Amir Dahina

Web-käyttöliittymä Watson Discovery -alustalle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto ja viestintätekniikka

Insinöörityö

15.5.2024

Tiivistelmä

Tekijä:	Amir Dahina
Otsikko:	Web-käyttöliittymä Watson Discovery -alustalle
Sivumäärä:	26 sivua
Aika:	15.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander

Projektin aiheena oli luoda käyttöliittymä IBM:n Watson Discovery -haku- ja tekstianalyysialustalle. Sovelluskokonaisuuden tarkoituksena oli toimia hakukoneena, jonka avulla käyttäjät voivat hakea haluamaansa tietoa laajasta dokumenttivalikoimasta. Projekti tuottaisi web-sovelluksen, joka on rakennettu käyttäen teknologioita React, Express, Watson Discovery ja Node.js. Sovelluksen on tarkoitus toimia MVP:nä (Minimum Viable Product) eli käytännössä todentaa Watson Discovery -palvelun kyvykkyyksiä.

Suunnitteluvaiheessa projektiin tuli huomattavan paljon erilaisia vaatimuksia, joista laadittiin toiminnallisuuskaaviot. Sovelluskehitysvaiheen alussa suuri osa ajasta meni teknologioiden opetteluun ja niihin perehtymiseen. Kehittämisessä oli tarkoitus käyttää mikropalveluarkkitehtuurilähestymistapaa. Frontend tuotettiin käyttäen Reactia ja Carbon Design System -teknologioita. Backend-puoli toteutettiin käyttämällä Node.js- ja Express-työkaluja.

Projekti tuotti vielä hyvin keskeneräisen kokonaisuuden, mikä vaatii vielä huomattavasti työtä kaikilta osa-alueiltaan. Suunnittelu auttaa projektia jatkavia tahoja edistämään sovellusta tai oppimaan hyviä käytäntöjä sovelluskehityksestä.

Avainsanat: React, Express, Watson Discovery, Käyttöliittymä, Sovelluskehitys, Carbon Design System

Abstract

Author:	Amir Dahina
Title:	Web User Interface for Watson Discovery Search and Text Analysis Platform
Number of Pages:	26 pages
Date:	15 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer

The project aimed to create a user interface for IBM's Watson Discovery search and text analysis platform. The overall purpose of the application was to serve as a search engine, allowing users to retrieve desired information from a vast collection of documents. The project would produce a web application built using technologies such as React, Express, Watson Discovery, and Node.js. The application is intended to function as an MVP (Minimum Viable Product), essentially validating the capabilities of the Watson Discovery service.

During the planning phase, the project had a significant number of diverse requirements, which were translated into functional diagrams. In the initial stages of application development, a substantial amount of time was devoted to learning and becoming familiar with the technologies. Microservices architecture was intended to be utilized in development. The frontend was developed using React and Carbon Design System technologies, while the backend was implemented using Node.js and Express tools.

The project delivered a relatively unfinished product, requiring considerable work in all areas. The planning phase will assist future developers in advancing the application and acquiring best practices in application development.
.

Keywords: React, Express, Watson Discovery, FrontEnd, Backend, Carbon Design System

Sisällys

Lyhenteet

API: Application Programming Interface. Ohjelmointirajapinnan avulla ohjelmat voivat tehdä pyyntöjä tai vaihtaa tietoa keskenään.

DOM: Document object model. Menetelmä strukturoitujen dokumenttien, kuten HTML- tai XML- rakennepuuna, jonka eri olioita voi hakea tai käsitellä JavaScript-komennoilla.

IDE: Integrated Development Environment. Ohjelma tai sovelluskokoelma, jota ohjelmoija käyttää ohjelmistojen kehittämiseen.

Kontti: Container. Säiliöt ovat suoritettavia ohjelmistoja, joiden sovelluskoodi pakataan kirjastojen ja riippuvuuksien kanssa, jotta koodia voidaan ajaa missä tahansa.

Koukku: Hook. Reactissa koukut ovat funktioita, jotka mahdollistavat tilan ja muiden React-ominaisuuksien käytön funktionaalisissa komponenteissa ilman luokkakomponenttien käyttöä.

Palvelin: Backend. Verkkosivuston tai sovelluksen osa, joka hoitaa ohjelmiston tai palvelun toiminnan sekä tiedon tarjoamisen ja käsittelyn.

Selainpuoli: Frontend. Verkkosivuston tai sovelluksen osa, jonka kanssa käyttäjä on tekemisissä. Se sisältää sivuston käyttöliittymän ja toteutetaan eri ohjelmointikielellä.

1 Johdanto

Nykypäivän digitaaliaikana yritykset ja organisaatiot tuottavat valtavia määriä dataa päivittäin. Nämä tiedot voivat olla monessa muodossa, mukaan lukien sähköpostit, raportit, sopimukset ja muut arvokasta tietoa sisältävät asiakirjat. Tietojen valtava määrä voi kuitenkin vaikeuttaa käyttäjien tarvitsemien tietojen tehokasta etsimistä. Monia olemassa olevia hakukoneita ei ole räätälöity yksittäisten yritysten tai organisaatioiden erityistarpeisiin, ja ne tarjoavat usein vähemmän kuin optimaalisen käyttökokemuksen.

Tämän insinöörityön tarkoituksena on suunnitella ja kehittää web-pohjainen käyttöliittymä IBM:n kehittämään Watson Discovery -palveluun. Sovelluksen avulla oli tarkoitus, että yritykset ja organisaatiot pystyisivät tekemään hakuja laajoista dokumentaatiokirjastoista tehokkaammin. Hyödyntämällä kehittyneitä koneoppimisalgoritmeja ja luonnollisen kielen käsittelyä tämä hakukone voi nopeasti tunnistaa ja hakea asiaan kuuluvia asiakirjoja käyttäjän hakuehtojen perusteella. Verkkopohjainen käyttöliittymä suunnitellaan intuitiiviseksi ja käyttäjäystävälliseksi, jolloin käyttäjät voivat helposti suodattaa ja tarkentaa hakutuloksiaan.

Insinöörityö pyrkii ratkaisemaan yritysten tiedonhakuongelmat tarjoamalla käyttäjille innovatiivisen ja tehokkaan työkalun, jonka avulla he löytävät tarvitsemaansa tietoa nopeasti ja helposti. Tehokkaamman tiedonhakupalvelun ansiosta käyttäjät voivat säästää aikaa ja resursseja merkittävästi, mikä samalla parantaa heidän työnsä laatua. [1.]

2 Tavoitteet

Tämän opinnäytetyön tavoitteena oli suunnitella ja rakentaa käyttöliittymä Watson Discovery -palvelulle, jota käyttäjät voivat muokata helposti ja tehokkaasti. Sovelluksen oli tarkoitus hyödyntää IBM Cloudissa toimivaa Watson Discovery -tekstianalyysialustaa rajapintojen kautta. Projektin alussa kävin läpi vaatimukset siitä, mitä sovelluksen tulisi sisältää sekä toiminnallisesti että ei-toiminnallisesti. [2.]

2.1 Toiminnalliset vaatimukset

Toiminnallisuusnäkökulma oli keskeinen osa sovelluksen suunnittelua ja toteutusta. Käyttöliittymän käyttäjäystävällisyys on yksi tärkeimmistä tavoitteista. Sovelluksen on tarjottava käyttäjilleen helppokäyttöinen ja intuitiivinen käyttöliittymä, joka mahdollistaa tehokkaan ja sujuvan vuorovaikutuksen. Käyttöliittymän muokattavuus on myös olennaista, jotta sovelluksen ulkoasu voidaan helposti mukauttaa käyttäjän tarpeisiin (taulukko 1).

Taulukko 1. Projektin toiminnalliset vaatimukset.

Toiminnallisuus	Prioriteetti
Käyttöliittymän ulkoasu pitää olla helposti muokattavissa.	Tärkeä
Sovelluksessa pitää olla filtteri osuus tuloksille.	Tärkeä
Sovelluksessa pitää olla hakutoiminnallisuus.	Tärkeä
Sovelluksessa pitää olla navigaatio toiminnallisuus	Tärkeä

Tulosten suodatus on tarpeellinen suuren tietomäärän hallinnassa ja analysoinnissa. Se mahdollistaa käyttäjille tietojen tehokkaan tarkastelun ja rajauksen.

Lisäksi sovelluksen integrointi Watson Discoveryn rajapintaan avaa mahdollisuuden tiedonhakuun ja analysointiin Watson Discovery -palvelun avulla.

Hakutoiminnallisuus on projektin keskeinen komponentti. Käyttäjien on voitava suorittaa nopeita ja tarkkoja hakuja luonnollista kieltä käyttäen, jotta he voivat löytää tarvitsemansa tiedon helposti. Suodatus käyttöliittymässä tukee tietojen seulontaa ja rajauksia, mikä parantaa käyttäjien mahdollisuuksia tarkastella ja analysoida tietoa yksityiskohtaisesti.

Lisäksi käyttöliittymän ulkoasu on suunniteltava huolellisesti. Carbon Design System -kirjasto tarjoaa standardoidun ja tyylikkään suunnittelujärjestelmän, joka mahdollistaa yhdenmukaisen ja helposti hallittavan käyttöliittymän.

Nämä toiminnalliset vaatimukset muodostavat perustan sovelluksen toiminnalliselle suorituskyvylle ja käyttäjä kokemukselle. Niiden täytäntöönpano on välttämätöntä käyttäjäkokemuksen varmistamiseksi.

2.2 Ei-toiminnalliset vaatimukset

Ei-toiminnalliset vaatimukset viittaavat järjestelmän ominaisuuksiin, jotka eivät liity suoraan sen toiminnallisuuteen, enemmän esimerkiksi mittaristoihin, luotettavuuteen tai saatavuuteen. Lisäksi ei-toiminnalliset vaatimukset voivat sisältää käyttöliittymän selkeyteen tai helppokäyttöisyyteen liittyviä osia. Näiden vaatimusten huomioiminen on tärkeää järjestelmän suunnittelussa ja kehityksessä, jotta lopputulos vastaisi käyttäjien tarpeita ja odotuksia (taulukko 2).

Taulukko 2. Projektin ei-toiminnalliset vaatimukset.

Nimet	Arvo
Liiketoiminta mittaristo	Järjestelmän pitää tukea yhtäaikaisia käyttäjiä.
Saatavuus	Pitää olla saatavilla vähintään työpäivän ajan. Käyttöliittymän pitää olla käyttäjäystävällinen.

Luotettavuus	Sovelluksen suunnittelu vaiheessa on otettu huomioon mikropalvelu arkkitehtuuri. Vaikka yksi komponentti poistettaisiin ei sillä saa olla vaikutusta sovelluksen toimintaan.
Skaalautuvuus	Sovellus skaalautuu horisontaalisesti konttien avulla.
Tietoturva	Käyttäjien tunnistautuminen tapahtuu IBM Cloud autentikaation kautta. Palvelimien välinen keskustelu tapahtuu token pohjaisen autentikoinnin kautta.
Siirrettävyys	Applikaation tulisi olla helposti siirrettävissä.
Rajoitukset & Määräykset	Teknologian pitää tukea IBM Cloudia. Teknologia valintojen pitää olla yhteensopivia Watson Discovery alustan kanssa.
Tekniset Standardit	Sovellus rakennetaan mikropalvelu arkkitehtuurilla. komponentit pitää olla irrotettavissa ilman, että toiminnallisuus kärsii. Käyttöliittymän ulkoasu pitää tehdä käyttäen Carbon Design System kirjastoa.

Ensimmäisessä vaatimuksessa mennään läpi käyttäjämäärän tukemista. Sovelluksen on oltava kykenevä tukemaan vähintään muutamia samanaikaisia käyttäjiä tarpeen mukaan. Tämä varmistaa, että sovellus voi joustavasti vastata muuttuviin tarpeisiin.

Saatavuusvaatimuksissa selvitettiin, kuinka sovelluksen on toimittava vähintään normaalin työpäivän ajan. Tämä takaa, että käyttäjät voivat luottaa siihen, että sovellus on käytettävissä silloin, kun he sitä tarvitsevat, ilman pitkiä käyttökatkoja.

Skaalautuvuusosiossa vaatimukset toteutetaan horisontaalisen konttien (containers) avulla. Tämä mahdollistaa resurssien tehokkaan hallinnan ja joustavan skaalautumisen, mikä varmistaa sovelluksen suorituskyvyn ja tehokkuuden.

Tietoturvaosiossa vaatimuksena oli, että käyttäjien tunnistautuminen tapahtuu IBM Cloudin autentikaatiopalvelun kautta, ja palvelimien välinen viestintä on suojattu token-pohjaisella autentikaatiolla. Tämä varmistaa käyttäjätietojen ja liikenteen turvallisuuden.

Siirrettävyys tarkoittaa, että sovellus suunnitellaan helposti siirrettäväksi eri alustoille ja ympäristöihin ilman suuria muutostöitä. Tämä lisää järjestelmän joustavuutta ja mahdollistaa sen mukauttamisen eri tarpeisiin.

Jatkuvan integraation ja jatkuvan toimituksen (CI/CD) tuki on keskeistä ylläpidon näkökulmasta. CI/CD-pipeline auttaa nopeuttamaan sovelluksen kehitystä ja ylläpitoa mahdollistamalla automatisoidut testaukset, integraation ja toimituksen. [3.]

Sovelluksen on oltava yhteensopiva IBM Cloudin ja Watson Discovery -alustan kanssa ja sen on noudatettava mikropalveluarkkitehtuuria, jossa komponentit voidaan irrottaa ilman toiminnallisuuden heikkenemistä.

Tämä vaatimusmäärittely luo vahvan perustan sovelluksen tulevalle kehitykselle ja toteutukselle. Se auttaa varmistamaan projektin sujuvan etenemisen ja tulosten saavuttamisen. Kaikki nämä vaatimukset yhdessä luovat vankan perustan sovelluksen menestykselle ja sen kyvylle vastata tehokkaasti käyttäjien tarpeisiin.

3 Käytetyt ohjelmistot ja teknologiat

3.1 Ohjelmointiympäristö

Työssä käytettiin Visual Studio Code -kehitysympäristöä (IDE), joka on ilmainen ja avoimen lähdekoodin ohjelmisto. Visual Studio Code (VS Code) on erittäin

suosittu ohjelmointityökalu, joka tarjoaa monipuolisen ja käyttäjäystävällisen työympäristön useille ohjelmointikielille. Valitsin VS Coden sen monipuolisten ominaisuuksien, kuten koodin automaattisen korjauksen ja testauksen ansiosta.

Versionhallintaan käytin Github-sovellusta, joka on suosittu web-pohjainen versionhallintapalvelu. Github mahdollistaa kooditiedostojen tallentamisen ja jakamisen sekä yhteistyön muiden kehittäjien kanssa. Valitsin Githubin sen yleisyyden ja helppokäyttöisyyden vuoksi. Githubin avulla pystyin seuraamaan muutoksia kooditiedostoissa, hallitsemaan haaroja ja yhdistämään muutokset päähaaraan. Tämä helpotti projektin hallintaa huomattavasti ja piti minut ajan tasalla projektin tapahtumista. [4.]

3.2 React.js

Valinta Reactin hyödyntämisestä ohjelmistosovelluksen kehittämisessä perustui moniin harkittuihin tekijöihin. Erityisesti sen erinomaisen suorituskyvyn ja joustavuuden ansiosta se sopii mainiosti vaatimusmäärittelyn (taulukko 1 ja 2) vaatimuksiin.

React on laajalti tunnettu ja hyödynnetty JavaScript-kirjasto, joka on suunniteltu erityisesti dynaamisten käyttöliittymien ja yksisivuisten sovellusten rakentamiseen tehokkaasti ja vaivattomasti. React on tunnettu sen käyttämästä virtuaalisesta DOM-rakenteesta. Se mahdollistaa sivujen tehokkaan päivittämisen ilman uudelleen latauksen tarvetta. Sen avulla React kykenee reagoimaan muutoksiin nopeasti ja optimoimaan käyttöliittymän päivitykset

Reactin laaja kirjasto ja aktiivinen kehittäjäyhteisö takaavat, että siihen on saatavilla runsaasti eri kirjastoja ja komponentteja, mikä helpottaa sovelluksen laajentamista ja räätälöintiä. Nämä ominaisuudet tekevät siitä monipuolisen vaihtoehdon sovelluskehityksen tarpeisiin. [5; 6.]

3.3 Watson Discovery

Watson Discovery edustaa IBM:n huippuluokan älykkään haku- ja tekstianalyysialustan kehitystä, mikä tarjoaa käyttäjilleen ainutlaatuisen mahdollisuuden nopeaan tiedonhakuun ja syvällisten liiketoiminnallisten oivallusten saamiseksi. Tämä alusta on suunniteltu erityisesti rakenteettoman datan käsittelyyn ja analysointiin, mikä tekee siitä erinomaisen valinnan monipuolisissa tiedonhakuprojekteissa. Discovery hyödyntää edistyksellistä koneoppimista ja tekoälyä datan prosessoinnissa, ja tämä ominaisuus tulee erityisen hyvin esiin suurten datamäärien käsittelyssä. Tämä teknologia mahdollistaa tarkan tiedon erottamisen massiivisista tietovirroista ja tarjoaa helposti luettavaa ja ymmärrettävää tietoa käyttäjilleen. Yksi sen merkittävimmistä vahvuuksista on IBM:n rakentama analytiikkamoottori, joka tarjoaa kognitiivisia rikasteita ja syvällisiä havaintoja käsiteltävästä datasta. Moottori ymmärtää dataa ihmismäisesti eri lähteistä. Se mahdollistaa tiedon rikastamisen aihepohjaisesti ja auttaa käyttäjiä saamaan parempaa ymmärrystä isosta data määrästä. [7.]

3.4 Node.js

Node.js on avoimen lähdekoodin alustariippumaton ajoympäristö, joka mahdollistaa JavaScript-koodin suorittamisen palvelimella. Tämä innovatiivinen teknologia on erityisen sovelias reaaliaikaisten sovellusten kehittämiseen, ja tarjoaa monipuolisen ja tehokkaan työkalun ohjelmoijille. Yksi Node.js:n merkittävistä eduista on sen saumaton integroituminen erilaisten käyttöliittymäteknologioiden, kuten React.js:n, kanssa. [8.]

React.js on suosittu JavaScript-kirjasto, jota hyödynnetään modernien käyttöliittymien rakentamisessa. Node.js:n yhteistoiminta React.js:n kanssa avaa monia mahdollisuuksia kokonaisvaltaisten sovellusten kehittämiseen. Tämä integraatio mahdollistaa JavaScriptin käytön sekä asiakas- että palvelinpuolella, ja luo yhtenäisen ja johdonmukaisen ohjelmointiympäristön. Johdonmukainen ja helposti ymmärrettävä ympäristö tehostaa kehitystyötä ja helpottaa sovellusten ylläpitoa. Node.js- ja React.js-yhteiskäyttö avaa ohjelmoijille mahdollisuuden luoda

tehokkaita, reaaliaikaisia ja monipuolisia sovelluksia, jotka täyttävät modernien web-sovellusten vaatimukset.

3.5 Express

Express on Node.js-ympäristöön toteutettu sovelluksen palvelimen puoleinen osa. Express on siis kehys, joka on suunniteltu web-sovelluskehitykseen Node.js-alustalla. Se mahdollistaa tehokkaan web-sovellusten ja RESTful API -palvelimien kehityksen. Express on hyvin suosittu web-kehysvaihtoehto JavaScript-ohjelmointikielen ympärillä.

Express toimii kehyskomponenttina, jossa on sisään rakennettu middleware- sekä reititin. Reitittimen avulla Express käsittelee erilaiset HTTP-pyynnöt ja -vastaukset. Middlewaret ovat toiminnallisuuksia, jotka käsittelevät HTTP-pyyntöjä ja -vastauksia ennen tai jälkeen niiden saapumista reitittimille.

Kehys mahdollistaa HTTP-palvelimen luomisen, joka kuuntelee tulevia HTTP-pyyntöjä ja ohjaa ne Express-sovellukselle. Tämä mahdollistaa sovelluksen vastaamisen pyyntöihin ja toiminnan tarjoamisen selaimen ja palvelimen välillä.

Express on kevyt ja joustava vaihtoehto erilaisille web-sovellusprojekteille. Sitä käyttämällä voidaan nopeasti ja tehokkaasti kehittää erilaisia sovelluksia ja API-palvelimia, sitä voidaan laajentaa lisäosilla sekä räätälöidä omilla väliohjelmilla (middleware) tarpeiden mukaan. [9; 10.]

3.6 Carbon Design System

Vaatimusmäärittelyssä tärkeää oli käyttää IBM:n luomaa Carbon Reactia, joka on avoimen lähdekoodin komponenttikirjasto. Sen ideana on käyttää uudelleenkäytettäviä komponentteja ja tyylejä auttaakseen sovelluskehittäjiä rakentamaan näyttäviä ja johdonmukaisia web-sovelluksia. Käyttöliittymän suunnittelun jälkeen tuli tilanne, jossa eteneminen vaati Carbon React -kehyksen käytön opettelua. [11.]

4 Sovelluksen arkkitehtuuri sekä ulkoasu

4.1 Korkean tason arkkitehtuuri

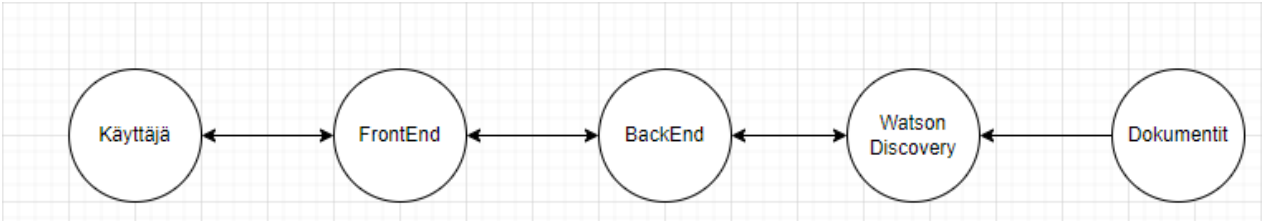
Ylätasolla arkkitehtuuri koostuu neljästä teemasta, jotka ovat käyttöliittymä, käyttäjä, Watson Discovery -palvelu sekä sen käyttämät dokumentit.

Käyttöliittymän edustaa työssä rakennettavaa kokonaisuutta, joka keskustelee suoraan käyttäjien sekä Watson Discovery -palvelun kanssa. Se mahdollistaa hakemisen dokumenteista luonnollista kieltä käyttäen, filtteröinnin sekä oikean dokumentin valitsemisen, jonka avulla käyttäjät löytävät heille relevantin tiedon nopeasti ja tehokkaasti (kuva 1).

Käyttäjän rooli kuvastaa loppukäyttäjiä ja kuinka he käyttävät käyttöliittymää. Tekevät hakukyselyt ja valitsevat heille relevantit dokumentit (kuva 1).

Watson Discovery -palvelu tarjoaa hakuominaisuuden. Se käsittelee haut, etsii siihen sopivat asiakirjat ja luokittelee hakutulokset niiden relevanssin perusteella. Se hyödyntää niin sanottua “Natural Language Processingia” sekä koneoppimisen algoritmeja. Watson Discovery analysoi asiakirjoja parantaakseen hakujen tarkkuutta (Kuva 1).

Dokumentit edustavat Discovery -palveluun tuotettuja dokumentteja, joihin hakuprosessit suoritetaan. Siihen kuuluu laaja valikoima erityyppisiä dokumenttejä kuten esimerkiksi tekstidokumentteja, artikkeleita ja raportteja. Watson discovery indeksoi dokumentit sekä tekee niistä haettavia ja saatavilla olevia käyttäjää varten käyttöliittymän kautta.



Kuva 1. Korkean tason näkymä sovelluksen arkkitehtuurista.

4.2 Käyttöliittymän ulkoasun suunnittelu

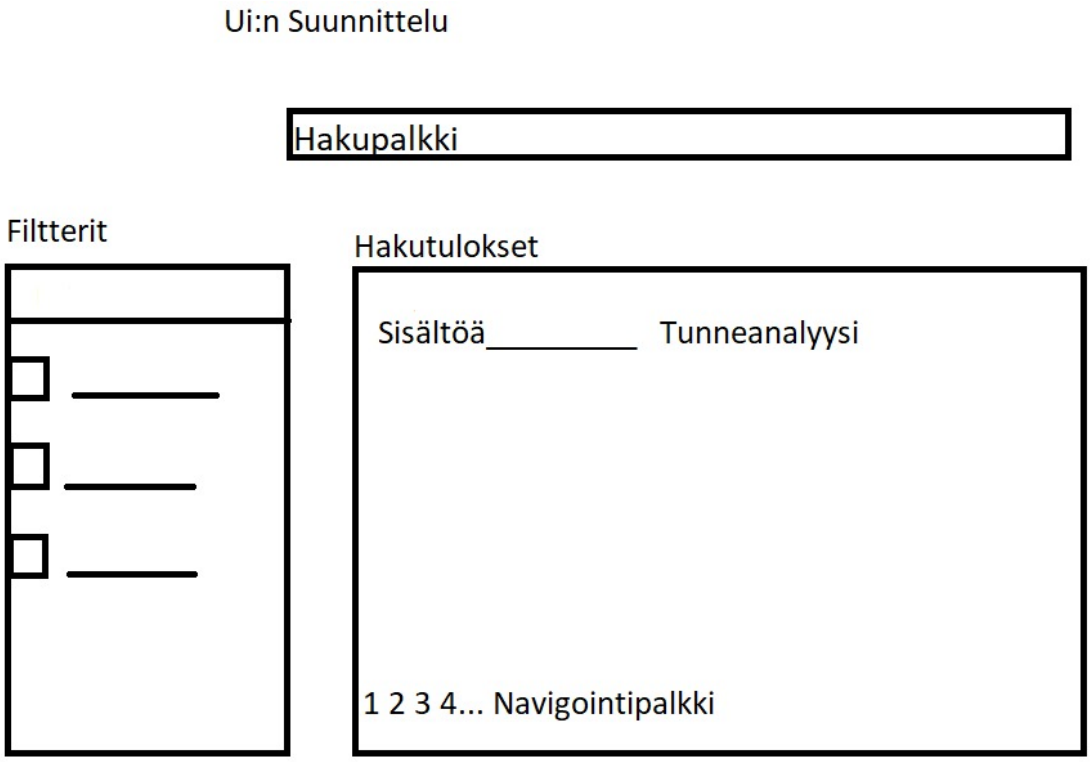
Käyttöliittymän suunnittelu alkoi katsomalla vaatimusmäärittelyä (taulukko 1 ja 2) ja siitä seurasi lopputulos, että siitä on rakennettava mahdollisimman helposti käytettävä ja käyttäjäystävällinen. Tavoitteena alkuun oli saada hyvin minimaalinen versio sovelluksesta, josta kokonaisuutta olisi helppo laajentaa. [12.]

Käyttöliittymän yleiskatsaukseen kuuluu hakupalkki, joka on ensisijainen työkalu Watson Discovery -palvelun kyselyihin. Hakupalkki sijaitsee käyttöliittymän yläosassa ja on helposti näkyvissä käyttäjälle.

Hakupalkin lisäksi on myös suodatinosio, joka käyttää Watson Discovery -palvelun yksiköitä antaakseen tarkemman hallinnan hakutuloksiin. Tämä mahdollistaa käyttäjille kapeampien hakujen tekemisen valitsemalla tiettyjä käsitteitä, kategorioita, konsepteja ja avainsanoja. Jos käyttäjä valitsee suodatinvaihtoehdon, uusi haku suoritetaan, ja hakutulokset päivittyvät automaattisesti.

Tulososiossa näkyy käyttäjän hakutulokset. Kunkin tuloksen esitetään vastaavan otsikon, katkelman ja muiden olennaisten tietojen kanssa. Lisäksi tuloksessa näkyy tunnearviointiasteikko, joka on yksi Watson Discovery -palvelun ominaisuuksista. Sen avulla voidaan nähdä, onko sisältö positiivista vai negatiivista. Tuloksiin olisi myös tarkoitus lisätä painike, josta voitaisiin näyttää alkuperäinen dokumentti, josta hakutulos löytyi.

Lopuksi käyttöliittymän alaosassa löytyy sivuston navigointipalkki, joka mahdollistaa käyttäjille navigoinnin eri sivujen välillä. Tämä mahdollistaa käyttäjille helpon tavan seurata suuria määriä hakutuloksia (kuva 2).



Kuva 2. Käyttöliittymän suunniteltu ulkonäkökuvana.

4.3 Mikropalveluarkkitehtuuri

Kun aloitin rakentamaan hakukonetta, minulla oli monia ajatuksia, jotka liittyivät sovellusarkkitehtuurin valintaan. Yksi tärkeimmistä päätöksistä oli rakentaa sovellus käyttäen mikropalveluarkkitehtuuria.

Mikropalveluarkkitehtuuri on lähestymistapa ohjelmistosovellusten rakentamiseen, joka sisältää suuren monoliittisen sovelluksen jakamisen pienempiin, itsenäisiin palveluihin, jotka voidaan kehittää, käyttöönottaa ja ylläpitää erikseen. Jokainen mikropalveluarkkitehtuurissa oleva palvelu vastaa tietystä toiminnosta ja kommunikoi muiden palveluiden kanssa hyvin määritellyn rajapinnan kautta.

Yksi sen tärkeimmistä eduista on sen joustavuus. Koska jokainen palvelu kehitetään ja ylläpidetään erikseen, muutokset yhdessä palvelussa eivät välttämättä vaikuta muihin palveluihin. Tämä helpottaa sovelluksen muutosten tekemistä aiheuttamatta häiriötä koko järjestelmään.

Toinen iso etu on skaalautuvuus. Koska jokainen palvelu on riippumaton, sitä voidaan skaalata ylös tai alas tarpeen mukaan ilman, että se vaikuttaa muihin palveluihin. Tämä mahdollistaa sovelluksen käsittelyyn vaihtelevat liikenteen ja käytön tasot tehokkaammin.

Lisäksi mikropalveluarkkitehtuuri edistää modulaarista ja ylläpidettävää koodia. Jakamalla sovellus pienempiin, itsenäisiin palveluihin koodi tulee helpommin ymmärrettäväksi ja ylläpidettäväksi ajan myötä. Se myös mahdollistaa tiimeille työskentelyn tietyissä palveluissa häiritsemättä muita osia sovelluksesta.

Mikropalveluarkkitehtuurin toteuttaminen ei kuitenkaan tule ilman haasteita. Yksi suurimmista haasteista on sovelluksen monimutkaisuuden hallinta. Monien palveluiden kommunikoidessa toistensa kanssa rajapintojen kautta tietojen hallinta voi tulla vaikeaksi ja varmistaa, että kaikki palvelut toimivat tehokkaasti yhdessä. [13.]

5 Sovelluksen toteutus

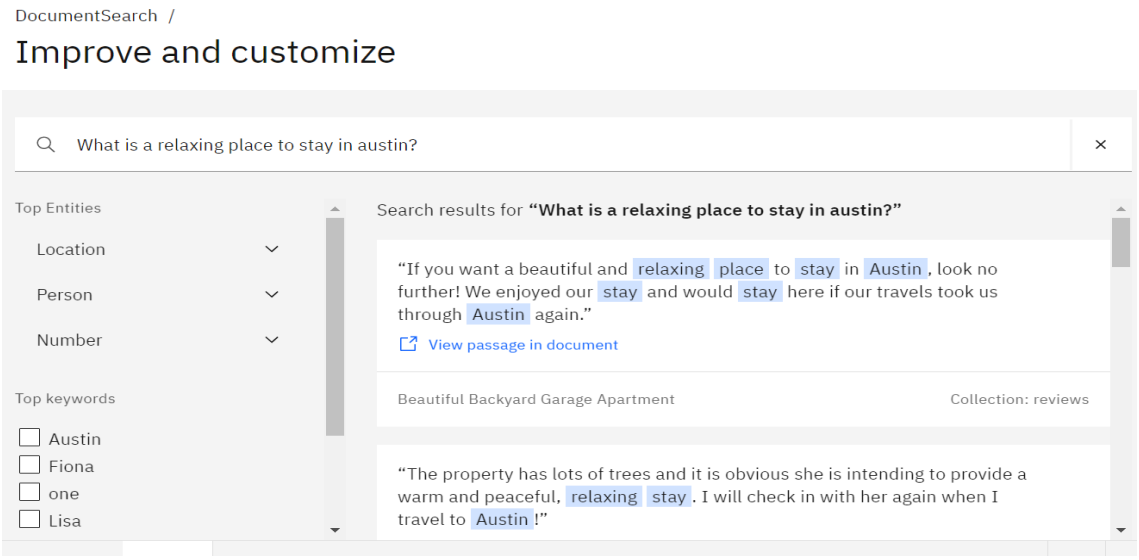
5.1 Watson Discoveryn asennus

Asennus alkoi varaamalla instanssin Watson Discovery -palvelusta IBM Cloudista. Tämä prosessi oli suoraviivainen ja kesti vain minuutin. Kun palvelu oli käynnissä, tehtiin uusi kokoelma, johon tallennettiin hakukoneen tarvitsema data.

Työssä oli käytössä aineisto, joka sisälsi noin tuhat arvostelua Airbnb-vierailta, jotka olivat yöpyneet Austinissa, Texasissa. Valitsin tämän aineiston, koska se

oli tarpeeksi suuri osoittamaan Watson Discoveryn hyödyt, mutta myös riittävän tarkka ollakseen relevantti mahdollisille käyttäjille.

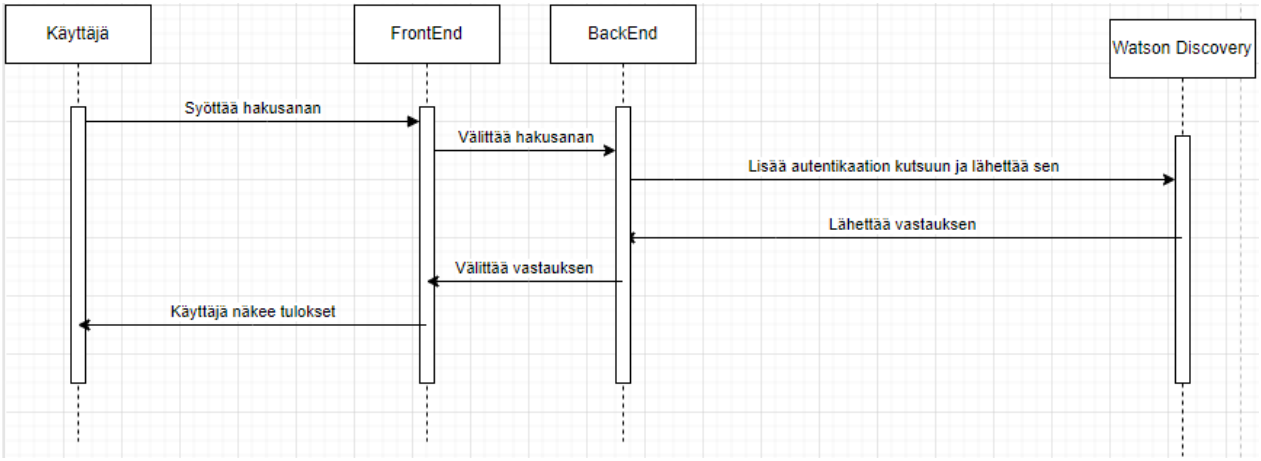
Olin vaikuttunut siitä, kuinka Watson Discoveryn rikastusominaisuudet mahdollistavat entiteettien, käsitteiden ja kategorioiden erottamisen ja luokittelun arvosteluista. Tämä mahdollisti tarkemman hallinnan hakutuloksista ja paransi hakukoneen tarkkuutta (kuva 2). [14.]



Kuva 3. Projektiin tehty Watson Discovery -instanssi IBM Cloudista ja demonstraatio sen hakutoiminnosta ja filttereistä.

5.2 Frontend

Frontend-arkkitehtuuri perustuu modulaariseen lähestymistapaan, jossa käytössä oli React-kirjasto käyttöliittymän kehittämiseen. Käyttöliittymä koostuu useista komponenteista, jotka kommunikoivat keskenään ja muodostavat monipuolisia toiminnallisuuksia. Sekvenssikaavio, jonka avulla kuvaamme komponenttien vuorovaikutusta, auttaa hahmottamaan, miten eri osat kommunikoivat keskenään ja millä tavalla tiedonkulku tapahtuu. Tämä auttaa meitä hahmottamaan sovelluksen kokonaisuutta ja varmistamaan sen sujuvan toiminnan (kuva 4).



Kuva 4. Sekvenssikaavio sovelluksen arkkitehtuurista, missä demonstroidaan, mitä sovelluksen puolella tapahtuu käyttäjän käyttäessä käyttöliittymää.

Käyttöliittymän rakentaminen mahdollistaa lukuisia etuja, kuten moni ihminen voi käyttää sovellusta samaan aikaan. Tämä antaa mahdollisuuden räätälöidä ja integroida eri palveluita, mikä helpottaa sovelluksen hallintaa ja ylläpitoa.

5.2.1 Hakutoiminnon rakentaminen

Käyttöliittymän kehitys alkoi React-projektin luomisella ja varmistamalla, että kaikki tarvittavat kirjastot ja riippuvuudet olivat mukana. React-projekti luotiin käyttäen Create React App -työkalua, jonka jälkeen asennettiin loput tarvittavat kirjastot.

Sovelluksen hakutoiminto vastaa käyttäjien syötteiden käsittelystä. Komponentti koostuu hakupalkista, johon käyttäjät syöttävät hakukyselyn (kuva 3). Käyttäjän syöttäessä hakusanan komponentti käyttää ‘useState’-koukkua hakukyselyn tilan hallintaan. Hakukysely tallennetaan ‘search’-tilaan aina, kun käyttäjä kirjoittaa hakupalkkiin. ‘OnChange’-tapahtuma päivittää ‘search’-tilan uudella arvolla (esimerkkikoodi 1).

```
const [search, setSearch] = useState("");
```

Esimerkkikoodi 1. Tässä luodaan muuttuja ‘search’ ja sille päivitysfunktio ‘setSearch’ käyttäen ‘useState’-koukkua.

Hakutoiminto sisältää viiveen, joka on rakennettu käyttäen ‘useEffect’-koukkua. Se asettaa 500 ms:n odotusajan ennen kuin hakukysely päivittyy uudelleen. Tämä estää tarpeettomia hakukyselyitä ja parantaa sovelluksen suorituskykyä sekä käyttöliittymän käyttäjäystävällisyyttä (esimerkkikoodi 2).

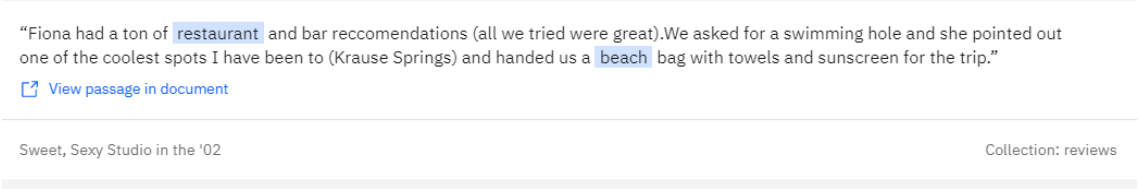
```
useEffect(() => {
 const timeOutId = setTimeout(() => setSearchText(search), 500)
; return () => clearTimeout(timeOutId);
}, [search, setSearchText]);
```

Esimerkkikoodi 2: Varmistaa, että ‘setSearchText’-funktiota kutsutaan 500 ms viiveen jälkeen, kun käyttäjä lopettaa kirjoittamisen.

5.2.2 Vastauksen näyttäminen käyttöliittymällä

Komponentin ideana oli, että rakennetaan erikseen yksittäisen ‘Card.js’-javascript-tiedoston, joka ottaa yksittäisen dokumentin tiedot. Sitten komponentista löytyy vielä ‘Cardholder.js’, jonka tehtävänä on renderöidä laatikko. Sen sisällä on Watson Discovery -palvelusta tulleet vastaukset edellä mainittujen korttien muodossa erottaen ne toisistaan.

Käytännössä ajatellen “Cardholder.js” on laatikon runko ja sen sisältö koostuu korteista, joista löytyy dokumenttien sisältö. Komponentti sisältää myös painikkeen, jonka avulla vastauksen sisältävä dokumentti voitaisiin avata siitä kohdasta mikä vastaa hakijan kysymykseen (kuva 5).



Kuva 5. Kuvassa näkyy kuinka vastaukset näkyvät, kun ne on otettu Watson Discovery -palvelusta, joka pyörii IBM Cloudissa.

5.2.3 Sivun navigointi

Vaatimusmäärittelyyn viitaten (taulukko 1) käyttäjäystävällisyys oli tärkeä toiminnallisuus, joten lähdin rakentamaan käyttöliittymään navigointiosuutta. Komponentin tehtävänä on käsitellä sivunumeroita ja näytettävien kohteiden määrän muutoksia (esimerkkikoodi 3).

```
const handlePageChange = (page, pageSizes) => {
setPageNumber(page);
setItemsPerPage(pageSize);
console.log(page);
console.log(pageSize);
};
```

Esimerkkikoodi 3: Tämä funktio kutsuu ‘setPageNumber’-funktiota uudella sivunumerolla ja ‘setItemsPerPage’-funktiota uudella näytettävien kohteiden määrällä.

Komponentti palauttaa navigointipalkin (kuva 6), jonka avulla voi selkeästi navigoida hakutulosten ylittäessä tietyn määrän seuraavalle sivulle. Yhteenvetona koodi määrittelee ‘Pagination’-komponentin, jossa on erilaisia vaihtoehtoja sivun vaihtamisen, ulkoasun ja toiminnan hallitsemiseksi. ‘handlePageChange’-funktiota kutsutaan, kun käyttäjät vuorovaikuttavat komponentin kanssa tilan muutosten käsittelyä varten (esimerkkikoodi 4).

```
return (
 <Pagination
 backwardText='Previous Page'
 forwardText='Next Page'
 itemsPerPageText='Items per page'
 page={1}
 pageNumberText="Page Number"
 pageSize={10}
 pageSizes={[10, 20, 50]}
 totalItems={documentCount}
 onChange={handlePageChange}
 />)
```

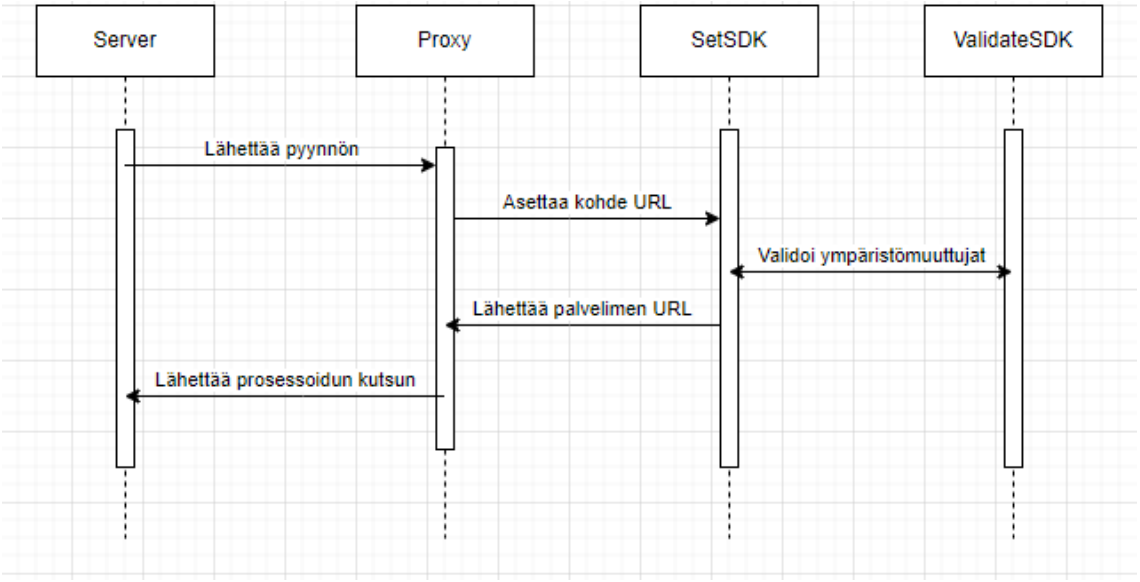
Esimerkkikoodi 4. Carbon Design Systemillä rakennetun Pagination-komponentin koodi, joka mahdollistaa sivujen selailun ja näytettävien kohteiden määrän asettamisen.

5.3 Backend

Tavoitteena oli rakentaa sovelluslogiikka sekä välityspalvelin, jonka kautta sovelluksesta lähtevät kutsut menisivät suoraan Watson Discovery -palvelun rajapintoihin. Samalla hoidetaan Watson Discoveryn ja sovelluksen välillä oleva token-autentikaatio niin kuin vaatimusmäärittelyssä oli päätetty (taulukko 2).

Palvelin tehtiin toimimaan käyttöliittymän ja Watson Discoveryn välillä, mikä mahdollistaa pyyntöjen käsittelyn ja valvonnan sekä tarvittaessa liikenteen ohjaamisen. Tällä eristetään käyttöliittymä ja Watson Discovery toisistaan, mikä tekee järjestelmästä joustavamman ja helpommin hallittavan. Lisäksi palvelimen avulla voidaan toteuttaa erilaisia autentikointi- ja turvamekanismeja, mikä parantaa järjestelmän tietoturvaa.

Watson Discovery -dokumentaatiosta löytyi discovery-components-GitHub-sivusto, jonka tarkoituksena on antaa Watson Discoveryn palvelun kanssa työskenteleville hyvät esimerkit siitä, miten backend logiikkaa kannattaa rakentaa validoinnin sekä token-pohjaisen autentikaation suhteen. Lähde [15.]



Kuva 5. Sekvenssikaavio Backend-sovelluslogiikan toiminnasta.

5.3.1 Palvelimen rakentaminen

Työ alkoi Server.js-tiedostosta (esimerkkikoodi 5), jonka on tarkoitus toimia sovelluksen palvelimena. Se käyttää Express.js-frameworkia, joka luo HTTP-palvelimen. Kun käyttäjä käy sovelluksen pääsivulla, palvelin toimittaa käyttöliittymä-HTML-tiedoston. Lisäksi koodi käyttää 'setupProxy.js'-tiedostoa määrittämään välityspalvelimen (proxy) reitittämään pyyntöjä.

```
const path = require('path');
const express = require('express');
const app = express();
const proxy = require('./src/setupProxy.js');
app.use(express.static(path.join(__dirname, 'build')));
app.get('/', async (req, res) => {
  res.sendFile(path.join(__dirname, 'build', 'index.html'));
});
proxy(app);
const port = 4000;
app.listen(port, () => {
  console.log('Discovery application is running at http://lo-
calhost:%s/', port);
});
```

Esimerkkikoodi 5. Koodissa luodaan express-sovellus sekä otetaan käyttöön välityspalvelin ja asennetaan se aiemmin luotuun express-sovellukseen. Sitten määritetään kuunneltava portti ja käynnistetään palvelin.

5.3.2 Välityspalvelin sekä autentikaatio

Seuraavaksi 'setupProxy.js'-tiedoston on tarkoitus toimia sovelluksen välityspalvelimena (proxy). Se käyttää 'http-proxy-middleware'-kirjastoa reitittämään pyyntöjä palvelimelle. Lisäksi se käyttää 'ibm-watson/auth'-kirjastoa Watson Discovery -palveluun kirjautumista varten.

Tiedosto myös varmistaa, että ympäristömuuttujat on määritetty, sekä käyttää 'setSDKurl'-tiedostoa asettaakseen kohdepalvelimen URL-osoitteen (esimerkkikoodi 6). Tämä välikäsi auttaa reitittämään kutsut Watson Discovery -alustalle sekä sen käytön tarvittaviin resursseihin. 'setSDKurl' myös prosessoi ja lukee .env-nimisen tiedoston, johon on tallennettu sensitiivisiä ympäristömuuttujia, API-avain sekä Watson Discovery -palvelun URL- ja token-pohjaisen autentikaation tarvittavat osat.

```
const addAuthorization = async (req, _res, next) => {
    const authenticator = getAuthenticatorFromEnvironment('discov-
ery');
    try {
      const accessToken = await authenticator.tokenManager.getToken();
      req.headers.authorization = `Bearer ${accessToken}`;
    } catch (e) {
      console.error(e);
    }
    return next();
  };
```

Esimerkkikoodi 6. Koodissa määritetään middleware-toiminto, joka lisää validoinnin pyyntöön.

5.4 Keskeisimmät ongelmat kehityksen aikana

Haasteita työn aikana tuli useampia, uusien teknologioiden omaksuminen pitkälti dokumentaatioiden sekä esimerkkien toteutusten kautta. Isoimmat ongelmat tulivat projektin alussa, kun aluksi oli tarkoitus käyttää 'discovery-components' -valmista kirjastoa ja laajentaa siihen vaatimusmäärittelyn mukaiset toiminnot. Tässä lähestymistavassa tuli eteen nopeasti iso määrään virhetilanteita ja sen seurauksena projekti tehtiin ilman discovery-components kirjastoa koodaten itse sovelluksen eri komponentit alusta loppuun. [15.]

Projektin loppupuolella uudet ongelmat syntyivät Node.js-riippuvuuksien kanssa. Suuri osa käytetyistä kirjastoista vaativat päivityksiä, joka johti suureen määrään virheitä. Tämä hidasti huomattavasti projektin etenemistä ja vaatii jatkokehitystä niiden korjaamiseksi.

6 Lopputulos ja yhteenveto

6.1 Lopputulos

Projektin loppupuolella saavutettiin vaihe, jossa useat komponentit ovat täysin valmiita, kuten sivun hakulogiikka, navigointiosuus sekä Backend.

Sivun navigointilogiikka toimii hakutulosten järjestämisessä ja esittämisessä käyttäjille. Kun kohteena on tyypillisesti isommat tietoaineistot se helpottaa hakutulosten pilkkomisessa pienempiin osiin tai navigoitaviin sivuihin. Toteutus korostaa käyttäjäystävällistä käyttöliittymän suunnittelua, joka oli vaatimusmäärittelyssä yksi tärkeä osa.

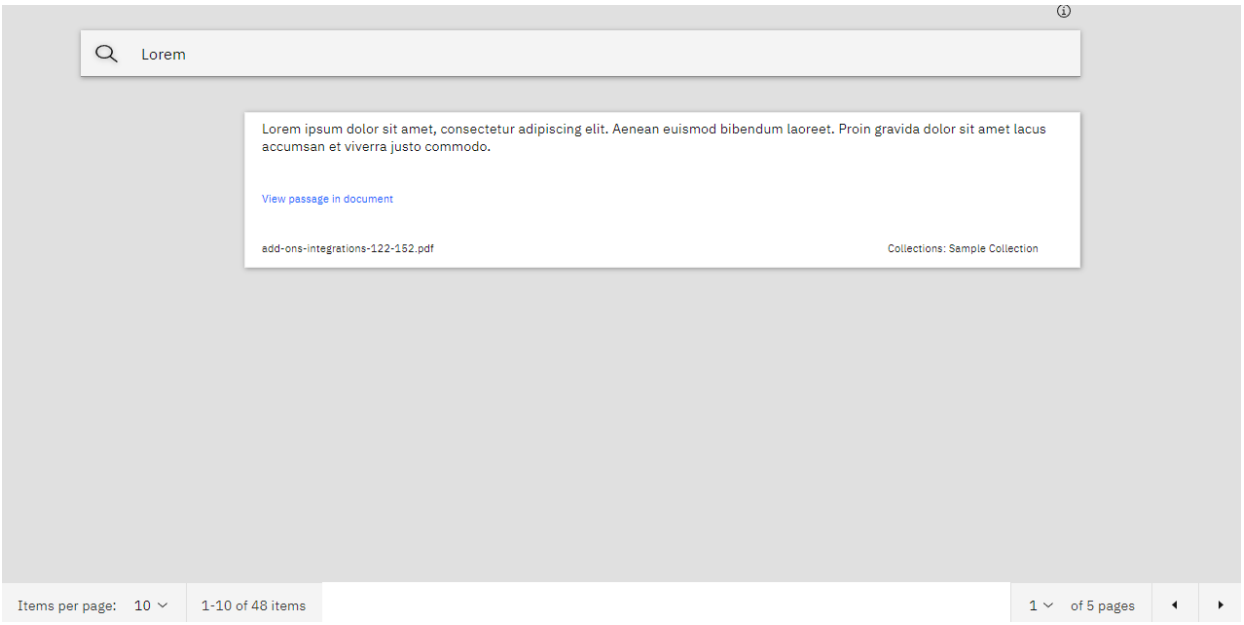
Kuva 6. Työhön rakennettu navigointikomponentti.

Hakutoiminto on tärkein komponentti työssä sekä mahdollistaa suoran vuorovaikutuksen käyttöliittymän kanssa. Sen tehtävänä on ottaa käyttäjän syöttämä teksti ja ohjata sitä eteenpäin sovelluksessa. Komponenttiin on myös rakennettu käyttäjäystävällistä logiikkaa, kuten esimerkiksi viiveen asettaminen jokaisen haun jälkeen estäen tarpeettomat vahinkopainallukset sekä oikealla puolelle painike, jolla voi helposti tyhjentää kentän.

Kuva 7. Hakukomponentti.

Palvelinlogiikka valmistui täysin työn lopussa, ja se toimii perustana tiedon käsittelylle sekä vuorovaikutuksen Watson Discovery -palvelun kanssa sovelluksessa. Sen avulla sovellus pystyi yhdistämään pilvessä pyörivään Watson alustaan käyttäen token-pohjaista autentikaatiota.

Yhteenvetona työssä saatiin monta komponettia valmiiksi sekä Backend-puoli. Vastauksien esittämisestä vastuussa oleva Card.js-komponentti on pitkällä kehityksessä, mutta sovelluslogiikka vaatii vielä kehitystyötä (kuva 6).



Kuva 6. Käyttöliittymä, jossa on Card.js-komponentti, alanavigaatiokomponentti sekä hakukomponentti.

Työstä jäi huomattava määrä suunnittelumateriaalia sekä lähdekoodia, minkä avulla mahdollinen tulevaisuuden jatkokehitys helpottuu huomattavasti.

6.2 Pohdinta

Projektin aikana oppimista tapahtui huomattavan paljon, mutta ehkä suurimmat oppitunnit olivat ajankäytön ja järjestelmällisyyden tärkeys. Projektia lähdettiin rakentamaan pelkästään opiskelusta saadun kokemuksen varassa liian kunnianhimoisesti. Myöhemmin tuli selväksi, kuinka paljon järkevämpää olisi ollut suorittaa projekti harkitummin. Projekti oli liian laaja yhdelle ihmiselle, varsinkin kun kyseessä oli aloitteleva sovelluskehittäjä.

Yksi tärkeä oppi oli parempien versionhallintatyökalujen käyttö, sillä valitettavan usein projektia päivitettiin GitHubin versionhallintaan vain harvoin. Tämä johti versioiden sekoittumiseen ja React-kirjastojen ongelmiin. Huolellisella versionhallinnalla projektia voisi pitää ajan tasalla muutoksista ja palata tarvittaessa vanhoihin versioihin.

Projekti olisi hyötynyt suuresti ketterämmästä lähestymistavasta sovelluskehitykseen. Jos olisin keskittynyt yhteen komponenttiin kerrallaan, projekti olisi edennyt pidemmälle. Projektissa selvän viikoittaisen aikataulun tekeminen ja suunnitteleminen olisi auttanut huomattavasti.

Lopputuloksena projekti on vielä kaukana valmiista toteutuksesta. Sekä frontend että backend vaativat edelleen huomattavaa työtä ja hienosäätöä. Koko projekti on kuitenkin toiminut tärkeänä oppimisprosessina, ja oppimista tuli paljon siitä, kuinka suunnitella ja toteuttaa ohjelmistoprojektia tehokkaasti ja järjestelmällisesti. Tämä kokemus tulee varmasti hyödyttämään tulevia projektejani ja edistymistäni IT-alalla.

6.3 Yhteenveto

Tämän työn tavoitteena oli suunnitella ja rakentaa käyttöliittymä Watson Discovery -palvelulle, jota käyttäjät voivat muokata helposti ja tehokkaasti. Sovelluksen oli tarkoitus hyödyntää IBM Cloudissa toimivaa Watson Discovery -tekstianalyysialustaa rajapintojen kautta. Käyttöliittymän tarkoituksena oli pystyä demonstroimaan Watson Discovery -tekstianalyysialustaa suorittamalla tarkkoja hakuja isoon määrään dataa.

Projekti lähti käyntiin haluttujen toiminnallisuuksien kartoittamisella ja ideoimalla, kuinka projektin toiminnallisuudet toteutuisivat käyttöliittymässä. Suunnittelupalavereissa määriteltiin käytettävät teknologiat sekä halutut toiminnallisuudet sekä käytännöt.

Projektiin sisältyi paljon sovelluskehitykseen kehittäviä työtehtäviä, ja projekti johti uusien taitojen oppimiseen lähinnä Watson Discovery -palvelusta sekä React-kirjaston käytöstä. Myös parannuskohteita löytyi projektin aikana muun muassa suunnittelun tärkeydestä, aikatauluttamisesta sekä sovelluksen versionhallinnan ylläpitämisestä.

Tuloksena oli Watson Discovery -palvelua hyödyntävä käyttöliittymän runko, jossa käyttöliittymän komponentit on viety pitkälle ja siinä on paljon käyttökelpoista koodia. Frontend- ja backend-komponenttien välinen integraatio tarvitsee hienosäätöä, mutta palvelinpuoli on muuten paljon valmiimpi kokonaisuus.

Lähteet

1 Watson Discovery. Verkkoaineisto. https://www.ibm.com/products/watson-discovery. Luettu 1.12.2022.

2 Tutorialspoint. Verkkoaineisto. https://www.tutorialspoint.com/software_engineering/software_requirements.htm. Luettu 1.12.2022.

3 CI/CD. Blogi. https://www.ibm.com/blog/ci-cd-pipeline/. Luettu 2.12.2022.

4 Visual Studio Code. Verkkoaineisto / dokumentaatio. https://code.visualstudio.com/. Luettu 2.12.2022.

5 React. Verkkoaineisto. https://en.wikipedia.org/wiki/React_(software). Luettu 15.12.2022.

6 React. Dokumentaatio. https://react.dev/learn/managing-state. Luettu 20.12.2023.

7 Watson Discovery. Dokumentaatio. https://cloud.ibm.com/apidocs/discovery-data. Luettu 1.2.2023.

8 Node. Verkkoaineisto. https://nodejs.org/en/about. Luettu 2.2.2023.

9 Express. Verkkoaineisto. https://www.geeksforgeeks.org/express-js/. Luettu 2.4.2023.

10 Express. Dokumentaatio. https://expressjs.com/en/5x/api.html#app. Luettu 1.2.2023.

11 Carbon Design React. Verkkoaineisto. https://carbondesignsystem.com/. Luettu 4.1.2023.

12 Importance of user interface. Verkkoaineisto. https://www.linkedin.com/pulse/importance-user-friendly-design-today-mountbirch-1f/. Luettu 6.1.2023.

13 Mikropalvelu arkkitehtuuri blogi. Verkkoaineisto. https://www.simform.com/blog/what-are-microservices/. Luettu 2.2.2023.

14 IBM. Verkkoaineisto. https://cloud.ibm.com/docs/discovery-data?topic=discovery-data-getting-started. Luettu 2.2.2023.

15 Watson discovery kirjasto. Repositorio. https://github.com/watson-developer-cloud/discovery-components/tree/master. Luettu 2.2.2023.