, som eksempelvis håndtering av database oppkobling med avansert tuning og mange oppkoblinger slik at bruken av databasen blir meget enkel og trygg fra utsiden.

Operativsystem
Dette er et synonym til plattform.

ORM
Object Relational Mapper. Dette er en betegnelse for en teknikk som abstraherer en databaseforekomst inn i objekt og attributter som er tilpasset det språket utviklerne bruker. De aller fleste slike verktøy gjør at databasen blir fristilt fra kildekoden så man enkelt kan bytte ut database laget uten å endre kildekoden.

Provisjonere
Å klargjøre, konfigurere og installere noe ved hjelp av automatiske prosesser. For eksempel kan man provisjonere en datamaskin med Windows og Microsoft Office automatisk ved bruk av automasjonsverktøy på driftssiden. Dette indikerer at en eller annen applikasjon laster inn Windows og Microsoft Office og klargjør denne datamaskinen uten behov for manuelle prosesser utover å koble opp og trykke på en knapp.

Plattform
Plattform henviser til kjøresystem, eksempelvis Windows, Linux, OSX som kjører i bunnen på en hver datamaskin som brukes. Plattformen er hvor applikasjoner installeres og gjøres tilgjengelig for bruk av sluttbrukere.

Relasjonsdatabase
En database som lagrer data i strukturerte enheter som skjema, tabell, kolonne og rad. Relasjonsdatabase oppbyggingen er ganske likt hvordan excel er oppbygd med dokument, ark, kolonner og rader, men hvor en kan strukturere informasjonen i separate tabeller med definerte relasjoner.

Språk
Et menneskelig oppbygd logisk språk for å beskrive hva en datamaskin skal gjøre. Språk kompileres ofte til maskinkode som er tilpasset ett spesifikt økosystem.

Språkplattform
Dette er et synonym til økosystem.

Techlead
En senior ressurs i et utviklingsteam som har erfaring og kunnskap om teknologien og arkitekturen som brukes i prosjektet. Denne ressursen har typisk ansvar for å hjelpe utviklerne i utviklingen for å oppnå den tekniske arkitekturen og de tekniske målene til arkitekturen.

Triplestore
En grafdatabase som bruker semantisk graf teori for å beskrive oppbygging. Styrken til denne graftypen er analyser som går på store komplekse datasett.

En triplestore bryter med tradisjonene med å lagre data som rader og kolonner i tabeller (jamfør relasjonsdatabaser). Triplestore lagrer objekter som har relasjoner ved hjelp av grafer. Dette gir et system muligheten til å beskrive relasjoner og sammenhenger direkte i databasen, uten å bruke avanserte strukturer for å tvinge disse sammenhengene inn i rader og kolonner.

En triplestore gir friheten til å beskrive tilstander på data som ville vært særdeles vanskelige å beskrive i en relasjonsdatabase, uten å bruke konstruksjoner som innfører høy kompleksitet for utviklere. Spørringer mot data i en relasjonsdatabase som har høy kompleksitet fører ofte til feil, selv for meget erfarne og dyktige utviklere.

En triplestore kan også beskrive en relasjon mot andre eksterne systemer enkelt og entydig, slik at det å beskrive en løsere kobling mot eksempelvis dokumenter som ligger lagret digitalt i et arkiv, en artikkel på web eller filsystem er beskrivende og ukomplisert ved hjelp av vanlige urler og mekanismene i standardene. I en relasjonsdatabase må man definere integrasjoner entydig gjennom metadata tabeller og integrasjonsmekanismer i forretningslogikken slik at applikasjonen må støtte hvert system spesifikt.

Det er heller ingen begrensning på hvor mange relasjoner som kan defineres, til og med av samme type, i en triplestore. Det er også veldig enkelt å legge til nye relasjonstyper/attributter for å utvide databasen under forvaltning.

Eksempelvis blir triplestore benyttet i bakgrunnen for tjenester som Difi eInnsyn for å kunne tilgjengeliggjøre innsynsdata på kryss av mange tjenester. Dette gjør Difi i stand til å utlede avanserte analyser raskt og uten nevneverdig kompleksitet.

Verbos
Hvis man må skrive mye for å beskrive lite er man verbos.

Økosystem
Økosystem henviser til hovedbiblioteker og standarder som brukes i bunn av et utviklingsverktøy, slike utviklingsverktøy kan støtte mange språk som kompileres til felles maskinkode. Økosystemene har meget ofte maskinkode i bunn som er uforståelig og vanskelig å jobbe med, men egner seg til å kjøre programkode.