Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Erlend Solbakken Nikolaisen

Et system som bruker semantisk nett data for søk i og forslag til anime

Masteroppgave i Datateknologi
Veileder: Trond Aalberg
Juli 2023

NTNU
Kunnskap for en bedre verden

Erlend Solbakken Nikolaisen

Et system som bruker semantisk nett data for søk i og forslag til anime

Masteroppgave i Datateknologi
Veileder: Trond Aalberg
Juli 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Sammendrag

Internett er fullt av informasjon og databaser. Søkemotorer har blitt utviklet slik at brukere kan søke etter informasjonen de er interessert i. Men for en bruker som kanskje leter etter regissøren av en film de liker og andre filmer av regissøren, kan det være nødvendig med flere spørringer for å finne ønsket informasjon. Spesielt hvis de bare vet navnet på filmen. Da må brukeren kanskje først søke etter navnet på regissøren og deretter andre filmer av regissøren. Det Semantiske Nettet kan inneholde informasjonen brukeren trenger på en strukturert måte. Men for at en bruker skal kunne søke på det Semantiske Nettet, trenger de informasjon om hvordan dataene er strukturert og kunnskap om et strukturert spørrespråk som SPARQL. Dette gjør det vanskelig for en vanlig bruker å bruke det Semantiske Nettet.

Systemet som er lagt frem her hadde som mål å gjøre det enklere for en vanlig bruker å spørre om anime, regissøren og produsenten, og annen anime laget av de samme regissørene og produsentene. En kunnskapsbase ble opprettet med anime, studioene som lagde dem, regissører og produsenter involvert i dem, og figurene i disse animene. Forslagene til anime laget av en regissør kommer fra å rangere hver anime i kunnskapsbasen basert på deres sammenheng som et mål på viktighet.

Testene som ble kjørt på systemet antyder at systemet er brukbart for å finne regissøren eller produsenten av en anime og få forslag til andre anime de har laget. Forslagene som gis er ikke de samme som de som er de høyest rangerte for en direktør på nettsteder som MyAnimeList, men inneholder en kombinasjon av både de som har høy rangering og de som er lavere rangert. Dette fungerer bra siden de ikke er anbefalinger, men forslag.

Abstract

The internet is full of information and databases. Search engines have been developed for users to query for the information they are interested in. But for a user that might be looking for the director of a movie they like and other films by the director, several queries might be necessary to find the desired information. Especially if they only know the name of the movie. Then the user might need to first query for the name of the director and then other movies by the director. The Semantic Web can contain the information the user needs in a structured manner. But for a user to be able to query the Semantic Web they need information on how the data is structured and knowledge about some structured query language such as SPARQL. This makes it difficult for a regular user to use the Semantic Web.

The system that was developed aimed at making it easier for a regular user to query about anime, its director and producer, and other anime made by those same directors and producers. A knowledgebase was created with anime, the studios that made them, directors and producers involved with them, and characters in those anime. The suggestions for anime made by a director comes from ranking every anime in the knowledgebase based on their interconnectedness as a measure of importance.

The tests that were run on the system implies that the system is usable for finding the director or producer of an anime and getting suggestions of other anime they have made. The suggestions that are made are not the same has the highest ranking ones for a director on sites like MyAnimeList but contain a combination of both ones that are high rank and ones that are lower ranked. This works well since they are not recommendations but suggestions.

Innhold

1 Introduksjon

En viktig del av internett er tjenester som tilbyr informasjon og anbefalinger. Noen av disse tjenestene kan være knyttet til media som Netflix og NRK, mens andre bare gir tilgang til søk etter informasjon som Google.

Når man leter etter informasjon og anbefalinger for spill, filmer, bøker, etc., så vil man ofte se på for eksempel regissøren av en film man liker og så se etter andre filmer med samme regissør. For filmer og serier kan man bruke IMDb hvor man kan søke etter filmer for å se hvem som var regissør og regissører for å se hvilke andre filmer de har vært involvert i. Denne prosessen krever flere steg hvis man ikke vet hvilken regissør man er interessert i og krever like mange steg om man heller vil se på skuespillere. Denne informasjonen kan finnes som semantisk web data i kunnskapsbaser, men disse baserer seg på spørringer gjort i spørrespråket SPARQL. Dette gjør det vanskeligere for en vanlig person å lete informasjon i slike kunnskapsbaser.

Jeg vil prøve å løse dette problemet for søk etter anime og manga. Jeg er ønsker å lage et system som lar en bruke søke etter anime eller manga og få som resultat produsent/forfatter og noen forslag på anime/manga som også har blitt laget av samme produsent/forfatter. Forslagene burde også inneholde noen populære anime/manga. Systemet burde også kunne utvides til å kunne gi forslag basert på kombinasjoner som skuespiller+produsent for anime og forfatter+kunstner for manga.

1.1 Forskningsspørsmål

For å nå målet med å lage et system som kan brukes til å søke i anime og manga fra semantisk web data har jeg definert et forskningsspørsmål.

Spørsmål 1 *Hvordan kan listebasert søk brukes til å forenkle søk i semantisk web data?*

2 Bakgrunn

For å lage systemet trengs det informasjon om delene som skal inngå i systemet. Jeg ønsker blant annet å se på om jeg kan utnytte utforskende søk og serendipitet (tilfeldig funn av nyttig informasjon). Dette er for hjelpe brukeren til å finne den informasjonen de trenger eller finne overraskende og ny informasjon.

Et slikt system trenger noen hovedaspekter:

- Ett system som gir forslag til brukeren basert på spørringen
- Ett robust datasett som minneholder den informasjonen som trengs for å kunne svare på spørringer fra brukeren
- Ett rangeringssystem for å kunne gi forslag på populære anime/manga
- Brukeren skal kunne gjøre spørringer

Dataen består av semantisk web data. For å lagre dette i en database skal det brukes RDF, som er hoved formatet for å representere semantisk web data. Data skal hentes fra en kilde på nettet og gjøres om til RDF format. Grunnlaget for RDF er et sett med tripler som hver består av ett subjekt, ett predikat, og ett objekt (Wood mfl. 2014). Et sett med slike tripler kalles en RDF graf og kan visualiseres som en rettet graf med noder og kanter. En grafdatabase kan brukes for å utnytte det faktum at settet med tripler former en graf.

(Kiryakos og Pfeffer 2021) benytter seg av RDF og ontologier for å lagre data om visuell media fra Japan. Dataen de benytter seg av har de hentet fra flere kilder for å lage en mest mulig komplett database. Målet deres var å lage en database som kunne brukes til å gjennomføre forskning på visuell media fra Japan og er derfor basert på at brukeren selv kan lage SPARQL spørringer.

(Zevio mfl. 2018) prøver å identifisere underholdning fra Japan som tilhører samme transmedia. Det identifisere underholdning fra DBpedia og finner lenker mellom de som kan tyde på at de høre transmedia. Dette arbeidet går mest ut på å identifisere underholdning fra Japan og ikke noe om å søke i dataen.

2.1 Semantisk web

Semantisk Web, eller nettet av data, er et nett av informasjon. Semantisk Web er en utvidelse av World Wide Web som skal gjøre det enklere for datamaskiner å forstå informasjonen. World Wide Web benytter seg av dokumenter og hyperlenker som

kobler dokumentene sammen. På den andre siden bruker Semantisk Web ressurser og relasjoner mellom dem. Det kalles for semantisk fordi relasjonene sier noe om ressursene de kobler sammen.

En ressurs i Semantisk Web representerer en digital eller fysisk ting, eller et abstrakt konsept. En ressurs er identifisert ved å bruke en Uniform Resource Identifier (URI). For eksempel er URIen for Norge i DBpedia https://dbpedia.org/page/Norway. Denne URIen navngir Norge som en ressurs og at data til ressursen kan nås ved å sende en forespørsel til URIen.

For å lagre informasjon om en ressurs bruker vi tripler. Disse triplene består av et subjekt, et predikat, og et objekt. Så hvis vi vil lagre informasjon om at Norge er en del av Europa kan vi bruke en trippel hvor *Norge* er subjektet, *Kontinent* er predikatet, og *Europa* er objektet. Denne metoden for å lagre informasjon om ressurser på er en del av Resource Description Framework (RDF).

Semantisk Web lagres ofte i et triple-store, og data hentes ut ved å bruke spørrespråket SPARQL. En spørring i SPARQL konstrueres ved at man definerer hvordan informasjonen man vil ha «ser ut». Hvis man vil finne alle landene i Europa kan man spørre etter ting som er land og at de tingene er en del av Europa. Listing 1 er et eksempel på den spørringen skreve i pseudo-SPARQL.

```
SELECT ?Country WHERE {
    ?Country isA Country .
    ?Country Continent Europe.
}
```

Listing 1: Eksempel på SPARQL spørring

2.2 Kunnskapsbaser

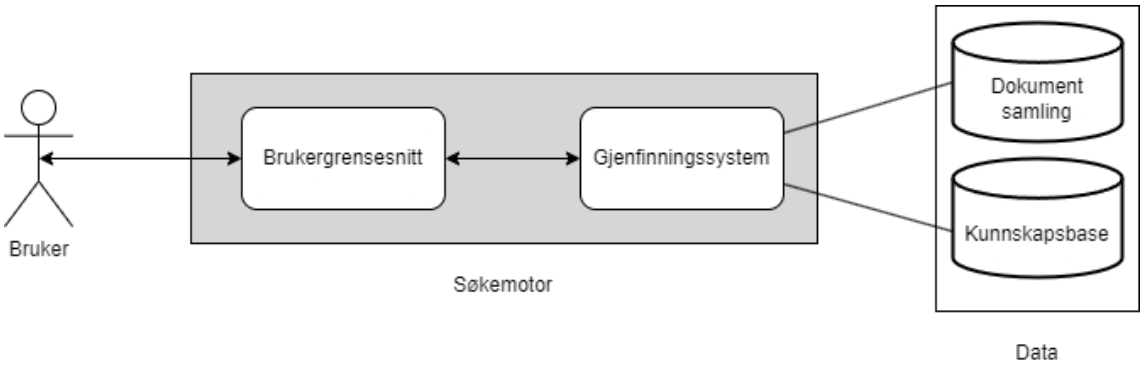
En kunnskapsbase er en stor samling av påstander om verden. Disse påstandene beskriver spesifikke entiteter og deres relasjoner. Dette er for å gjøre informasjonen leselig for maskiner (Balog 2018). Kunnskapsbaser kan være både generelle og domene spesifikk. Flere prosjekter jobber med å lage kunnskapsbaser med informasjon fra Wikipedia som DBpedia og YAGO. På nettet blir de fleste instanser beskrevet av RDF, og schema for kunnskapsbaser blir skrevet med et deklarativt språk som RDFS (for å uttrykke taksonomiske relasjoner) eller OWL (for å modellere hele ontologier). I informasjonsgjenfinning og naturligspråk prosessering har kunnskapsbaser blitt en sentral del for å få maskiner til å forstå naturlig språk. En kunnskapsbase vil aldri kunne være komplett for det vil alltid finnes ny informasjon som kan legges til.

En kunnskapsbase kan bli delt inn to deler.

- I schema delen har man en kunnskapsmodell som definerer semantiske klasser for entiteter, relasjonene mellom klassene og instansene, og egenskapene til en instans av en klasse.
- Instans delen består av et sett med påstander om spesifikke entiteter som beskriver typen, navnet, attributter, og relasjoner med hverandre.

2.3 Entitetsorientert søk

Entitetsorientert søk refererer til informasjons-hentende aktiviteter hvor entiteter brukes i stedet for dokumenter. (Balog 2018) definerer det som «søkeparadigme for organisering og tilgang til informasjon sentrert rundt enheter, og deres attributter og forhold». Paradigmet kan ses fra to perspektiver. For en bruker er entiteter en naturlig enhet for å organisere informasjon. Vi tenker oftest på ting i den virkelige verden og ved å la brukeren interagere med spesifikke entiteter gir det en bedre og mer effektiv brukeropplevelse. Fra perspektivet til en maskin gir entiteter en bedre forståelse av spørringer, innholdet i dokumenter, og brukere.



Figur 1: Overordnet oversikt av et entitetsorientert søkesystem

Sett ovenfra, som vist i Figur 1, består et entitetsorientert søkesystem av tre deler: brukeren og deres informasjonsbehov, en søkemotor, og data. En bruker kan uttrykke sitt informasjonsbehov på flere måter, som blir ofte kalt søkeparadigmer. Tradisjonelt har disse vært delt opp i tre typer spørringer: nøkkelord, strukturert, og naturlig språk. Nøkkelordspørringer er blitt den mest dominante måten å søke på grunn av nettsøkemotorer. Strukturerte spørringer brukes for å gjøre spørringer på en strukturert datakilde ved å bruke spørrespråk som SPARQL. Naturlig språk spørringer er formulert med naturlig språk slik som en person bruker i en samtale med en annen.

Søkemotoren består av to deler: et brukergrensesnitt og et gjenfinningssystem. Brukergrensesnittet tar seg av samhandling med brukeren, som formulering av informasjonsbehov og presentasjon av resultatene. Gjenfinningssystemet tolker spørringen og formulerer et svar.

I dette systemet består data av to deler: en dokumentsamling og en kunnskapsbase. Kunnskapsbasen inneholder minst en entitetskatalog med navn og unike ID-er, men også ofte beskrivelsen og egenskapene til entitetene i en strukturert eller semistrukturert form.

2.4 Rangering

For å kunne gi brukeren forslag fra de mest populære anime/manga for en produsent/forfatter er det et behov for en type rangering. Rangering er mye forsket på og det finnes flere måter å gjøre det på. For mitt formål er det viktig å rangere produsent/forfatter, og at anime/manga blir rangert i forhold til den personen den er relatert til.

(Dom mfl. 2003) har sett på flere metoder for å rangere noder i en graf. De ser på eposter mellom fagpersoner og prøver å finne ut hvem som har mest ekspertise på et fag ved å se på de to personene som to lag. Rangeringen ble gjort på grafer hvor man allerede visste den faktiske rangeringen. En av algoritmene som kom best ut var PageRank som er mest kjent for å være brukt av Google. PageRank er algoritmen som Google bruker til å rangere nettsider i søkemotoren deres. Den fungerer slik at antall lenker som peker til en nettside og kvaliteten på de brukes til å estimere hvor viktig en nettside er. Dette er basert på en antagelse at en viktig nettside har større sannsynlighet for å bli lenket til av andre nettsider.

2.5 Utforskende søk

Utforskende søk har en litt uklar og åpen definisjon. (Palagi mfl. 2017) prøver å komme med en definisjon ved å se på hvordan det har vært definert tidligere og hvilke karakteristikker utforskende søk har. Dette er karakteristikker som en søkeprosess som utvikler seg, mange mulige svar, og en serendipitøs holdning hos brukeren, som betyr at man er åpen for og legger merke til «tilfeldige» elementer. Et eksempel på utforskende søk er å ville ha mer informasjon om et tema som «Trondheim» men man vet ikke helt hva slags informasjon man er ute etter. Man kan starte å bare søke på Trondheim og få informasjon som areal og innbyggertall. Men så kan man også kommer over informasjon om turistattraksjoner i Trondheim som kan føre til

søk om andre attraksjoner i Trondheim.

(Golovchinsky mfl. 2012) prøver å designe et system som skal forbedre utforskende søk. De prøver å bruke historisk metadata som blir laget av søkeprosessen til å hjelpe med å utvikle søkeaktiviteten. De designer også et brukergrensesnitt som skal hjelpe brukeren å uttrykke sitt informasjonsbehov og vurdere resultatene. Design aspektene kan være brukbare her for å lage et system som er gir bedre mulighet for utforskende søk.

2.6 Serendipitet

Serendipitet er definert som forekomst og utvikling av hendelser ved en tilfeldighet på en lykkelig eller fordelaktig måte. (Toms mfl. 2000) ser på serendipitøs informasjonsgjenfinning som noe som skjer når en bruker uten spesifikke intensjoner oppdager nyttig informasjon ved å se på en node med informasjon.

(Kotkov mfl. 2016) ser på serendipitet i kontekst av anbefalingssystemer. De fant at det er ingen enighet om definisjonen på serendipitet og det er vanskelig å måle på grunn av dette. De bestemte seg for at i et anbefalingssystem så er serendipitet en egenskap som indikerer hvor bra systemet er til å foreslå serendipitøse elementer som er relevante, nye og interessante, og uventet for en bestemt bruker. De fant at for offline testing av serendipitet i anbefalingssystemer brukes kvantitativ og kvalitativ analyse. For kvantitativ analyse fant de at fordi datasett ikke inneholder data om serendipitøse elementer gjør forskere antagelser om serendipitøse elementer og bruker flere beregninger for å måle serendipitet. For kvalitativ analyse fant de en artikkel som sammenlignet en brukerprofil og anbefalte elementer for å måle serendipitet.

2.7 Tidligere arbeid

Kaufmann mfl. 2007 presenterer NLP-Reduce, et naivt men domeneuavhengig naturlig språk grensesnitt for å gjøre spørringer mot en Semantisk Nett kunnskapsbase. Den kalles naiv fordi den er simpel, behandler spørringer som en pose med ord, bruker bare et redusert sett med behandlingsteknikker for naturlig språk, for eksempel synonymutvidelse og stamming. NLP-Reduce prøver ikke å forstå spørringer, men prøver bare å sammenligne ord fra spørringene og deres synonymer med utrykkene som er brukt i kunnskapsbasen. Det er mulig å gjøre spørringer med hele setninger, deler av en setning, eller bare nøkkelord. For en kunnskapsbase som brukes i systemet så bygges det opp et leksikon av alle triplene og synonymer til merkelappene for hver trippel. En spørringsgenerator prøver å finne triplene i kunnskapsbasen som

passer sammen med ordene i spørringen. Triplene som passer, blir brukt til å generere en SPARQL spørring. Avhengigheter mellom ord og uttrykk i spørringene er bare identifisert av relasjonene mellom elementene i kunnskapsbasen. Tilnærmingen er derfor avhengig av kvaliteten og vokabularet til kunnskapsbasen som blir brukt. Denne svakheten er også en stor styrke fordi systemet trenger ingen endringer for å brukes på nye kunnskapsbaser.

I **Tablan mfl. 2008** presenterer de QuestIO, et naturlig språk-grensesnitt som gir tilgang til strukturert informasjon fra en kunnskapsbase. Det ble utviklet for å være robust med hensyn til språktvetydigheter og ufullstendige eller dårlig utformede spørringer. Dette prøver den å oppnå ved å utnytte strukturen til ontologier, uklar strengmatching og ontologi-motiverte likhetsmålinger. For å kunne håndtere alle inputer baserer ikke systemet seg på syntaks eller grammatisk korrekte spørringer. QuestIO legger mest vekt på å bruke informasjonen i kunnskapsbasen og bruker kun svært lett språklig behandling av spørringene. Systemet bygger opp en gazetteer ved å hente ut leksikaliseringer fra kunnskapsbasen. Gazetteeren brukes til å identifisere omtaler av klasser, egenskaper, forekomster og egenskapsverdier tilknyttet instanser. Ved en spørring blir det gjort en lingvistisk analyse, oppslag i gazetteren, iterativ konstruksjon av en SeRQL spørring, spørringen blir utført på en kunnskapsbase og resultatene blir vist til brukeren. QuestIO ble testet på 2 forskjellige ontologier. GATE (General Architecture for Text Engineering) ontologien var bygget opp av kildekoden til GATE og XML konfigurasjonsfiler. Det ble tilfeldig plukket ut 22 spørsmål som ontologien inneholdt svaret på og QuestIO fant svaret på 68% av spørsmålene. For å teste skalerbarheten og portabiliteten ble ≪Travel Guides Ontology≫ brukt. En mye større ontologi med informasjon fra turisme domenet. Testene viste at QuestIO kunne brukes uten endringer på begge ontologiene selv om de var veldig forskjellige både semantisk og i størrelse.

Ferrández mfl. 2009 presenterer QACID, et brukersentrert ontologibasert spørsmål-svar system. Det baserer seg på å samle spørringer fra et spesifikk domene og bruke disse spørringene til å kunne svare på nye spørringer fra en bruker. Spørringene samles inn, grupperes slik at en gruppe inneholder forskjellige uttrykk som ber om samme informasjon, og gruppene assosieres med SPARQL uttrykk. Disse gruppene samles i en database. Nye spørringer blir analysert for å finne semantiske likheter mellom spørringen og gruppene i databasen. Domenet de valgt i artikkelen å bruke for ontologien var kino. For å bygge opp databasen med spørringer fra brukere fikk de 50 personer til å lage spørringer til kino domenet. Personene ble vist ontologien og en liste med entiteter som fantes i ontologien slik at de kunne generere spørringer for all informasjonen de var interessert i. De endte opp med et sett av 500 spørringer. Etter spørringene var analysert og gruppert endte de opp med 54 semantiske grupper og hver gruppe ble assosiert med et SPARQL uttrykk. For å

sammenligne nye spørringer med spørringene i databasen forsøker QACID å etablere leksikalsk-semantiske slutninger mellom en ny spørring og spørringene fra databasen. Denne tilnærmingen anser implikasjonene som en enveis betydningsrelasjon mellom to spørringer, der betydningen av en må være utledet fra meningen av den andre. For å teste systemet laget 10 personer en spørring for hver gruppe, så de endte opp med 540 spørringer. Med test spørringene endte de opp med en presisjon på 89.24% og tilbakekall på 97.53%. I konklusjonen skriver de at løsningen kan brukes på andre domener enn det de brukte. Da trengs bare en ny ontologi som modellerer det nye domenet som kan brukes til å genere nye bruker spørringer.

3 Design

I dette kapittelet forklarer jeg designet av systemet. Det overordnete prinsippet jeg skal prøve ut er om semantisk web data og rangering kan benyttes til å gi forslag til andre serier som er relatert til serier det søkes med.

3.1 Datasettet

Datasettet er hentet fra MyAnimeList og inneholder data fram til 10.04.2023.

Det originale datasettet ble midlertidig lagret i en relasjonsdatabase uten definerte relasjoner mellom tabellene. Fordi målet er å bruke en database med Resource Description Framework (RDF), må dataen først analyseres for å bedre forstå datamodellen som er brukt. Det neste steget er å transformere dataen til rett format. Det finnes metoder for å direkte omgjøre relasjonsdata til RDF (Prud'hommeaux mfl. 2012, Sundara mfl. 2012) men disse baserer seg på at det finnes en ferdig relasjonsdatabase med komplett schema. Fordi relasjonsdatabasen bare brukes som mellomlagring, har den ikke ett komplett schema og mesteparten av dataen er lagret som JSON. Derfor kan transformasjonen av data anses som å gjøres direkte på JSON fra kilden (MyAnimeList).

Analyse av datasettet førte til at seks separate klasser ble definert. Disse skal brukes som utgangspunkt for å definere predikater og hvilken informasjon hvert objekt skal inneholde. De seks klassene som ble definert er:

- Anime (animerte filmer og serier)
- Manga (her både skrevne bøker og tegnede serier som manga)
- Personer (som har vært involvert i skapelsen av anime og/eller litteraturen)
- Karakterer (som dukker opp i anime og/eller litteratur)
- Studio (som har produsert anime)
- Sjanger (action, drama, osv.)

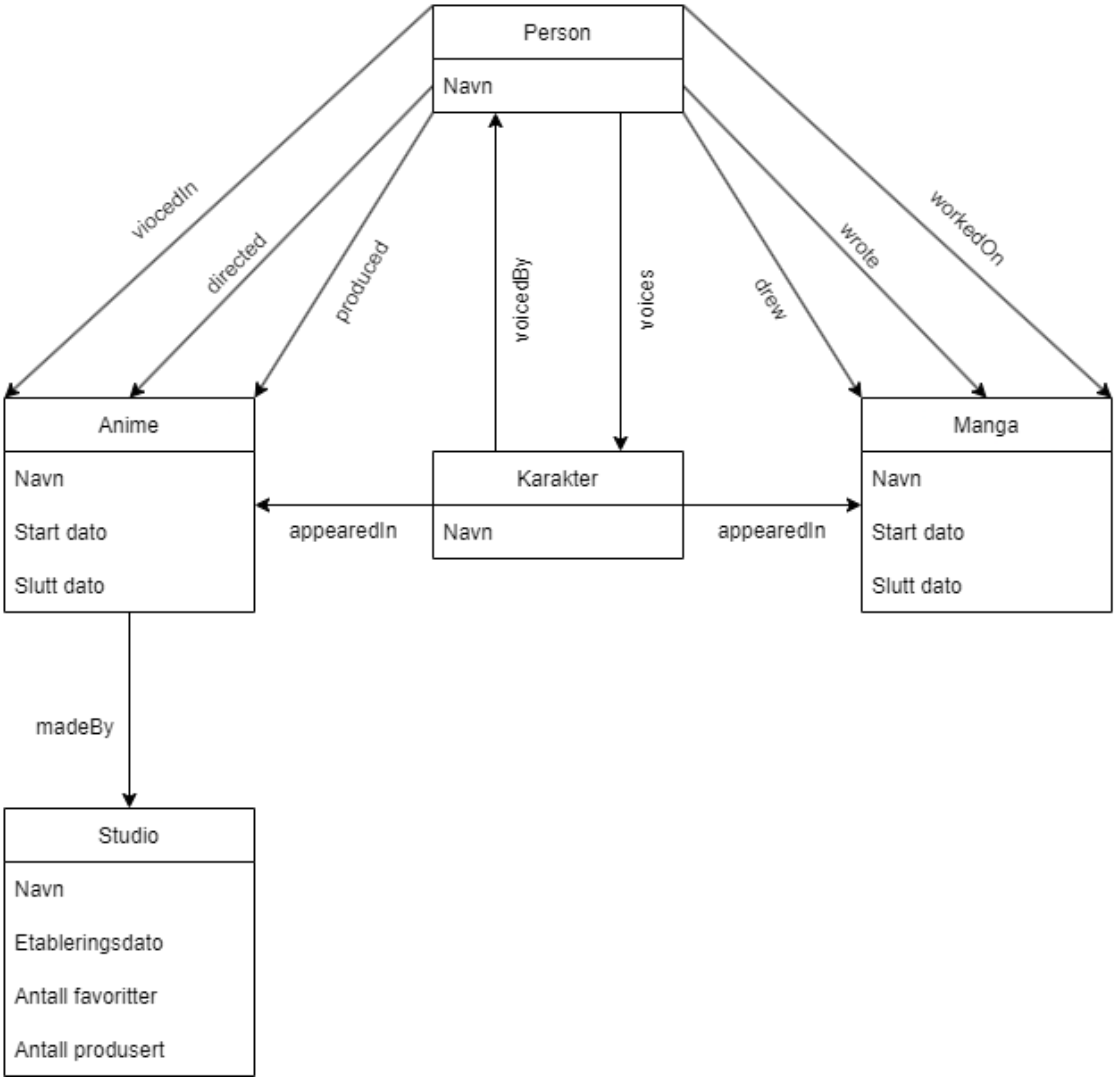
Datasettet vil bli lagret som sett med tripler i form av SPO (subjekt, predikat, objekt). Før dette ble gjort var det nødvendig å definere viktige predikater mellom dataene. I Tabell 1 vises alle predikatene som er definert.

S	**P**	**O**
Anime	madeBy	Studio
Anime	hasGenre	Genre
Person	directed	Anime
Person	produced	Anime
Person	voicedIn	Anime
Person	voices	Character
Person	drew	Manga
Person	wrote	Manga
Person	workedOn	Manga
Character	appearedIn	Anime
Character	appearedIn	Manga
Character	voicedBy	Person
Manga	hasGenre	Genre

Tabell 1: Predikater

Det finnes andre predikater som er mulig å definere, men som ikke ble valgt fordi de ikke var interessante å se på og ville bare føre til ett større datasett uten å være brukbare. F.eks. mellom Person og Anime kan man definere predikatene «ADR director», «Storyboard», «Key Animation». Disse kan brukes i et fremtidig større system som ser på flere predikater for å gi anbefalinger, men ble valgt bort for å ha et mer fokusert system.

Hvert objekt i datasettet inneholder informasjon som gir mer detaljer om objektet. Figur 2 viser hvilken informasjon hver entitet har samt relasjonene mellom klassene.



Figur 2: Diagram over klasser og relasjoner

3.2 Indeksering

Alle objekter i datasettet har en merkelapp som kan leses av mennesker. For alle objekter er dette egennavnet til objektet og det som vil være mest relevant når det skapes en spørring. Disse egennavnene blir indeksert for å gjøre det mulig med fulltekstsøk. Et av målene er å berike resultatene med resultater som kanskje ikke er direkte relatert til spørringen, men som fortsatt kan være relevant for personen som utfører spørringen. Fordi alle triplene danner en graf kan et mål på hvor viktig en node er være hvor mange kanter som er koblet til noden. Ved å utnytte dette kan en spørring som returnerer en node A også returnere node B som har en kant til A, og settet $\{C, D, E\}$ som også har kanter til B og kan derfor også være relevant til spørringen. Et eksempel: En spørring etter animen «No Game No Life». Fra grafen ser man at den er produsert av Shou Tanaka. Tanaka har også produsert mange

andre anime og har derfor mange kanter, som kan være et tegn på hans kvalitet som produsent. Resultatet fra spørringen kan da inneholde No Game No Life, Shou Tanaka, og et utvalg andre animer han har produsert. Disse koblingene kan brukes til å gi en node i grafen en rangering basert på hvor mange kanter den har inn og/eller ut og denne rangeringen kan så indekseres for å lett finne noder med høyest rangering for å berike resultatene.

3.3 Søk

For fulltekstsøk finnes flere metoder for hvordan teksten brukes i søket. En metode er å se etter alle dokumenter som inneholder søketermet, så hvis spørringen er «se alle» så returneres alle dokumenter som inneholder «se» og/eller «alle». Dette vil gi veldig mange resultater hvor ikke alle er relevant for hva brukeren faktisk leter etter. Dette kan også hjelpe med utforskendesøk. Hvis en bruker å finne anime med ett eller flere spesifikke ord i tittelen kan de søke etter ordet og få alle resultater som passer. Dette kan gi de andre titler og ord som virker interessante og leder de til ett nytt søk med nye ord. Detter passer til flere av kriteriene for utforskende søk definert i (Palagi mfl. 2017), blant annet:

- «En utviklende søke prosess» fordi brukeren endrer eller spesifiserer målet med søket ved å endre på spørringen.
- «Flere mulige svar» fordi brukeren har et vagt mål kan de ende opp med flere biter av relevant informasjon som hjelper videre i søkeprosessen i stedet for ett enkelt svar.
- «Flere mål med søket» brukeren kan ha et vagt mål med flere mindre mål som kan endre seg gjennom søkeprosessen for å oppnå det opprinnelige målet.

En annen mer spesifikk måte å søke på er å finne alle dokumenter som inneholder et spesifikt uttrykk. Dette vil si at hvis spørringen er «sa alle» så vil det kun returneres dokumenter som inneholder det nøyaktige uttrykket «sa alle». Dette passer når brukeren vet nøyaktig hva de skal se etter og dette vil gi brukeren forslag nøyaktig basert på det de søker etter.

4 Implementasjon

4.1 Innhenting av data

For å hente data fra MyAnimeList (MAL) som skulle brukes i datasettet ble det satt opp et Python-script som kommuniserte med MAL sitt API. Fra dette API-et var det mulig å hente informasjon om alle anime og manga som MAL hadde i sin database. MAL sitt API hadde ikke endepunkter for å hente informasjon om studioer, personer, eller karakterer så for å få tak i denne dataen var det nødvendig å bruke ett tredjeparts API. API-et som jeg brukte var Jikan API som er et uoffisielt API for MyAnimeList. Dette API-et ble valgt fordi det hadde støtte for alle endepunktene som nødvendige og hadde grundig dokumentasjon av API-et. Fordi dette er et tredjeparts API laget av frivillige er det en grense på 60 forespørsler som kan gjøres i løpet av et minutt. Dette førte til at innhenting av data gjennom dette API-et tok lengre tid enn forventet. Innhentingen tok også lengre tid på grunn av ustabilitet i API-et som førte til at mange forespørsler ikke ble besvart eller returnerte ingen data. Forespørslene som ble besvart uten noe data førte også til hull dataen, da spesielt og mest merkbart på personer. Fordi id-ene som ble brukt som primærnøkkel i databasen kom rett fra MAL og var sekvensielle var det mulig å finne alle id-ene som manglet og sende nye forespørsler for dem.

Datasettet som ble hentet fra MAL ble midlertidig lagret i en PostgreSQL database der data ble hovedsakelig lagret som JSON slik det kom fra kilden. PostgreSQL databasen ble bare brukt som mellomlagring for å unngå at data kunne gå tapt som følge av diverse årsaker. Av den grunn ble det satt opp et enkelt schema uten relasjoner mellom tabellene. Nyttig informasjon som id-en til objektet på MAL og navn ble hentet ut fra JSON objektet og lagret i tabellene sammen med resten av JSON objektet. PostgreSQL ble satt opp i en Docker container for å simplifisere oppgavene med å starte og stoppe databasen. Dette hjalp også i begynnelsen med å gjøre det enkelt å slette hele databasen og begynne på nytt de få gangene i starten hvor det var den enkleste løsningen på et problem. Det ble annen hver dag tatt backup av databasen og backupen ble lagret lokalt og i en sky for redundans.

4.2 Omgjøring til RDF

For å transformere datasettet til RDF tripler ble det laget et Python-script. Biblioteket RDFLib ble brukt og det lar deg definere egne RDF tripler med data som du har tilgjengelig. RDFLib gjør dette ved at det først lages et graf-objekt som det så legges noder og kanter til i form av RDF tripler. Hvert subjekt og objekt har

sin egen unike IRI. Denne IRI-en er bygget opp av et domene og en unik id, f.eks. «http://example.org/anime/5206». For hver klasse som ble definert fra datasettet ble et eget domene opprettet, f.eks. «http://example.org/people/» for klassen Person og «http://example.org/manga/» for klassen Manga. Scriptet som går gjennom all data fra relasjonsdatabasen tar alle klassene hver for seg. Prosessen er lik for alle klassene og forskjellene er hvilke tripler som legges til grafen og informasjonen de inneholder. For å forklare prosessen brukes klassen Anime som eksemplet:

1. Alle rader fra tabellen til klassen Anime hentes fra relasjonsdatabasen
2. Følgende gjentas for alle radene, en rad representerer en anime:
3. En IRI (animeIRI) lages for denne animen med domenet og den unike id-en fra MAL
4. Tripler for informasjon som klasse, navn, og start- og sluttdato legges til
5. For hver sjanger som ligger i JSON objektet:
   (a) Det lages det en IRI (genreIRI) med domenet til sjanger og navnet på sjangeren
   (b) En trippel **animeIRI hasGenre genreIRI** legges til grafen
6. For hvert studio i JSON objektet:
   (a) Det lages en IRI (studioIRI) for med domenet for studio og den unike id-en til studioet
   (b) En trippel **animeIRI madeBy studioIRI** legges til grafen

Når alle tabellene er blitt gått gjennom har man en graf som består av alle RDF triplene som ble definert for datasettet. Denne grafen lagres så til en fil med syntaksen Turtle for å kunne brukes i en egen database. Turtle er en syntaks for RDF som gjør det mulig å beskrive en RDF graf på en kompakt måte med naturlig tekst (Beckett mfl. 2014). Turtle ble brukt fordi det bruker naturlig tekst og dette simplifiserer oppgaven med å identifisere feil som oppsto i prosessen av å transformere data fra MAL til RDF tripler. Listing 2 viser data beskrevet med Turtle.

```
@prefix anime: <http://example.org/anime/> .
@prefix character: <http://example.org/character/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .

anime:15607 a dbpedia:Anime ;
    rdfs:label "Ai no Wakakusa Monogatari Specials";
    anime:ended "2001" ;
    anime:hasGenre genre:Drama,
        genre:Historical,
        genre:Slice_of_Life ;
    anime:madeBy studio:22 ;
    anime:started "2001" .

anime:21509 a dbpedia:Anime ;
    rdfs:label "Trip Trek (2010)" ;
    anime:ended "2011-01-26" ;
    anime:hasGenre genre:Adventure,
        genre:Comedy,
        genre:Fantasy ;
    anime:madeBy studio:32 ;
    anime:started "2010-11-10" .
```

Listing 2: Data beskrevet med Turtle

4.3 Behandling av RDF

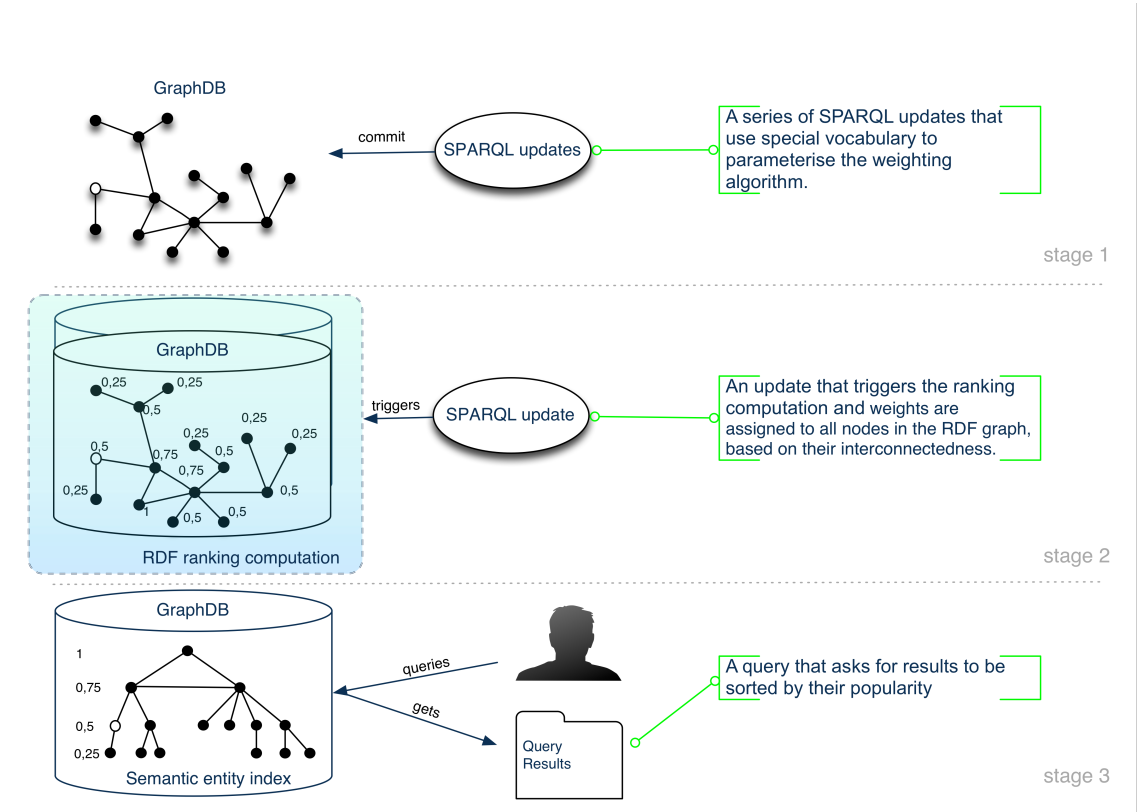
For å bruke og behandle RDF grafen ble GraphDB valgt som database. GraphDB ble valgt fordi det er en database som er designet for å brukes med RDF grafer. GraphDB bruker SPARQL som spørrespråk som er utviklet for å brukes på RDF (Harris og Seaborne 2013). Et eksempel på en SPAQLE spørring vises i Listing 3. Det er flere andre nyttige funksjoner som støttes direkte av GraphDB som er viktige for at systemet som utvikles skal fungere som ønskelig.

```
PREFIX onto: <http://www.ontotext.com/>
PREFIX rank: <http://www.ontotext.com/owlim/RDFRank#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX people: <http://example.org/people/>

SELECT ?director ?dname ?entity ?value ?rankDir ?rank0 WHERE{
    ?director people:directed ?entity .
    ?director rdfs:label ?dname .
    ?director rank:hasRDFRank5 ?rankDir .
    {
     SELECT ?entity ?value ?rank0 {
            ?value onto:fts '"no game no life"' .
            ?entity rdfs:label ?value .
            ?entity rank:hasRDFRank5 ?rank0 .
        }
    }
}
```

Listing 3: Eksempel på SPARQL spørring som kan bli konstruert av backend 4.4.1

GraphDB har blant annet direkte støtte for fulltekstsøk ved å indeksere IRI-er og «literals» som navn. Det har også støtte for det som kalles *RDF Rank* som gir en rangering til alle nodene i en graf. Rangeringen er basert på hvor mange innkommende kanter en node har og rangeringen til andre noder som har en kant til noden, samme prinsipp som PageRank (Ontotext 2023). Nodene får en rangering mellom 0 og 1. Rangeringene til nodene blir mer spredt ut hvis algoritmer kjøres flere ganger. Standardinnstillingen til RDF Rank er å kjøre algoritmen 20 ganger. Dette førte opprinnelig til at noen noder endte opp med en veldig høy rangering (0.834) mens andre noder fikk så lav rangering at den var nesten lik 0. For å løse dette ble innstillingen til RDF Rank satt opp til 60 som førte til bedre rangering av spesielt anime. Hvor mange kanter en node har kan være et tegn på hvor viktig eller populær den noden er. Figur 3 viser hvordan rangering blir gjort i GraphDB.

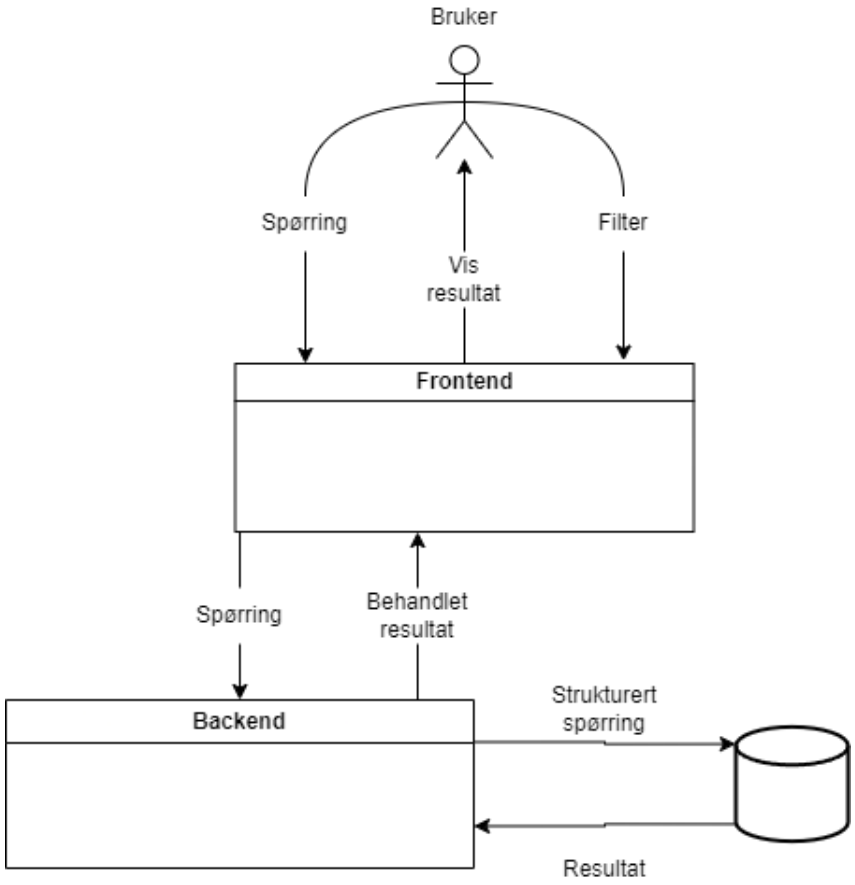


Figur 3: Illustrasjon av hvordan RDF Rank brukes i GraphDB. Kilde:Ontotext

Fulltekstsøket som er implementert i GraphDB tillater flere måter å søke på. De som ble valgt for dette systemet var søk på termer og uttrykk som er beskrevet i 3.3. Ved å implementere disse to metodene for å gjøre fulltekstsøk blir det mulig å både gjøre «lookup» søk og utforskende søk (2.5).

4.4 Systemet

Systemet som brukeren samhandler med består av to deler, ett brukergrensesnitt som mottar søkeord og en backend som håndterer spørringer. Figur 4 viser et overblikk av systemet.



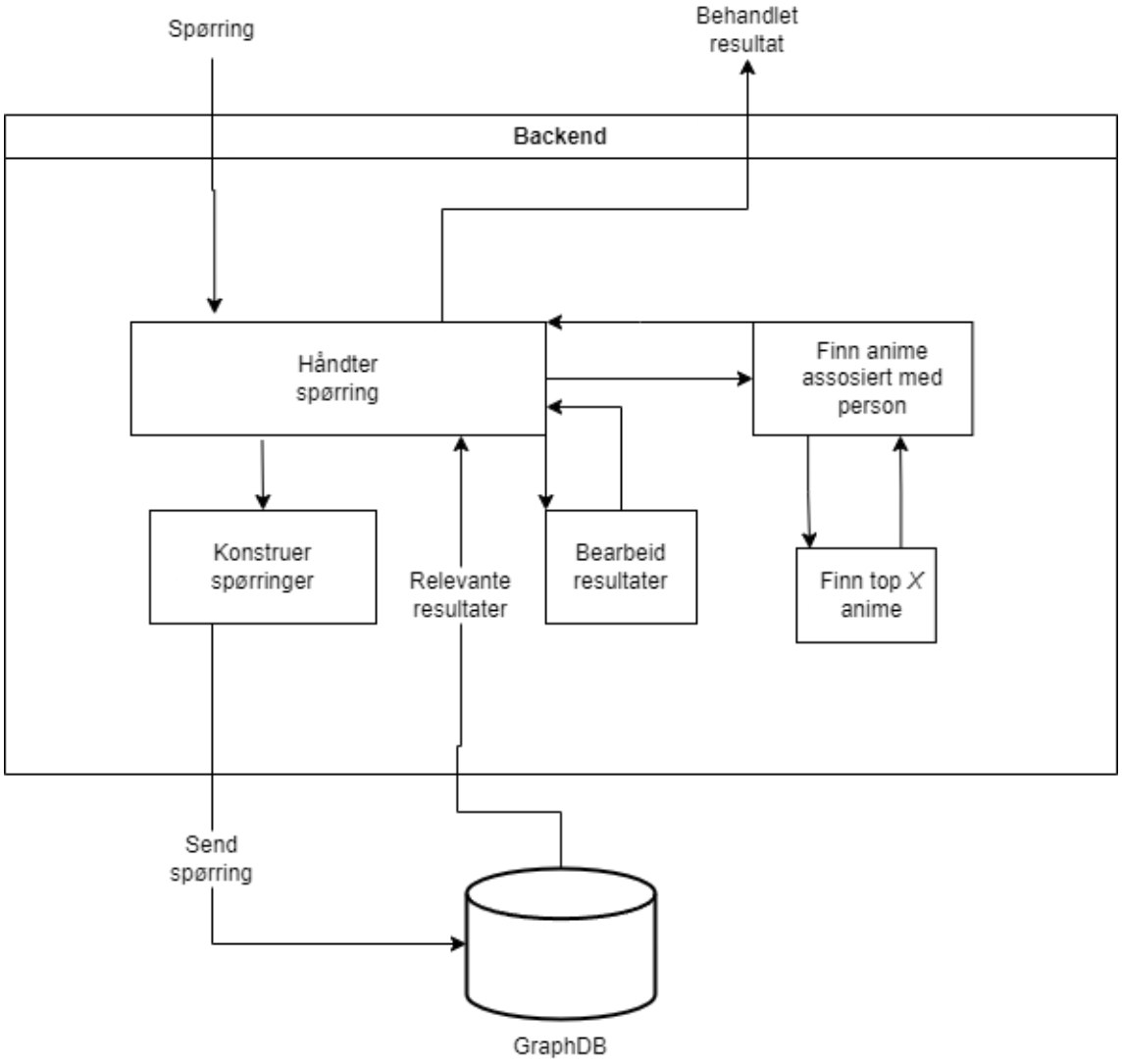
Figur 4: Overblikk av systemet

4.4.1 Backend

Backend delen av brukergrensesnittet er skrevet i Java. Denne delen er ansvarlig for å kommunisere med grafdatabasen, gjøre spørringer for brukeren, og håndtere resultatene som kommer fra databasen. Figur 5 viser et overblikk av backend.

Backend delen mottar søkeord fra brukergrensesnittet og konstruerer en SPARQL spørring som inneholder søkeordet og sender spørringen til grafdatabasen. En bruker kan velge om de vil søke med termer eller uttrykk og produsent eller regissør, så spørringen spesialiseres etter hva brukeren er ute etter. Ved søk på termer kan de boolske operatorene **AND**, **OR**, og **NOT** brukes for mer komplekse uttrykk. Den ferdige spørringen sendes til databasen som returnerer resultatet som kom fra spørringen.

Resultatet av spørringen er en liste med elementer, hvor hvert element inneholder en IRI til en anime, navn på animen, rangeringen til animen, IRI til en person, navn på personen, og rangeringen til personen. Elementene blir gått gjennom en etter en og det lages et objekt for hver anime og person som kobles til hverandre. Det lages



Figur 5: Overblikk av backend.

et anime objekt for hver unike anime IRI og objektene samles i en liste, *resultList*. En person kobles til den anime de var i samme element med.

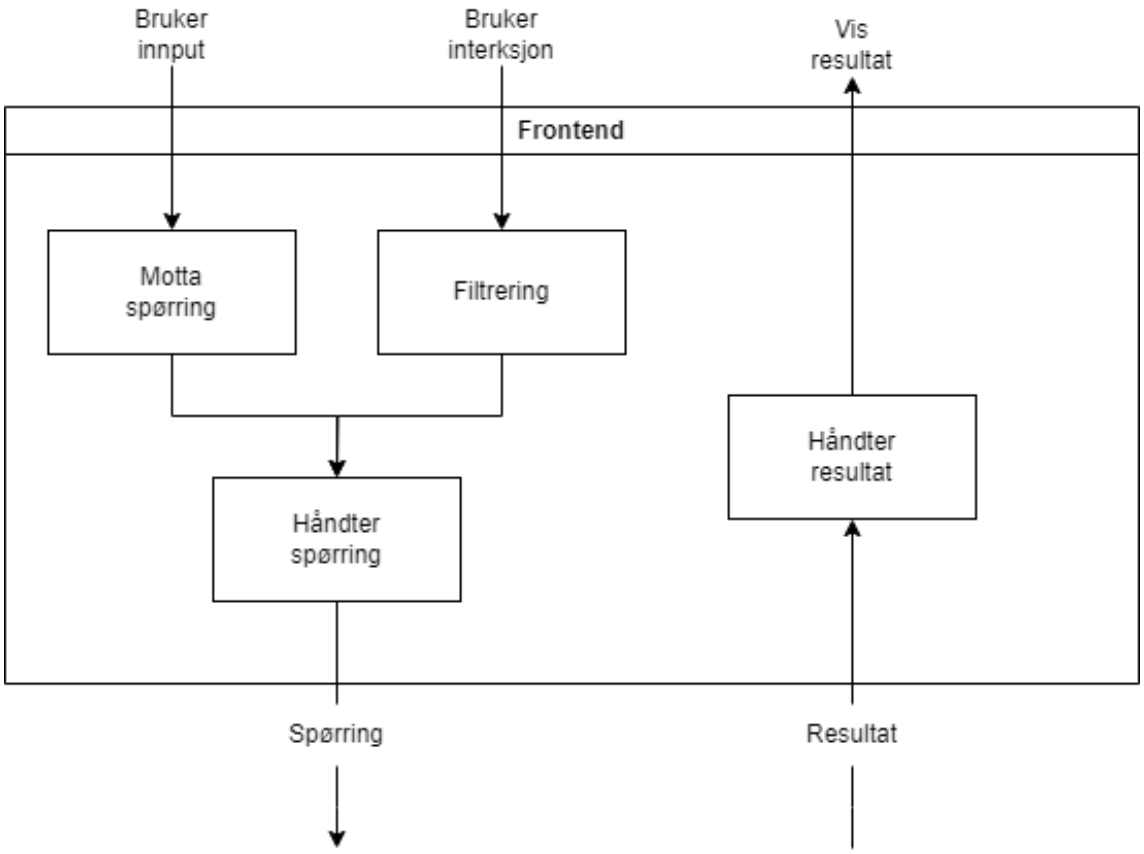
Hver anime kan være assosiert med mange personer så brukeren har muligheten til å bestemme hvor mange personer per anime som skal vises. Vi er ute etter å finne de personene som har høyest rangering for å bruke de til å anbefale andre anime de har vært involvert i. Listen med all anime som kommer fra resultatet blir gått gjennom og personene som er koblet til en anime blir sortert etter rangering og de topp X øverste blir valgt ut. Man sitter igjen med en liste med anime hvor bare de topp X øverste personene er representert per anime.

Når de personene med høyest rangering per anime har blitt funnet kan disse brukes til å finne andre animer de har vært involvert i. For hver anime og for hver person assosiert med den animen lages det en spørring som finner all anime som er produsert eller regissert av den personen, sorterer dem etter rangering, og begrenser resultatet til de topp X animene med høyest rangering. Resultatet blir en liste med animer. På backend lages det objekter for disse animene som kobles til personene.

For å vise resultatet av søkene går man gjennom listen med anime, *resultList*, og for hver anime og person lages det et rad-objekt som inneholder navnet på animen, navn på personen, og en liste med navn på andre anime som personen har produsert/regissert. Disse radene lagres i en liste som brukes av frontend for å vise resultatet til brukeren.

4.4.2 Brukergrensesnitt

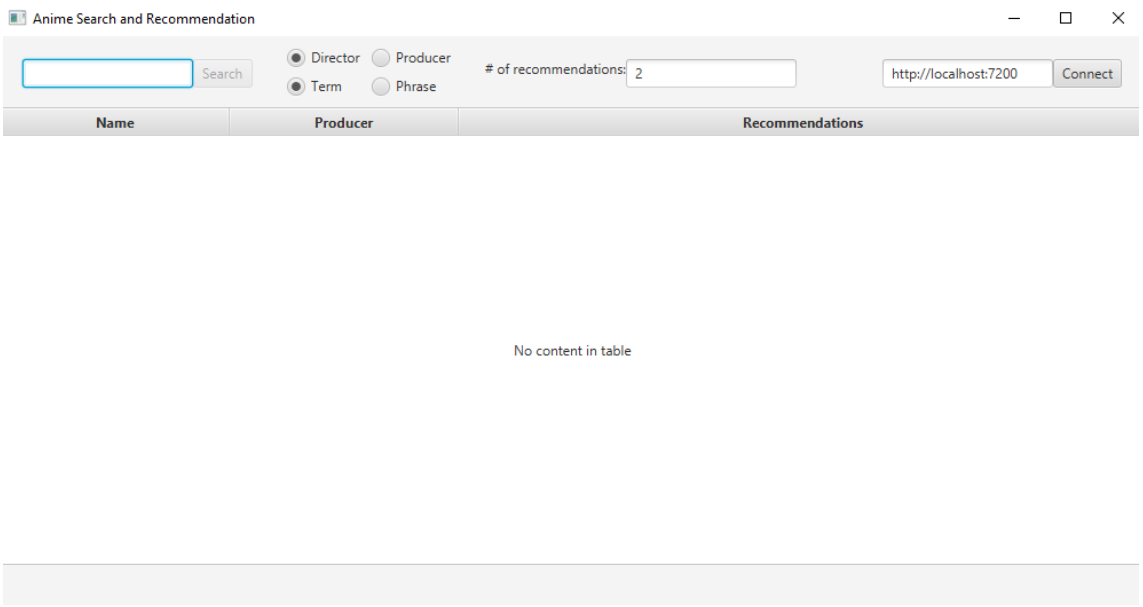
Frontenden som brukeren samhandler med, er også lager i Java med JavaFX. Figur 6 viser et overblikk av funksjonene til frontend.



Figur 6: Overblikk av funksjonene til frontend

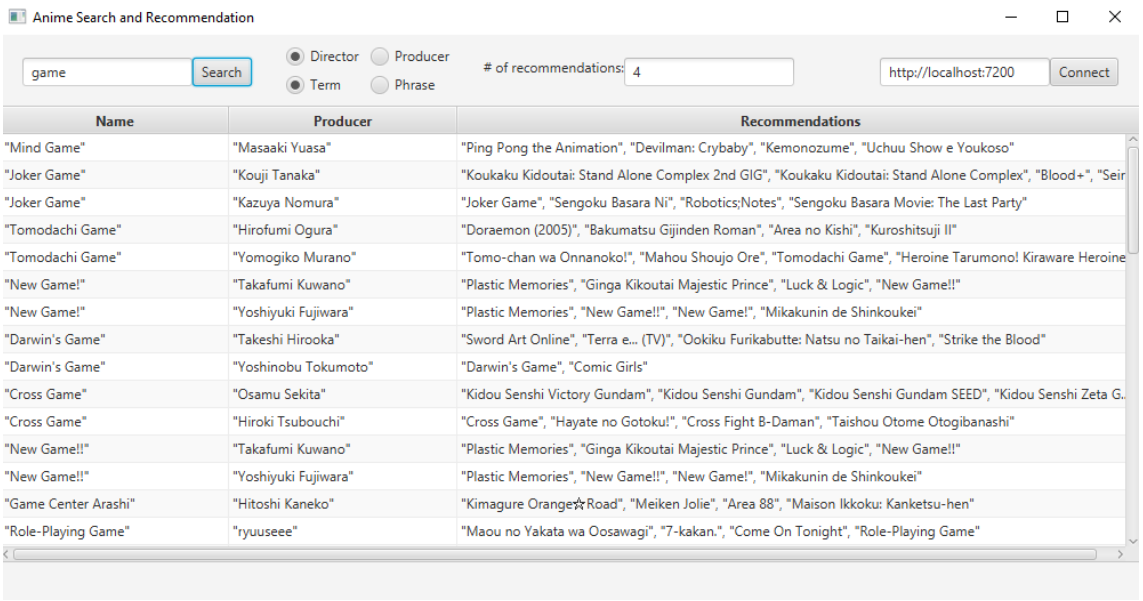
Det er et enkelt brukergrensesnitt som skal tillate brukeren å søke etter anime og få forslag basert på hva de søker etter. Som vist på Figur 7 så har frontenden disse elementene:

- Et felt for å skrive inn søkeordene med en knapp for å initiere søket.
- Fire radioknapper hvor to og to er koblet sammen, henholdsvis «Director» og «Producer», og «Term» og «Phrase».
- Et felt for å skrive inn hvor mange anbefalinger man ønsker.
- Et felt for å skrive inn adressen til databasen med en knapp for å koble seg til.
- En tabell hvor resultatene fra et søk vil vises til brukeren.



Figur 7: Frontend før det er foretatt ett søk

Når en bruker har skrevet inn søkeordene sine og valgt de innstillingene de ønsker så sendes dette til backend som foretar søket i databasen som beskrevet i 4.4.1. Når backend er ferdig med å prosessere søket havner resultatene i en liste som brukes av frontend til å fylle inn tabellen for å vise resultatene. Figur 8 viser brukergrensesnittet etter et søk på ett term har blitt utført.



Figur 8: Brukergrensesnitt fylt med resultater

For å prøve å integrere utforskende søk brukes radioknappene beskrevet over. Ønsker en bruker å utforske anime og anbefalinger eller ikke er helt sikre på hva de leter

etter kan de søke med termer ved å velge «Term» kappen. Dette kan gi dem mange resultater i starten hvor mange kan være unyttig mens andre resultater gir de ny informasjon. Denne informasjonen kan brukes til å endre på søket ved å legge til nye termer og boolske uttrykk. Muligheten til å søke på produsent eller regissør er også til stede for å hjelpe brukere å finne informasjon som kan gjøre et vagt mål mer presist. Disse sammen med muligheten til å velge antall forslag er der for å oppnå kriteriene for utforskende søk spesifisert i 3.3.

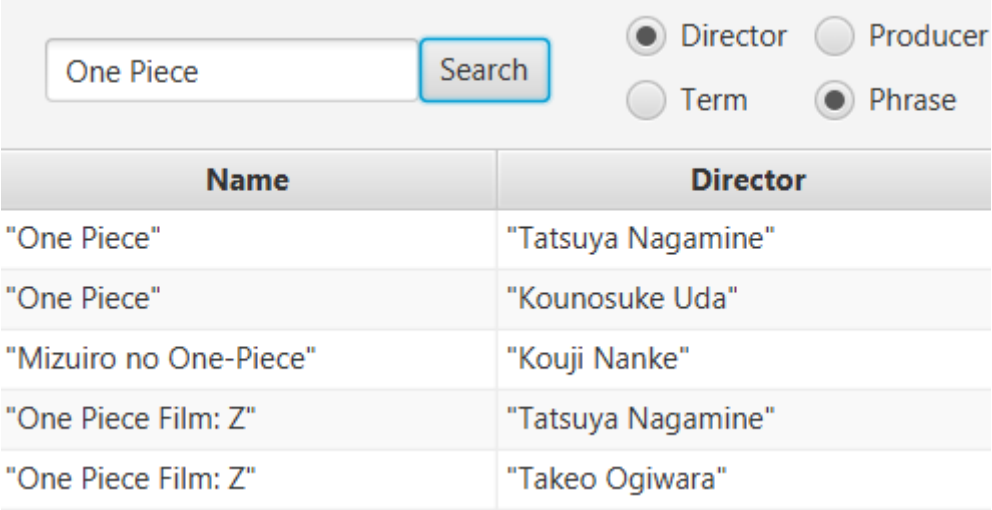
5 Evaluering og Diskusjon

5.1 Evaluering

For å evalueringen av systemet er målet å vise at løsningen fungerer. For å gjøre dette presenteres det her noen eksempelsøk og resultater.

All evalueringen har vært gjort uten brukertesting på grunn av tidsbegrensninger. Det er derfor ikke klart hvordan en bruker som ikke er kjent med systemet ville ha evaluert det. Mangelen på brukertesting har derfor ført til at systemet bare har blitt evaluert av meg, som allerede kjenner til systemet godt og hvordan det brukes.

Eksempel 1 En bruker kjenner til en anime som de liker og er interessert i å finne ut hvem som er regissør og/eller produsent for den. De gjør dette ved å bruke navnet på animen og radioknappene for å velge uttrykk (phrase) og regissør og/eller produsent. Figur 9 viser et søk etter regissøren på animen «One Piece».

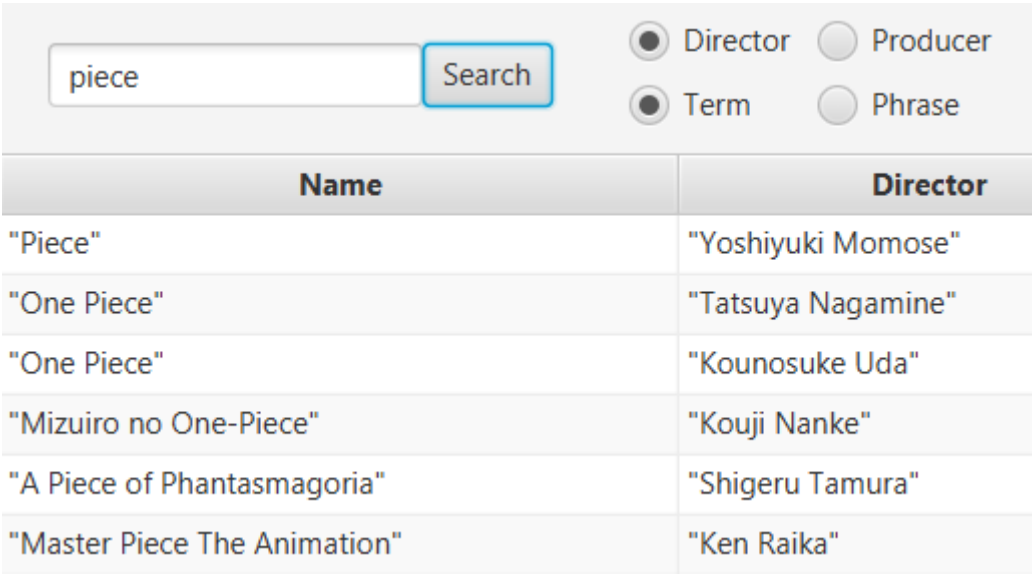


Name	Director
"One Piece"	"Tatsuya Nagamine"
"One Piece"	"Kounosuke Uda"
"Mizuiro no One-Piece"	"Kouji Nanke"
"One Piece Film: Z"	"Tatsuya Nagamine"
"One Piece Film: Z"	"Takeo Ogiwara"

Figur 9: Søk etter regissør ved kjent navn på anime

Resultatet fra søket i Figur 9 viser at systemet finner animen «One Piece» og viser to av regissørene som har vært involvert. Resultater viser også andre anime som inneholder «One Piece» i tittelen. Ved søk som bruker et uttrykk får man alle resultat som inneholder dette uttrykket. I dette tilfellet er det filmer som tilhører samme serie som animen det ble søkt etter.

Eksempel 2 En bruker ønsker å finne regissør og/eller produsent for en anime de har sett før. I dette tilfellet vet brukeren bare et ord fra tittelen på animen. Søket blir da gjort ved å bruke ordet og velge «Term» på radioknappene. Figur 10 viser deler av resultatet fra et slik søk.

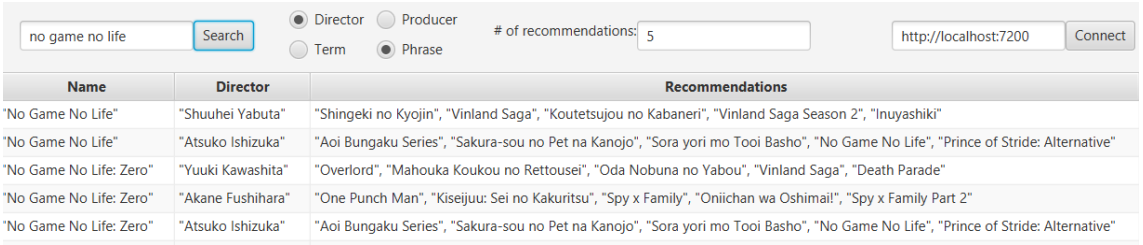


Figur 10: Resultat fra søk når bare deler av tittelen er kjent

Resultatet viser at om man var ute etter «One Piece» men bare kjente til «Piece» så finner man ønsket resultat men det kommer også mange andre resultater som ikke er interessante for brukeren. I dette tilfellet havnet «One Piece» langt oppe i resultatene men det er mulig at ønsket resultat havner langt nede og forsvinner i «støyen». Med mindre det kjente ordet fra tittelen er mer unikt så vil denne typen søk ikke gi veldig gode resultat. Et eksempel som er vist i Figur 11 er hvis man bare husker ordet «Boku» fra «Boku no Hero Academia» så er det ønskede resultatet vanskeligere å finne.

Figur 11: Søk etter vanlig term, ønsket resultat merket med rødt.

Eksempel 3 En bruker kjenner til en anime de liker og ønsker å finne flere anime fra samme regissør. Dette gjøres ved å bruke navnet på animen brukeren kjenner til, velge «Director» og «Phrase» på radioknappene og velge hvor mange anime fra samme regissør som skal vises. Figur 12 viser resultatet fra et slik søk.

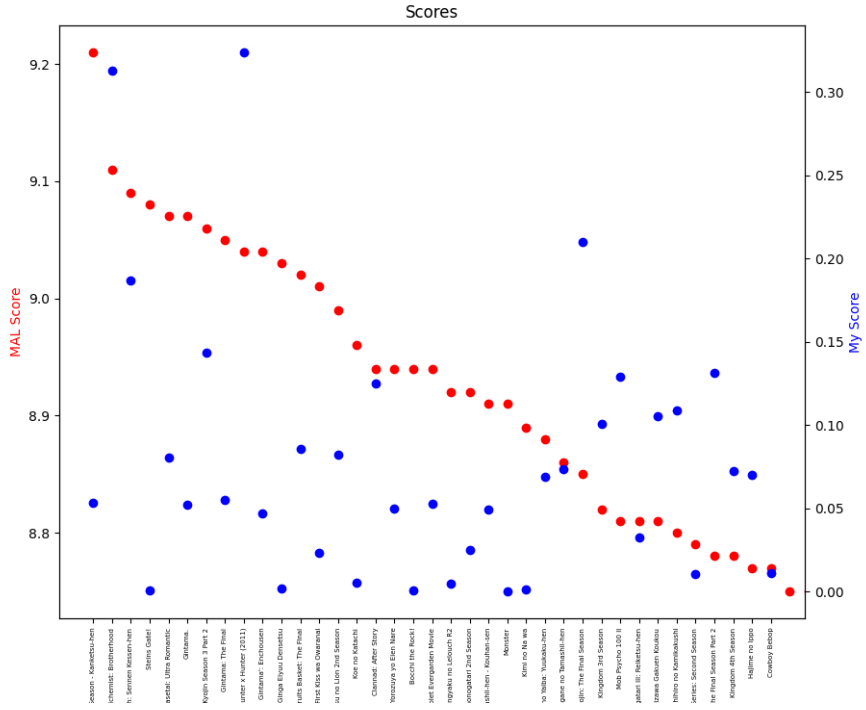


Figur 12: Resultat fra søk på anime med 5 anbefalinger per regissør

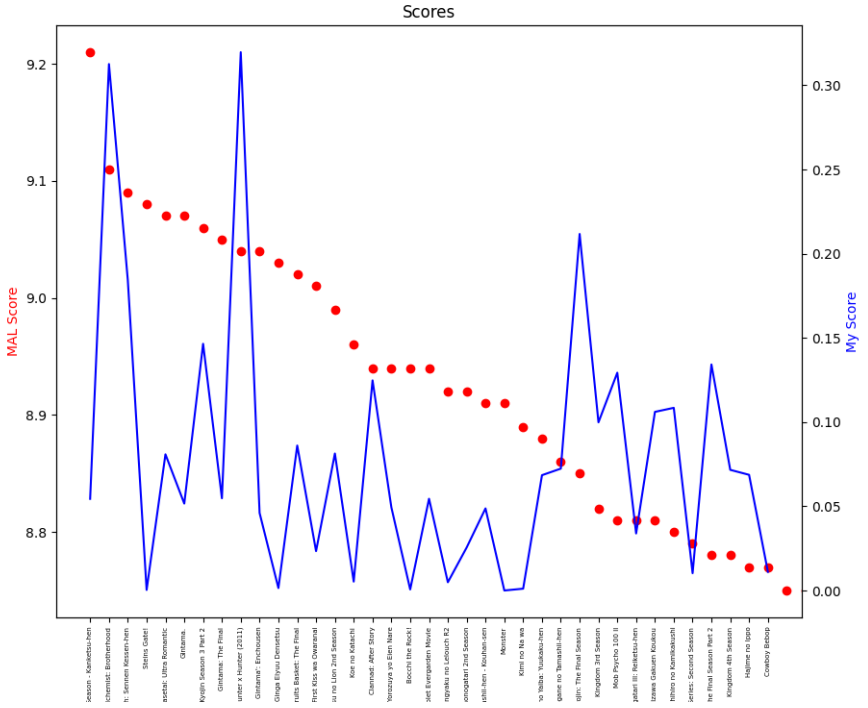
I resultatet vises fem anime for hver regissør. Dette er de fem animene som har høyest rangering i databasen per regissør. Rangeringen har litt blandet resultat hvis vi sammenligner med MyAnimeList. For Shuuhei Yabuta er både «Shingeki no Kyojin» og «Vinland Sage» på topp 5 i rangeringen av anime han har vært involvert i. Men for Atsuko Ishizuka er det kun en.

Eksempel 4 For å se på rangeringen som har vært gjort på mitt datasett har jeg sammenlignet rangeringen til de 40 animene på MyAnimelist som har høyest score med scoren de har i mitt system. Rangeringene på MyAnimeList kommer fra en

gjennomsnittlig score gitt av brukere. I dette systemet kommer rangeringen fra en score kalkulert med RDF Rank algoritmen i GraphDB som beskrevet i 4.3. Figurene 13 og 14 viser grafer som sammenligner rangeringene. MyAnimeList er merket med rød og min score er merket med blå.



Figur 13: Rangering av anime på MyAnimeList sammenlignet med rangeringen i systemet



Figur 14: Samme graf som Figur 13 men viser bedre forskjellene i score mellom animer

Som man kan se av grafene er scorene og derfor rangeringene på animer i mitt system veldig spredt. Dette viser at det er stor forskjell i hva RDF Rank ser på som viktig og hva brukere av MyAnimeList mener er viktig/bra. Denne forskjellen stammer nok fra at en bruker bare ser på animen i seg selv når de skal gi den en score. RDF Rank på den andre siden rangerer alle nodene ikke bare animer og tar i betraktning scoren til andre noder når den skal gi en node en score. Noen av animene har lik rangering, men fordi de er få og de andre animene er så spredt tyder dette på at likheten i rangering bare er tilfeldig. En bruker som bruker systemet for å få forslag til animer laget av en regissør/produsent vil ikke få de samme forlagene som en bruker som går inn på MyAnimeList og ser på topplisten.

Fordi rangeringen er kalkulert på forhånd så kan ikke dette systemet brukes som et anbefalingssystem. Alle søk som er like, vil gi de samme forslagene på andre animer. Det bygges ikke opp noen slags profil om en bruker så ingen personlige anbefalinger kan gjøres.

Eksempel 5 Systemet støtter også boolske uttrykk som kan brukes sammen med søke-termer. Hvis vi ser på eksempel 2 hvor en bruker bare vet ett eller noen få ord fra tittelen på en anime, så kan de med boolske uttrykk enklere finne det de leter etter. Så hvis en bruker bare husker «Boku» fra «Boku no Hero Academia» kan de

først bare søke med det og så bruke uttrykkene *AND* og *NOT* til å fjerne titler som ikke er relevant. Figur 15 viser et eksempel på dette.

Figur 15: Søk på termer med boolske uttrykk

Å bruke boolske uttrykk gjør det mulig for brukeren å fjerne uønskede resultater og gjør det enklere å finne resultatene man er interessert i. Men boolske uttrykk er ikke noe en vanlig bruker er vant til å bruke og derfor øker det vanskelighetsgraden av å konstruere et søk. Det vil være nødvendig å gi brukeren informasjon om hva boolske uttrykk er og hvordan de kan brukes.

5.2 Diskusjon

Disse resultatene viser at systemet fungerer bra til å søke med hvis man vet hele eller flere deler av en tittel. Hele eller store deler av tittelen gir nøyaktige resultater i form av nøyaktig det man søkte etter. Men fordi søket returnerer resultat som ikke eksklusivt inneholder søkeuttrykket kan noen av resultatene ikke ha en relasjon med det ønskede resultatet.

Resultatene har også vist at hvis det bare søkes etter en term, som når man ikke kjenner hele tittelen, så får man mange resultater som ikke nødvendigvis er nyttige. Men dette kan brukes til å gjøre utforskende søk hvor en bruker ønsker å finne anime som inneholder spesifikke termer i tittelen og utvikle spørringen. Å søke med termer kan forbedres vet at brukeren benytter boolske uttrykk. Dette gir mer nøyaktige resultater for hva brukeren er ute etter ved å fjerne titler som inneholder termer brukeren ikke er interessert i. Men dette øker kompleksiteten til spørringen og en

vanlig bruker er ikke nødvendigvis kjent med hvordan man bruker boolske uttrykk.

Rangeringen som er gjort vil ikke kunne brukes til et anbefalingssystem. Men det ser ut til å fungere for å gi forslag til andre anime laget av regissører og produsenter. Et studio som lager mange animer og en regissør som har vært involvert i mange animer vil få en høy score. Det at begge partene har vært involvert i flere animer kan være et tegn på at animene blir populære og har suksess, og studioet og regissøren får muligheten til å jobbe på flere. Derfor kan det at en anime er laget et studio og en regissør med høy score da være et tegn på at animen har høy kvalitet.

Som nevnt er det ikke gjort brukertester, så det er ikke visst hvordan en bruker uten kjennskap til systemet vil evaluere det.

Bibliografi

Balog, Krisztian (2018). *Entity-Oriented Search*. eng. Cham.

Beckett, David mfl. (2014). «RDF 1.1 Turtle». I: *World Wide Web Consortium*, s. 18–31.

Dom, Byron mfl. (2003). «Graph-Based Ranking Algorithms for e-Mail Expertise Analysis». I: *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. DMKD '03. San Diego, California: Association for Computing Machinery, s. 42–48. ISBN: 9781450374224. DOI: 10.1145/882082.882093. URL: https://doi.org/10.1145/882082.882093.

Ferrández, Oscar mfl. (2009). «Addressing ontology-based question answering with collections of user queries». I: *Information Processing & Management* 45.2, s. 175–188.

Golovchinsky, Gene, Abdigani Diriye og Tony Dunnigan (2012). «The Future is in the Past: Designing for Exploratory Search». I: *Proceedings of the 4th Information Interaction in Context Symposium*. IIIX '12. Nijmegen, The Netherlands: Association for Computing Machinery, s. 52–61. ISBN: 9781450312820. DOI: 10.1145/2362724.2362738. URL: https://doi.org/10.1145/2362724.2362738.

Harris, Steven og Andy Seaborne (mar. 2013). *SPARQL 1.1 Query Language*. W3C Recommendation. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/. W3C.

Kaufmann, Esther, Abraham Bernstein og Lorenz Fischer (2007). «NLP-Reduce: A naive but domainindependent natural language interface for querying ontologies». I: *4th European Semantic Web Conference ESWC*. Springer Berlin, s. 1–2.

Kiryakos, Senan og Magnus Pfeffer (2021). «The Benefits of RDF and External Ontologies for Heterogeneous Data: A Case Study Using the Japanese Visual Media Graph». I: *Information between Data and Knowledge*. Bd. 74. Schriften zur Informationswissenschaft. Session 5: Knowledge Representation. Glückstadt: Werner Hülsbusch, s. 308–320. URL: https://epub.uni-regensburg.de/44950/.

Kotkov, Denis, Shuaiqiang Wang og Jari Veijalainen (2016). «A survey of serendipity in recommender systems». I: *Knowledge-Based Systems* 111, s. 180–192. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2016.08.014. URL: https://www.sciencedirect.com/science/article/pii/S0950705116302763.

Ontotext (2023). *Ranking Results — GraphDB 10.2.0 documentation*. URL: https://graphdb.ontotext.com/documentation/10.2/ranking-results.html#rdf-rank (sjekket 31. mai 2023).

Palagi, Emilie mfl. (2017). «A Survey of Definitions and Models of Exploratory Search». I: *Proceedings of the 2017 ACM Workshop on Exploratory Search and Interactive Data Analytics*. ESIDA '17. Limassol, Cyprus: Association for Com-

puting Machinery, s. 3–8. ISBN: 9781450349031. DOI: 10.1145/3038462.3038465. URL: https://doi.org/10.1145/3038462.3038465.

Prud'hommeaux, Eric mfl. (sep. 2012). *A Direct Mapping of Relational Data to RDF*. W3C Recommendation. https://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/. W3C.

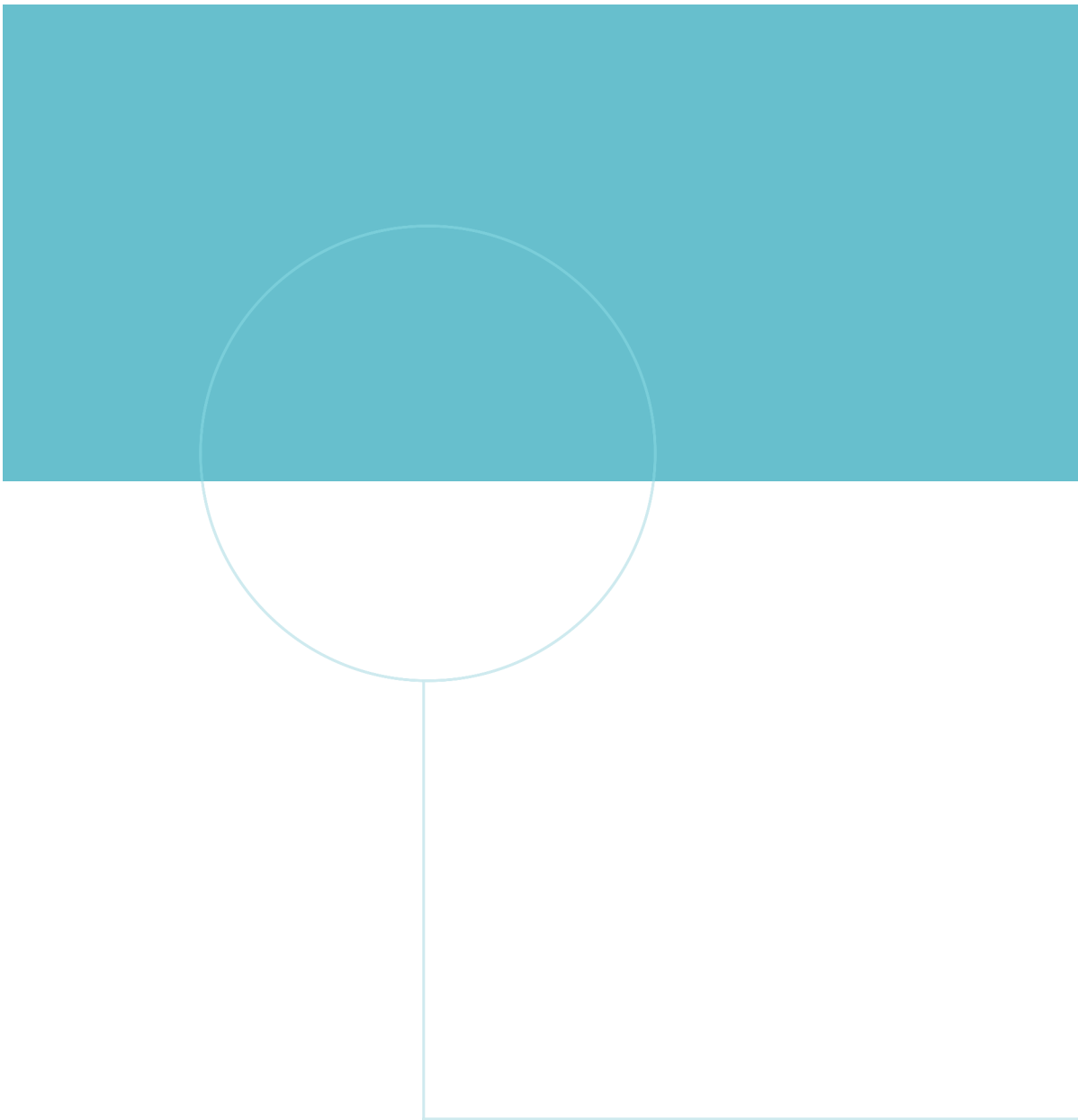
Sundara, Seema, Souripriya Das og Richard Cyganiak (sep. 2012). *R2RML: RDB to RDF Mapping Language*. W3C Recommendation. https://www.w3.org/TR/2012/REC-r2rml-20120927/. W3C.

Tablan, Valentin, Danica Damljanovic og Kalina Bontcheva (2008). «A natural language query interface to structured information». I: *The Semantic Web: Research and Applications: 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008 Proceedings 5*. Springer, s. 361–375.

Toms, Elaine G mfl. (2000). «Serendipitous Information Retrieval.» I: *DELOS*. Citeseer.

Wood, David, Richard Cyganiak og Markus Lanthaler (feb. 2014). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. W3C.

Zevio, Stella, Tetsuya Mihara og Shigeo Sugimoto (nov. 2018). «Identifying Transmedia Works from User-Generated Knowledge Bases : Japanese Pop Culture Study Case». I: *The Fifth International Workshop on Practical Application of Ontology for Semantic Data Engineering (PAOS2018)*. Awaji, Japan. URL: https://hal.science/hal-02004676.



NTNU

Kunnskap for en bedre verden