SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

PIKAHAKU KÄYTTÄEN RE-DIKSEN HAKUINDEKSIÄ

TEKIJÄ/T: Severi Kupari

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Severi Kupari	
Työn nimi Pikahaku käyttäen Rediksen hakuindeksiä	
Päiväys 21.5.2020	Sivumäärä/Liitteet 28/0
Ohjaaja(t) Jussi Koistinen, Sami Lahti	
Toimeksiantaja/Yhteistyökumppani(t) Ropo Capital	

Tiivistelmä

Opinnäytetyön tavoite oli suunnitella sekä toteuttaa pikahaku käyttäen Redikseen luotua hakuindeksiä. Opinnäytetyön tilaajana toimi Ropo Capital Oy. Opinnäytetyön tuotos oli osa yrityksen käyttöliittymäuudistus-projektia. Pikahaku sijaitsee käyttöliittymän kojelaudalla, mistä asiakas käyttää hakua. Pikahaku etsii asiakkaan sisällöstä tehtäviä, asiakkaita ja tuotteita.

Lopputuotteena valmistui pikahaun käyttöliittymäkomponentti, joka luotiin käyttämällä TypeScriptiä ja JavaScriptin luokkakirjastoa Reactia. Redikseen luotu hakuindeksi toteutettiin PHP-ohjelmointikielellä. Käyttöliittymän ja hakuindeksin välisessä kommunikoinnissa käytettiin GraphQL-kyselykieltä.

Opinnäytetyön tuotoksena luotiin käyttöliittymä pikahakutoiminnolle ja mahdollistettiin täten helppo tapa hakea tietoja NoSQL-tietokannasta Rediksestä, vaivattomasti automaattisen täydennyksen avulla. Uudistus oli asiakkaille tärkeä nopean hakutoiminnallisuuden vuoksi.

Avainsanat
NoSql-tietokanta, Redis, ReactJs, automaattinen täydennys, GraphQL, TypeScript, PHP

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Severi Kupari			
Title of Thesis Instant Search by Using Search Index in Redis			
Date	21 May 2020	Pages/Appendices	28/0
Supervisor(s) Jussi Koistinen, Sami Lahti			
Client Organisation /Partners Ropo Capital			

Abstract

The aim of this thesis was to plan and execute instant search by using the created search index in Redis. The thesis was commissioned by Ropo Capital Oy. The output of the thesis was part of the company's user interface reform project. Instant search is located on the dashboard of the user interface, where the customer uses the search. In customer content, this searches for tasks, customers, and products.

As an end product, an instant search UI component was completed and created using TypeScript and React, which is a JavaScript class library. The search index created in Redis was implemented by the PHP programming language. The GraphQL query language was used to communicate between the user interface and the search index.

The result of this thesis was an interface to the instant search function, thus enabling an easy way to retrieve data from the NoSQL database from Redis, effortlessly with auto-completion. The reform was important for customers due to its fast search functionality.

Keywords
NoSql-Database, Redis, ReactJs, autocomplete, GraphQL, TypeScript, PHP

ESIPUHE

Haluan kiittää Ropo Capital Oy:tä mielenkiintoisesta ja antoisasta projektista sekä erityisesti Ossi Pesosta sekä Esa Anttilaa opinnäytetyön aikana saamastani tuesta ja kannustuksesta. Erityisesti Ossin ammattitaito ja panostus projektin alkuunsaattamisessa oli korvaamatonta. Esitän myös kiitokset Savonia-ammattikorkeakoulun lehtori Jussi Koistiselle sekä lehtori Sami Lahdelle opinnäytetyön laadukkaasta ohjauksesta.

Kuopiossa 21.5.2020
Severi Kupari

LYHENTEET JA MÄÄRITELMÄT

NoSql-tietokanta perinteisestä relaatiomallista poikkeava tietokanta.

Redis muistin sisäinen tietorakennevarasto.

ZSET lajitellun sarjan tietotyypin nimi Rediksessä.

Mutaatio GraphQL kysely, millä voidaan lisätä, päivittää tai poistaa tietoja tietokannassa.

Käänteinen indeksi tietorakenne, mikä sisältää linkityksen tietojen sijainnista asiakirjoissa.

Lajiteltu sarja sarja, jonka jäsenet järjestetään luonnolliseen järjestykseen jäsenien lisäyksen yhteydessä.

Hakuindeksi tietokantahakemisto, joka koostuu indeksoitavista tiedoista.

Etuliite hash-puut hajautettuja tietorakenteita, jotka mahdollistavat monimutkaisten kyselyjen tekemisen hajautettuun hash-tauluun.

Tokenisointi prosessi, missä tiedot normalisoidaan yksittäisiksi tyypeiksi.

RDB-tiedosto binäärinen esitysmuoto tallennetuista tiedoista.

SISÄLTÖ

1 JOHDANTO

Hakukoneiden toiminta perustuu indeksin luomiseen. Hakuja optimoidaan kokoamalla indeksiin valmiiksi vastauksia pyyntöjen käsittelemisen nopeuttamiseksi, sillä olennaisten tietojen löytäminen erillisiltä sivuilta on hidasta ja raskasta. Käänteisen indeksin luominen on tunnettu tapa, jota käyttämällä esimerkiksi Googlen hakukoneet löytävät hakua vastaavia tuloksia sivustoilta. (Marsden, 2018)

Opinnäytetyö on kehittämistyö, jonka tilaajana toimi Ropo Capital Oy. Ropo Capital on Suomen johtava laskutusyritys ja asiakasyrityksiä Ropo Capitalilla on yli kahdeksan tuhatta (Ropo Capital). Ruotsin markkinoille Ropo Capital laajensi yritysostojen kautta ostettuaan joulukuussa 2019 ruotsalaisen yrityksen Colligent Inkasso AB:n. Käyttöliittymäuudistuksen myötä Ropo Capital tulee tarjoamaan palveluitaan kansainvällisille markkinoille uudistetulla ilmeellä. Käyttöliittymäuudistuksen yhtenä osana luotiin käyttöliittymä pikahakutoiminnolle ja mahdollistettiin täten helppo tapa hakea tietoja NoSQL-tietokannasta Rediksestä vaivattomasti automaattisen täydennyksen avulla. Pikahakuun liittyvät haasteet ja mahdollisuus tuottaa asiakkaille tärkeä ja nopea hakutoiminnallisuus hakuindeksiä käyttäen vaikutti mielenkiintoiselta. Työn tarkoituksena oli rakentaa pikahaku, jossa on automaattisen täydennyksen toiminnallisuus. Käyttöliittymällä on kojelauta, josta asiakas käyttää hakua, joka etsii asiakkaan sisällöstä tehtäviä, asiakkaita ja tuotteita. Työn tavoitteena oli laittaa kaikki edellä mainittu sisältö yhtenäiseen muotoon Redikseen, jossa haku suoritetaan hakuindeksistä. Nopea tiedonhaku on tärkeä osa käytettävyyttä. Työ auttoi työyhteisössä käyttöliittymäprojektin edistymistä ja samalla opin uusimpia käyttöliittymä teknologioita.

2 YHTEISTYÖKUMPPANIN ESITTELY

Yhteistyökumppanina projektissa toimi Ropo Capital Oy, joka on teknologiatalo, missä keskitytään yhdessä asiakkaiden kanssa laskutuksen palveluiden kehittämiseen yrityksen omaa teknologiaa käyttäen. Ropo Capital Oy tarjoaa palveluitaan jo yli 8 tuhannelle asiakkaalleen laadukkaasti. Yksi monista yrityksen tavoitteista on tehdä laskusta iloinen asia ja mahdollistaa yrityksien saavan palveluistaan tai tavaroistaan sovittu ostohinta.

Yritys työllistää jo noin 180 työntekijää ja hallitsee asiakasvaroja vuoden aikana 5 miljardin edestä. Ropon teknologia on 100% suomalaista, yrityksen tarkoituksena on muuttaa koko toimialaa ja luoda laskutukselle uusi aikakausi. Tähän Ropo tähtää tarjoamalla ratkaisuja rautaisella teknologiaosaamisella ja vahvan automaation lisäksi prosessien tehokkaan digitalisoinnin avulla. Yrityksen tehtävänä on mahdollistaa vaivaton laskutus yrityksille ja huolehtia laskutuksesta.

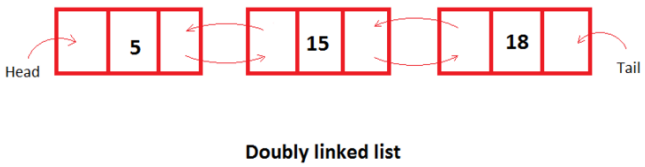
Laatupolitiikan peruselementit avautuvat yrityksen arvojen kautta, joita ovat työn ilo, rohkeus, uuden oppiminen, suorapuheisuus ja tuloksellisuus. Asiakkaille tarjottavan laadun perustana toimii Ropolaisten sitoutuminen ja innostus työhönsä. Työyhteisön työtyytyväisyyttä mitataan aktiivisesti työpaikan kehittämiseksi. (Ropo Capital)

3 REDIS NOSQL-TIETOKANTA

Redis eli remote dictionary server tukee 11:tä eri koodauskieltä, joita ovat Python, Java, PHP, Perl, Go, Ruby, C/C#/C++, JavaScript ja Node.js (Amazon 2012). Redis on avoimen lähdekoodin muistin sisäinen tietorakennevarasto, jota hyödynnetään tietovarastona, välimuistina ja sanomanvälittäjänä. Se on vahvistettu luontaisella replikaatiolla, Lua-komentosarjoilla, LRU-häädöllä (LRU eviction), eristetyillä sanomilla sekä tasoilla, jotka päättävät väliaikasesti levylle tallennettavan tietosisällön säilytysajan. Redis on laadittu ANCI C:llä ja se toimii useimmissa POSIX- järjestelmissä, kuten Linux, *BSD ja OSX ilman ulkoisia yhteyksiä. Rediksen kehitys ja testaus suoritetaan suurimmaksi osaksi Linux- ja OSX-käyttöjärjestelmillä. Redis ei tue Windows-käyttöjärjestelmää. (Introduction to Redis)

3.1 Rediksen tietorakenteet ja tyypit

Redis mahdollistaa viiden erityyppisen avaimen perusteella tiedon yhdistämisen erilaisiin tietotyyppehin. Näitä tietotyyppejä ovat merkkijonot, listat, sarjat, hajautustaulut (hash table) sekä lajitellut sarjat. Jokaisella viidestä rakenteesta on joitain jaettuja komentoja: poistaminen, avaimen tietotyypin palauttaminen sekä avaimen uudelleen nimeäminen. Niiden toteutus ja semantiikka ovat vastaavia kuin samoilla tietorakenteilla, jotka ovat rakennettu muilla ohjelmointikielillä. Näistä tietotyypeistä ainutlaatuisin Redikselle on lajiteltu sarja, jota kutsutaan ZSET:iksi. (Redis Labs, What Redis data structures look like) Merkkijonot ovat Rediksen kaikista tietotyypeistä yksinkertaisin ja yleisin. Rediksen merkkijonot ovat binääriturvallisia mikä tarkoittaa, että merkkijono voi sisältää kaikenlaista dataa esimerkiksi serialisoidun PHP-objektin. (Introduction to Redis)



KUVA 1. Linked representation of Deque (Bhavikp19 2011-10-10).

Listat Rediksessä ovat merkkijonoja, jotka on lajiteltu lisäysjärjestyksen mukaan. Listaan voidaan lisätä uusia merkkijonoja joko luettelon päähän vasemmalle tai häntään oikealle (Kuva 1). Rediksen hajautustaulut sisältävät avaimien ja merkkijonoarvojen välisen yhteyden linkityksen, joten ne ovat täydellinen tietotyyppi objektien esittämiseen. Esimerkiksi Käyttäjä-niminen objekti, jolla on useita kenttiä kuten nimi, sukunimi ja ikä, on helppo esittää hajautustaulun avulla. Redis-sarjat ovat järjestämätön kokoelma merkkijonoja, joihin on mahdollista lisätä, poistaa ja kysyä jäsenten olemassaoloa listasta vakiokompleksisuudessa O (1) -ajassa riippumatta sarjan sisältämien elementtien lukumäärästä. (Data types)

3.1.1 Rediksen lajiteltu sarja

Hajautustaulujen lisäksi myös lajitellut sarjat sisältävät avain-arvo-parilinkityksen. Avaimia kutsutaan jäseniksi ja ne ovat uniikkeja (unique), eli lajiteltu sarja ei voi sisältää kahta samanlaista avainta. Jäsenet sisältävät myös pisteitä (scores), joiden perusteella jäseniä voidaan arvottaa eriarvoisiksi ja hakea jäseniä pisteiden arvojen perusteella. Pisteitä voi antaa vain liukulukuina (float). (Redis Labs, Sorted sets in Redis)

Rediksen lajiteltu sarja toimii kuin normaalit sarjat, eli näissä ei esiinny toistuvia merkkijonoja. Ero on siinä, että kaikkiin lajitellun sarjan jäseniin liittyy pistemäärä, jonka perusteella jäsenet lajitellaan pienimmästä suurimpaan. Käytettäessä lajiteltuja sarjoja voidaan jäseniä lisätä, muokata ja poistaa nopeasti suhteessa jäsenien lukumäärän logaritmiin. Luokiteltuun sarjaan jäseniä lisättäessä järjestetään ne lisäyksen yhteydessä jäsenien raaka-arvoihin perustuen. Lajittelu mahdollistaa jäsenien nopean hakemisen pisteiden ja sijainnin perusteella lajitellusta sarjasta. Jäsenien ollessa järjestettyinä voidaan sekä poistoja että hakuja suorittaa erinomaisella suorituskyvyllä lajitellusta sarjasta. (Data types) Lajiteltujen sarjojen mielenkiintoinen ominaisuus on sanakirjamainen indeksi (lexicographical indexes). Jäseniä lisättäessä samalla pistemäärällä ne lajitellaan vertaamalla sanakirjamaisesti merkkijonoja binaaritiedoiksi memcmp-funktion avulla. Lajitellun sarjan yhtenäisen pistemäärän sisältämät jäsenet lajitellaan vertaamalla niiden raaka-arvoja tavu tavulta läpi. Jos ensimmäinen tavu on sama, toinen tarkistetaan ja niin edelleen. Kahden merkkijonon etuliitteen ollessa yhtäläinen, pidemmän merkkijonon katsotaan olevan isompi näistä, joten ”foobar” on isompi kuin ”foo”. Rediksessä on komentoja, kuten ZRANGEBYLEX ja ZLEXCOUNT, jotka kykenevät kysymään ja laskemaan alueet sanakirjamaisesti olettaen, että niitä hyödynnetään lajiteltujen sarjojen kanssa, joissa kaikilla elementeillä on yhtenäiset pisteet. (Secondary indexing with Redis)

3.2 Redis verrattuna muihin tietokantoihin ja ohjelmistoihin

Redis ja MongoDB ovat NoSQL-tietokannoista käytetyimpiä relaatiotietokantojen skaalautuvuusongelmien ratkaisemisessa. Molemmat NoSQL-tietokannat pystyvät tallentamaan ja hakemaan tietoja käyttäjäystävällisesti, minkä lisäksi ne tarjoavat riittävän skaalautumisen ja tarkkuuden tietojen tallennukseen. NoSQL-tietokantojen eroja tutkiessa on olennaista tietää käyttötapaukset, joihin sovelluksesi täytyy vastata. Selkeyttääkseen käytettävän NoSQL-tietokannan valintaansa kehittäjä määrittää tarkasti, kumpi tietovarasto tukee paremmin tarvittavia ominaisuuksia tiedon määrän ja vaihtuvuuden perusteella. MongoDB:n ja Rediksen tiedonsiirron nopeutta vertaillessa Redis on näistä nopeampi, mutta vaatii tietokannan koon ennakointia nopeasti täyttyvän RAM-muistin vuoksi. Toisin kuin Redis, MongoDB pohjautuu dokumenttipohjaiseen ratkaisuun ja säilöö tietoja levyllä. (AppWorks)

Nykyään kehittäjillä on mahdollisuus valita monesta erilaisesta tietovarastotyypistä. Tietokantojen tyypit voi jakaa kahteen eri tyyppiin, NoSQL- ja SQL-tietokantoihin. Näiden tyyppien vertailemisesta haastavaa tekee se, että kaikki tietokannat, jotka eivät ole SQL-tietokantoja voidaan määritellä olevan NoSQL-tietokantoja. Ero näiden tyyppien välillä syntyy relaatioden käytöstä tietojen välillä. SQL

tietokannat käyttävät tietojen yhdistämiseen erilaisia relaatioita tietojen välillä, toisin kuin NoSQL-tietokannat. NoSQL tyyppisiin tietokantoihin hyväksytään jokainen tietokantatyyppi, oli se sitten avain-arvo-pari, dokumentti tai tapahtumapohjainen tietokanta. Suorituskykyä miettiessä on hyvä ottaa huomioon sovelluksesi toimintojen lukumäärät, ja onko niitä mahdollista suorittaa yksittäin vai ryhmässä. Yksittäisissä kirjoitus- ja lukutoiminnoissa NoSQL- tietokannat ovat nopeampia, kun taas SQL on suorituskyvyltään parempi isoissa ryhmissä lukiessa ja kirjoittaessaan. Tietokantatyyppi valitaan sovelluksen prosessien lukumäärän perusteella. (Majerczyk, 2019)

3.3 Rediksen muut ominaisuudet

Redistä hyödynnettäessä kysymyksiä tulee siitä, miten tiedot säilyvät, kun ne ovat muistin sisäisessä tietokannassa eivätkä tallennettuna levyllä kuten relaatiotietokantojen palvelimen sammuessa. Tähän Rediksessä on kaksi erilaista muistitietojen tallennustoimintoa, jotka tallentavat tietoa levylle palvelimen ollessa vielä käynnissä. (Redis Labs, Other Features) Ensimmäinen näistä on tietojen tallennus levylle tiettynä ajankohtana: Toiminto tallentaa RDB-tiedoston, joka sisältää muistin sisäisen tietokannan binäärisen esitysmuodon. Esimerkiksi 10 minuutin välein tallennetun tiedoston avulla pystytään palauttamaan tietokanta toimintakuntoon nopeasti palvelimen kaatumisen jälkeen. Tallennuksien välillä tapahtuneiden operaatioiden tiedot palautuvat palautuksessa aikaisempaan versioon, joten on varauduttava tietojen menettämiseen. Toinen tietojen tallennustoiminto perustuu kirjoitusoperaatioiden tallentamiseen AOF-lokitiedostoon operaatioiden tapahtuessa tai asetetun viiveen perusteella: Tallennustoiminto aktivoidaan vain kirjoitusoperaatioiden yhteydessä, joten tallennettujen operaatioiden perusteella voidaan palauttaa tietokanta aikaisempaan versioon vain suorittamalla lokitiedosto alusta loppuun yksi operaatio kerrallaan. AOF-lokitiedoston kirjoittaminen vaatii enemmän suoritustehoa kuin RDB-tiedoston kirjoittaminen, minkä lisäksi AOF-lokitiedosto on tiedostokooltaan isompi kuin RDB-tiedosto. (Redis Persistence)

Redis tukee sisäänrakennetun Lua-komentotulkin avulla EVAL- ja EVALSHA-ohjelmaskriptien kirjoittamista sekä suorittamista (EVAL script). Useimmille Rediksessä käytetyille avaimille voi asettaa automaattisen poistoajankohdan millisekuntien tarkkuudella. TTL-toimintoa (time to live) hyödyntämällä on mahdollista vähentää tarpeettomien tietojen pysymistä Rediksen tietokannan muistissa ja vähentää tulevaisuudessa tarvittavia poisto-operaatiota asettamalla alhainen poistoajankohta avaimille niiden luonnin yhteydessä. (EXPIRE key seconds)

4 HAKUINDEKSI

Hakuindeksi koostuu indeksoitavista tiedoista eli asiakirjakokoelmista (corpus) (Arden Dertat). Hakukoneen etsiessä tietyn kyselyn tuloksia asiakirjakokoelmasta viittaa se silloin jäsenneltyyn tietojen kokonaisuuteen eli indeksiin. Käytettyjen algoritmien tulee olla personoituja hakuindeksin nopean toiminnan takia. Hajanaisten tietojen indeksoinnin aikaisesta prosessista on silloin kyse, kun tiedot jalostetaan hakuindeksin algoritmin kelpuuttamaan strukturoituun muotoon. Indeksejä voidaan ajatella yksinkertaisesti hajallaan olevien tietojen yhdistäjänä. Tietojen yhdistäminen indeksiin nopeuttaa tiedonhakua ja mahdollistaa hakualgoritmin kehittämisen hakuindeksille. Hakualgoritmia hyödyntäen voidaan järjestetyistä tiedoista tunnistaa olennaisia tietoja, tietojen ollessa valmiiksi jaoteltuina hakuindeksissä. (Swiftype)

4.1 Hakuindeksin tietorakenteet

Hakuindeksien tietorakenteista yleisin on käänteinen indeksi: tietorakenne, joka tallentaa linkityksen sanojen ja numeroiden sijaintien välillä asiakirjoissa. Käänteisen indeksin perusteella voidaan nopeasti hakea ja siirtyä löydettyyn asiakirjaan verkkosivustolla. Käänteiset indeksit jaetaan yleisesti kahteen eri tyyppiin, sanatasoiseen ja asiakirjatasoiseen indeksiin. Sanatasoiset käänteiset indeksin sisältävät linkityksen asiakirjan jokaisen sanan ja sijainnin perusteella. Asiakirjatasoinen käänteinen indeksi ei sisällä sanan sijaintia asiakirjassa, joten se vaatii vähemmän prosessointitehoa ja tilaa kuin sanatasoinen käänteinen indeksi. (Jain, 2019)

Käänteisten indeksien sanojen ja numeroiden linkityksessä asiakirjoihin käytetään useasti lajiteltuja sarjoja. Lajiteltu sarja pitää sisällään merkkijonon ja numeron välisen linkityksen asiakirjaan. Monissa tapauksissa lajitellut sarjat pitävät sisäisesti huolta lisättyjen arvojen ainutlaatuisuuden ylläpitämisestä. Etuliitettä hash-puu (prefix hash tree) voidaan käyttää normaalin hakupuun sijasta ajan säästämiseksi haetun tiedon palauttamisessa, koska jokainen etuliite sisältää myös sen osoittaman tiedon samassa puussa. Hash-puun heikkouksena voidaan mainita sen vaativan enemmän tilaa säilyttääkseen etuliitteen ja haettavan tiedon välisen yhteyden. Sen vuoksi on tärkeää miettiä, ollaanko valmiita vaihtamaan tilaa lukunopeuden parantamiseksi hash-puu-tietorakennetta hyödyntäen. (Building Prefixy)

4.2 Hakuindeksin luonti

Hakuindeksin luonnista löytyy erilaisia toteutuksia ja rakenteita tietojen säilyttämiseen Rediksessä. Asiakirjoista sanojen nopeampi etsiminen vaatii tietojen esikäsittelyn ennen niiden lisäämistä hakuindeksiin. Tätä toimintoa kutsutaan indeksoinniksi. Yleensä hakuindeksi luodaan käyttäen käänteistä indeksiä. (Redislabs, Basic search theory) Ennen moduulien ilmestymistä Redikseen, kokonaistekstihaku (full-text search) suoritettiin natiivien Redis-komentojen avulla. RedisSearch-moduuli antaa paremman maksimitehon kuin mainituilla natiiveilla komennoilla luotu kokonaistekstihaku. Kaikissa ympäristöissä moduulien hyödyntäminen ei kuitenkaan ole sallittua, joten natiiveilla komennoilla luotuja hakuindeksejä käytetään ja jalostetaan yhä. (Redis Labs, Full Text Search)

Ennen tietojen lisäämistä hakuindeksiin suositellaan tehtäväksi tietojen tokenisointi. Yksinkertaisimmillaan tokenisoinnissa on kyse tiedon normalisoinnista esimerkiksi sanan isojen kirjaimien muuntamisesta pieniin kirjaimiin. Tokenisaation jälkeen hakuindeksistä on mahdollista hakea tietoja välittämättä isoista ja pienistä kirjaimista. (Sam, 2018)

Tärkeintä hakemisessa ovat hakuindeksit, jotka säilyttävät luetteloa haettavista tiedoista ja rajoittavat hakujen laajuutta. Asiakkaan hakiessa tietoja sivustosta haetaankin välttämättömät tiedot hakuindeksistä eikä tietovarastosta. (Kentico 8 Documentation)

4.2.1 PHP

PHP (Hypertext Preprocessor) on ohjelmointikieli, jota hyödynnetään etenkin web-sivujen kehittämisessä. Sen ohjelmakoodia tulkataan vasta sovelluksen suoritusvaiheessa, joten se kuuluu komentosarjakieliin. PHP:llä kirjoitettu ohjelmakoodi antaa kehittäjälle mahdollisuuden olla vuorovaikutuksessa luomansa web-sivun kanssa. Koodin parsinta eli tulkkaus tapahtuu palvelinpuolella eikä asiakkaan selaimessa, kuten monissa muissa web-kehittämisessä käytetyissä ohjelmointikielissä. (2Kmediat) Unix-pohjaista ajastuspalvelu Cronia hyödyntämällä voidaan ajastaa automatisoitu komentosarja, jolla hakuindeksin luonti voidaan automatisoida (Code Institute).

4.2.2 Haku indeksistä

Indeksille laadittu hakuehto palauttaa hakuindeksin dokumentit, joihin hakuehto kohdistuu. Hakuindeksi palauttaa yleisiä tietoja löydetyistä dokumenteista. Yleisesti tunnettu hakuindeksi on paperinen puhelinluettelo, josta löytyy aakkosjärjestyksessä etunimet, sukunimet ja yrityksien nimet. Järjestettyjen nimien perusteella löydät niiden viittaaman puhelinnumeron. (AddSearch, 2020).

Hakuindeksin luonnissa käytetty indeksointityyli vaikuttaa siihen, kuinka tietoja voidaan hakea eri hakuehdoilla. Hakuindeksin sisältäessä vain dokumentin sanojen ja yksilöllisen tunnistenumeron välisen käänteisen indeksin voidaan tietoja hakea vain etuliitteen avulla. Lauseiden purkaminen erillisiin sanoihin ja käänteisen indeksin luominen jokaiselle sanalle ja yksilöllisen tunnistenumeron välille mahdollistaa hakemisen myös sanalla lauseen keskeltä. Edellä mainittu toiminto vaatii huomattavasti muistikapasiteettia hakuindeksiltä, joten se on harvemmin käytetty tapa. Hakeminen indeksistä toteutetaan vertailemalla käänteisiä indeksejä toisiinsa ja löytämällä liitokset indeksien välillä, jotka sisältävät etsityt dokumentit. (Redislabs, Basic search theory)

5 HAKUINDEKSIN KÄYTTÖLIITTYMÄSSÄ KÄYTETYT OHJELMOINTITEKNIIKAT

5.1 React

React eli ReactJS on JavaScriptin luokkakirjasto, jonka avulla on mahdollista tehdä interaktiivisia käyttöliittymiä. Se on ollut alun perin Facebookin yksityisessä käytössä vuodesta 2011 lähtien. Reactin kehitykselle tärkeää oli siirtyminen suljetusta lähdekoodista avoimeen lähdekoodiin mahdollistaen nopeamman käyttöönoton kehittäjien parissa. (Hámori, 2018) Reactin avulla voidaan suunnitella yksinkertaiset näkymät jokaiselle tilalle sovelluksessa, minkä lisäksi React osaa sekä päivittää, että tuottaa oikeat komponentit tehokkaasti, kun tiedot muuttuvat näkymässä. React mahdollistaa suurien ja monimutkaisten sovelluksien käyttöliittymän tekemisen helpommaksi pilkkomalla suuret näkymät pienempiin yksittäisiin komponentteihin. Komponenteille on määritelty oma tilansa, jota ne hallitsevat. Komponentti näyttää tai piilottaa tietoja käyttöliittymällä asetettuun tilaan perustuen. (Facebook, 2020)

Komponentteja on kahta eri tyyppiä, funktionaalisia ja luokkakomponentteja. Selkein ero näiden komponenttien välillä on syntaksi. Funktionaalinen komponentti on JavaScriptin funktio, joka ottaa vastaan parametreja argumentteina ja palauttaa React-elementin. Luokkakomponentin luonti vaatii komponentin laajentamisen Reactin yleiseen komponenttiin ja renderöintifunktion luomisen, joka palauttaa React-elementin. Ennen Reactin versiota 16.8 funktionaalinen komponentti ei pystynyt säilömään tilatietoja, mutta tässä mainitussa versioissa julkaistu uusi toiminnallisuus ”koukut” (hooks) teki tilatiedon säilömisen mahdolliseksi myös funktionaalisessa komponentissa. Koukkujen käyttäminen tekee lähdekoodin lukemisesta helpompaa, ymmärrettävämpää ja testattavampaa, koska funktionaaliset komponentit ovat puhtaita JavaScript-funktioita. (Jöch, 2018)

5.2 TypeScript

TypeScript on Microsoftin kehittämä ja ylläpitämä avoimen lähdekoodin ohjelmointikieli. Se auttaa kehittäjiä tekemään JavaScriptistä tyypitetympää käyttämällä apuna TypeScriptin tuomaa vahvempaa tyypitystä. TypeScript ei ole kokonaan oma kielensä vaan se lisää JavaScriptiin uusia ominaisuuksia. Vahvan tyypityksen avulla kehittäjiltä vaaditaan vähemmän testaamista eri tyypeille, mikä vähentää tyyppien välisien muunnoksien aiheuttamia ongelmia. Tyypeille voidaan määritellä omat nimensä käyttäen TypeScriptin rajapintaluokkia (interfaces). Määrittelemällä TypeScriptin tyypit tarkasti luodaan sopimuspohjaista ohjelmointia hyväksikäyttäen selkeä sopimus koodin ja tietokoneen välille. (Tarvainen, 2016)

Tyyppimerkinnät (annotations) TypeScriptissä ovat kevyitä tapoja tallentaa funktion tai muuttujan aiottu sopimus (Typescriptlang). Yleistäen voidaan sanoa, että vahvasti tyypitetyn kielen kirjoittaminen on vaivalloisempaa, mutta se helpottaa erityisesti suurien ohjelmistojen ylläpitoa ja mahdollistaa paremman suorituskyvyn. Tämä johtuu siitä, että ohjelmoijan komennot tulevat koneelle selkeämmäksi. (Tarvainen, 2016)

Tyyppien spesifiointi avustaa myös kehittäjiä, jotka hyödyntävät ohjelmointiympäristöä. Tämä edistää TypeScriptin tyyppien ja rajapintaluokkien voimin dynaamisesti luotua toimintojen loppuunsaattamista, mikä poistaa mahdollisia lyöntivirheitä. Eräs monista ohjelmointiympäristöistä, joka tukee kyseisiä palveluita, on PhpStorm. PhpStormin TypeScript-tietoinen koodausapu kattaa avainsanojen, nimikkeiden, muuttujien, parametrien ja toimintojen dynaamisen loppuunsaattamisen, virheiden ja syntaksien voimistamista, muotoilun, staattisen kooditarkistuksen ja pikaparannus ehdotukset sekä yleisen, että TypeScript spesifisen refaktoroinnin. (PhpStorm, 2020)

5.3 GraphQL

GraphQL on kehitetty vastaamaan uusien käyttöliittymätekniikoiden mahdollistamaa vain tarvittavan datan hakua, jossa palvelin ei palauta perinteisesti selaimelle aikaisemmin lähetettyjä kokonaisia sivuja ulkoasuineen ja datoineen (Tarvainen, 2016). Sen kehittämisen on aloittanut Facebook vuonna 2012 sisäiseen käyttöön yrityksen natiiviissa mobiiliapplikaatiossa. Yrityksen mobiiliapplikaatio IOS- ja Android-käyttöjärjestelmille oli vain ohut kerros näkymien ja mobiiliverkkosivustonsa välissä, mikä aiheutti ongelmia suorituskyvyssä ja johti yllättäviin mobiiliapplikaation kaatumisiin. Tämän perusteella yritys alkoi kehittämään GraphQL:ää yhdessä suunnittelijoiden ja kehittäjien kanssa perustuen ”Kirjoita kerran, suorita missä tahansa” -periaatteeseen. Periaate auttoi yritystä miettimään uudelleen mobiiliapplikaatioiden välisen datan siirtoa API:ien välityksellä. (Byron, 2015) Facebook ilmoitti virallisesti GraphQL:n julkaisemisesta avoimen lähdekoodin alaisuuteen vuonna 2015, mikä on sittemmin nostanut GraphQL:n suosiota kehittäjien parissa merkittävästi. GraphQL on määritelmä, ei kirjasto, tuote eikä tietokanta. (Giroux, 2019) Sen suurin ero REST:in verrattuna on se, että teoriassa sovellukset, jotka tukevat GraphQL määritelmää ja skeemaa voivat keskustella tietämättä toistensa käytämästä http-protokollasta tai URL-päätteistä tietoja hakiessa tai muokatessa. (Tarvainen, 2016).

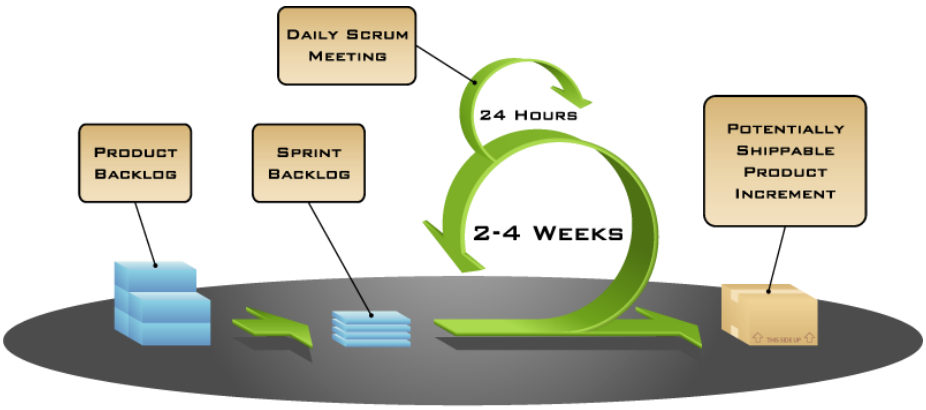
GraphQL on vahvasti tyypitetty samoin kuin TypeScript, minkä takia ne toimivat hyvin yhdessä vahvasti tyypitetyn sovelluksen rakentamisessa. Kentät GraphQL:ssä ovat määritelmä tietystä objektin tiedosta. Kentän tietoja haetaan ja muokataan käyttämällä GraphQL:n kyselyitä ja mutaatioita. (Queries and Mutations) Argumenttien avulla voidaan hakea, muokata tai poistaa tiettyjä objekteja käyttäen argumenttina välitettävää tietoa, esimerkiksi objektin ainutlaatuista tunnistenumeroa (Arguments). GraphQL:n kyselyitä voi tehdä monta yksittäistä tai ne voidaan yhdistää yhdeksi isommaksi, joilloin ne suoritetaan rinnakkain. Mutaatioita voi myös yhdistää, mutta ne suoritetaan aina sarjassa toinen toisensa jälkeen. (Multiple fields in mutations)

6 TYÖN SUUNNITTELU

Opinnäytetyöni alkumäärittelyissä tehtäväksi määriteltiin automaattisen täydennyksen toiminnallisuuden omaavan pikahakukomponentin suunnittelu ja rakentaminen asiakkaiden toiveiden mukaisesti. Komponentti tuli rakentaa Ropo Capitalin uuden käyttöliittymän kojelautaan. Käyttöliittymän tyylien ja ulkoasun osalta pikahakukomponentin tuli vastata annettuja käyttöliittymän rautalankamalleja (eng. wireframe). Pikahaun tuli olla nopea ja helppokäyttöinen. Suunnittelupalaverissa määriteltiin pikahaun käyttämän hakuindeksin tietovarasto, jollaiseksi valikoitui Redis. Pikahaun tuli sisältää seuraavia asioita asiakkaan sisällöstä: tehtäviä, asiakkaita ja tuotteita. Hakuindeksiin tuli laittaa kyseiset tiedot yhtenäisessä muodossa.

Alkumäärittelyiden jälkeen suunnitteluprosessi aloitettiin laajentamalla teoriapohjaa Rediksen käytöstä hakuindeksinä. Lisäksi tutkittiin opinnäytetyön vaatimusten täyttämiseksi lähdekriittisesti tietoa erilaisista verkkosivustoista, tietokannoista, artikkeleista ja näiden tietojen avulla luotiin luotettavuutta työlle. Näiden jälkeen selvitettiin osana suunnitteluprosessia hakuindeksin luonnin vaatima tietojen esikäsittelyn toimintaperiaate. Opinnäytetyölle muodostui otsikko aikaisessa vaiheessa, myös keskeiset käsitteet muodostuivat selkeästi projektin alkuvaiheessa. Nämä olivat NoSql-tietokanta, Redis, ReactJs ja automaattinen täydennys. Selkeä otsikko ja keskeisien käsitteiden määrittely helpottivat suunnitteluprosessia ja työn viitekehyksen muodostumista.

Projekti oli laajuudeltaan suuri, minkä vuoksi laajojen kokonaisuuksien pilkkominen pienempiin osiin oli tärkeä osa suunnitteluprosessia. Projektin sisältämät työvaiheet suunniteltiin ja kirjoitettiin määritelmät toteutuksen jokaiselle osakokonaisuudelle. Osakokonaisuudet olivat JIRA tehtävienhallintaohjelmiston avulla luotujen tehtävien osatehtäviä, joiden avulla huolehdittiin, että kokonaisuus valmistuu projektille asetettuun aikatauluun mennessä. Hakuindeksin tietorakenteen suunnittelu ja määrittely oli tärkeä osa suunnitteluprosessia. Tietorakenteen suunnittelussa otettiin huomioon tietoihin kohdistuvien hakulauseiden erilaiset tietotyypit.



KUVA 2. Scrum Diagram. (Payravi 2014-06-24).

Projektin suunnittelu ja viitekehys toteutettiin scrum projektinhallinnan menetelmää käyttäen. (Kuva 2) Testausautomaatio oli isossa osassa suunnittelua koko projektin ajan. Ennen työn toteutusta

suunniteltiin ja määriteltiin tarkasti erilaiset testausautomaatiot pienemmille kokonaisuuksille. Suunnittelun tavoitteena oli työstää projektia pikkuhiljaa kohti valmiimpaa tuotetta. Yrityksessä on käytössä scrum-projektinhallinnan viitekehys, mihin perustuen pidettiin kahden viikon välein suunnittelupalavereita kehitystiimin jäsenten kesken.

Tärkeä painoarvon suunnitteluprosessissa asetettiin myös aikataulun suunnittelemiselle, sillä suunnitelmallinen ote ja ajallisten resurssien optimointi opinnäytetyöprosessissa olivat tärkeässä asemassa. Työpaikkani asettama deadline loi minulle selkeän tavoitteen saavuttaa projekti valmiiksi annetun aikataulun puitteissa. Aikataulussa pysymistä seurattiin säännöllisesti työpaikallani kahden viikon sprinteissä. Säännöllinen yhteydenpito opinnäytetyön ohjaavaan opettajaan mahdollisti hyvän ja toimivan kommunikoinnin opinnäytetyön edistymisestä. Suunniteltiin opinnäytetyön valmistuvan loppukevääseen 2020 mennessä.

7 TYÖN TOTEUTUS

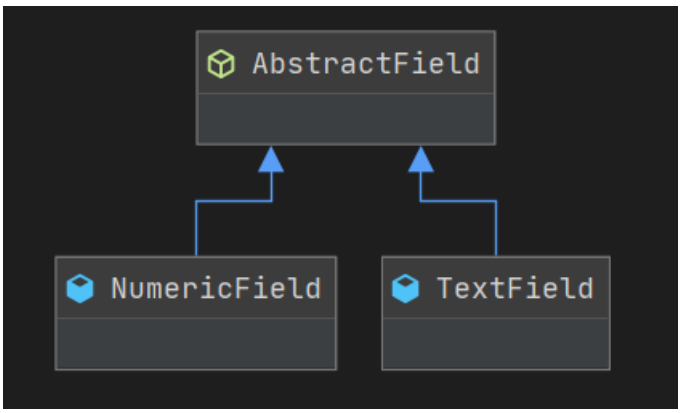
Sovellus tuotettiin mukaillen Scrumin ja ketterän kehityksen menetelmiä. Sovelluksen kehityksessä otettiin huomioon koodauskielten parhaat kehityskäytännöt sekä toteutettiin sovellus vastaamaan sille asetettuja määrittelyitä. Työn toteutuksessa käytettiin hyödyksi hajautettua versionhallintajärjestelmää nimeltä Git. Työn osien ollessa pilkottuna pieniksi osakokonaisuuksiksi alkoi työn toteutus uuden kehityshaaran luonnilla. Pieniin kokonaisuuksiin jaettujen osatehtävien avulla sovellus kehittyi koko ajan kohti valmiimpaa tuotetta pienissä osissa ja tuotetta ei tarvinnut suunnitella kerralla toteutuskuntoon.

7.1 Back-end

Back-end toteutettiin käyttämällä PHP-ohjelmointikieltä, noudattaen PSR-12 koodityylin standardia ja noudattaen SOLID-kehitysperiaatteita. Kehityksessä noudatettiin yrityksen asettamia ohjelmistoarkkitehtuurisia käytänteitä ohjelmointikoodille ja testaukselle.

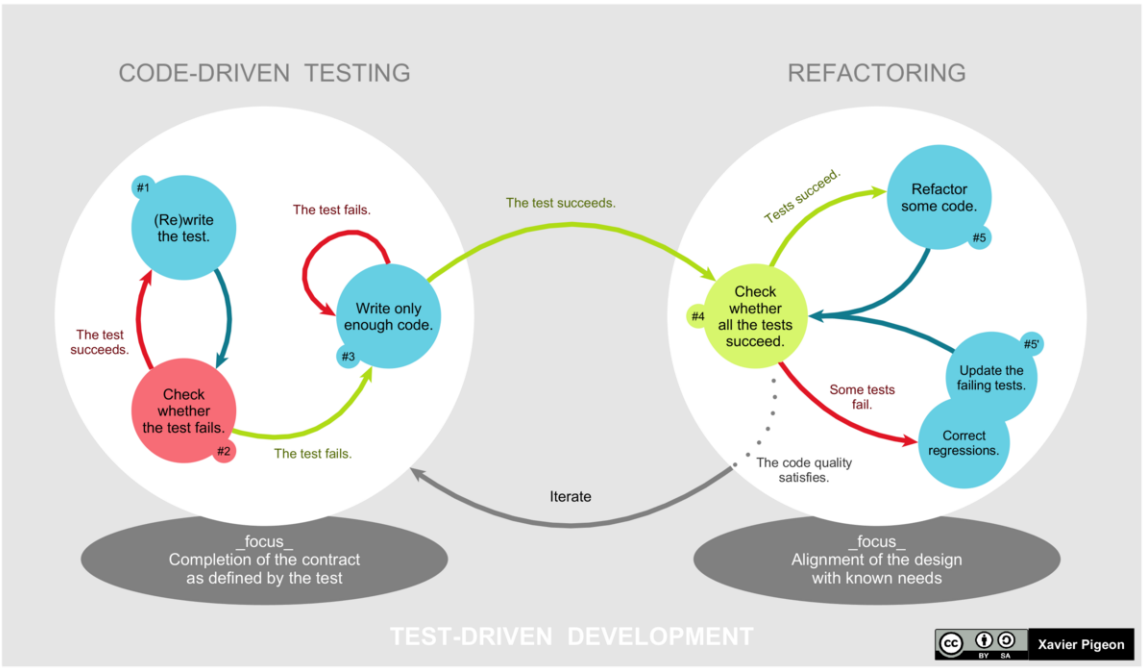
Työn back-endin toteutus alkoi PHPUNIT-automaatiotestien kirjoittamisella vastaamaan asetettuja määrittelyitä työn kokonaisuudelle. Sovelluksen pitää olla testattu yksikkö-, komponentti- tai funktionaalisin automaatiotestein ennen sen hyväksymistä yhteiseen kehityshaaraan. Ensimmäisenä vaiheena oli selvittää ja asentaa luokkakirjasto PHP:lle, mikä tekee Rediksen käytöstä helpompaa ja yksinkertaisempaa. Luokkakirjastoksi valikoitui Predis sen ollessa monipuolinen sekä joustava asiakasohjelma (client) PHP:n ja Rediksen väliselle kommunikoinnille. Predis asennettiin käyttäen Composeria, joka on riippuvuuksien hallintatyökalu PHP:lle. Seuraavana tehtiin tarkastelu ohjelmistokoodin oikeellisuudelle ja täytettiin lista sekä tarkastettiin organisaation asettamat määritelmät valmiille sovellukselle. Aikaisemmat vaiheet suoritettiin paikallisessa haarassa (branch), joka oli haarautettu yleisestä kehityshaarasta, joten muutoksien commitointi eli tallentaminen suoritettiin paikalliseen haaraan. Paikallisen haaran muutoksien julkaisemisella eli puskemisella viedään muutokset yleiseen etäsäilöön, josta haaran muutokset voidaan yhdistää yleiseen kehityshaaraan tekemällä siihen pyyntö versiohaarojen yhdistämisestä (pull request). (Lofhjelm)

Asennuksen ja versiohaarojen yhdistämispyynnön ollessa valmis sekä ohjelmakoodin noudattaessa organisaation koodauskäytänteitä, voidaan se hyväksytysti laittaa katselmointiin eli ohjelmiston tarkastukseen. Ohjelmiston tarkistuksen on tarkoitus löytää viat ja puutteet sovelluskoodista. Tätä kutsutaan yleisesti koodin katselmoinniksi. Koodin katselmointi on tärkeä osa sovelluskehitystä, sillä se auttaa virheiden huomaamisessa sekä käytetyn koodauskielen kehityskäytänteiden noudattamisen tarkistuksessa. (Meyer, 2018) Koodin katselmoijat päättävät - joko hylkäämällä tai hyväksymällä yhdistämispyynnön - täyttääkö sovelluskoodi sille asetut vaatimukset. Yhdistämispyynnön mahdollisen hyväksynnän jälkeen katselmoijat yhdistävät (merge) muutokset yhteiseen kehityshaaraan, jolloin muutokselle suoritetaan automaatiotestit.



Kuva 3. Tyyppiluokkien luokkakaavio. (Kupari, 2020)

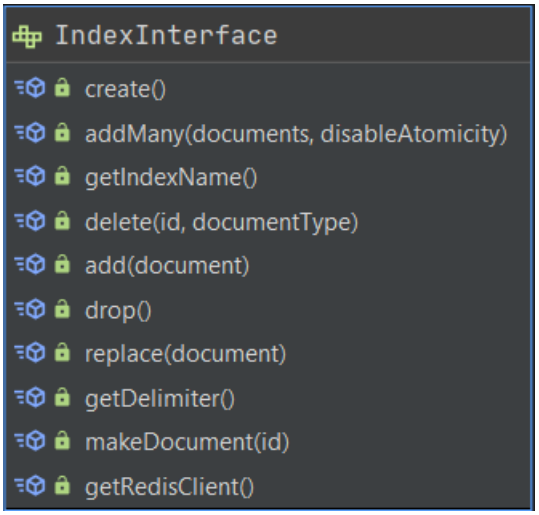
Muutoksien ollessa kehityshaarassa voidaan jatkaa uusien toimintojen kehittämistä. Predis-kirjaston asennuksen jälkeen alettiin rakentamaan automaattisen täydennyksen tarvitsemaa hakuindeksiä asiakkaiden tiedoista. Hakuindeksin vaatimuksena oli, että sen on mahdollistettava tietojen hakeminen sekä numeraalisella että tekstityyppisellä hakuehdolla. Hakuindeksin rakentaminen alkoi tietojen tyyppien määrityksellä. Määrityksien pohjalta tehtiin abstrakti luokka nimeltä AbstractField (Kuva 3), joka sisälsi yleiskäyttöisen rungon sen periville luokille. Numeraaliset sekä tekstityyppiset tiedot määriteltiin omiin luokkiinsa, jotka perivät abstraktin luokan ja toteuttavat niille määritellyt metodit.



Kuva 4. Lifecycle of the Test-Driven Development method (Xarawn 2015-11-4)

Tyyppiluokkien metodeille kirjoitettiin yksikkötestit käyttäen kehityksessä testivetoista kehitystä (Kuva 4), eli aloitettiin kehitys automaatiotestien kirjoittamisella. Yksikkötestit testasivat erityyppisillä tiedoilla luokkien palauttamaa tyyppitietoa. Seuraavaksi luotiin luokan luontimallia noudattaen tehdasfunktio, joka kykenee tunnistamaan parametrina annetun tiedon tyypin ja palauttamaan sitä vastaavan tyyppiluokan.

Hakuindeksiin on tarpeellista tallentaa tiedot yhtenäisessä muodossa, joten aikaisemmin luoduille tietokentille rakennettiin dokumentti, joka voi pitää sisällään montaa erityyppistä tietokenttää. Dokumentti on ryhmä tietokenttiä ja arvoja, jotka tallennetaan hakuindeksiin tiedon tyypin perusteella. Dokumentille luotu luokka noudattaa sille asetettua rajapintaluokkaa, joka pakottaa luokalle tietyt metodit ja paluuarvot. Luokan toteuttaessa rajapintaluokkaa se allekirjoittaa ja lupaa luokan sisältämien metodien olemassaolon, muutoin sovellus ei toimi. Dokumentti-luokka toteuttaessaan rajapintaluokkaa noudattaa ohjelmistojen suunnittelutapaa, joka on tyypiltään sopimuspohjaista ohjelmointia. Dokumenteille luotiin myös luokan luontimallia käyttäen tehdasfunktio, jonka avulla voidaan luoda dokumentteja suoraan PHP:n assosiatiivisesta taulukosta, taulukon tietojen tyypit tunnistetaan ja luodaan tyyppiä vastaava tietokenttä-arvo-pari dokumentille.



Kuva 5. Hakuindeksin rajapintaluokka. (Kupari, 2020)

Automaattisen täydennyksen on oltava asiakaskohtainen, joten jokaisen hakuindeksin pitää olla uniikki, etteivät asiakkaat voi hakea toistensa tietoja. Rediksessä tietojen palauttamista ei voi hallita asiakaskohtaisesti, joten on luotava luokka, joka hallitsee hakuindeksin hakua ja luontia asiakaskohtaisesti. Asiakaskohtaisen hakuindeksin luonnissa käytettiin eristävänä tekijänä uniikkia numerosarjaa. Hakuindeksin luonnin yhteydessä luodaan valmiiksi määritellyt dokumenttityypit: asiakkaan tehtävät, asiakkaat ja tuotteet. Näin ollen jokaiselle asiakkaalle luotiin kolme erityyppistä automaattisen täydennyksen indeksiä perustuen dokumenttityyppeihin. Hakuindeksiä hallitsevan luokan avulla suoritetaan dokumenttien sekä hakuindeksin luonnit asiakaskohtaisesti. Hakuindeksin luokalle luotiin myös rajapintaluokka, joka sisälsi metodit ja niiden palautusarvot (Kuva 5). Hakuindeksiluokalle määritellään käytettävä Redis-asiakasohjelma, joka on Predis-luokkainstanssi. Asiakaskohtaisen hakuindeksiluokan automaatiotestien ollessa valmiit, siirryttiin hakuindeksin luontivaiheeseen eli asiakkaan tietojen hakemiseen tietokannasta ja tietojen lisäämiseen rediksen hakuindeksiin. Tietokantahakuja voidaan tehdä vain tietojen käyttökerrosluokan metodien kautta, jotta voidaan pitää sovelluksen kerrokset eristettyinä toisistaan. Hakuindeksiin tallennettavien tietojen määrä on valtava, joten tietokantahaut pitää tehdä pienempiin osiin jaettuina. Tietokantakyselyt jaettiin pienempiin osiin sovelluksen palvelukerroksessa, joka hallitsee tietojen käyttökerrosluokan metodien kutsumista ja tiedon välittämistä sitä pyytäville esityskerroksille.

Automaattisen täydennyksen hakuindeksin luonti toteutetaan ajastettuna ajona hyödyntäen sovelluksen rinnakkaista suoritusta hakuindeksin luontiin kuluvan ajan vähentämiseksi. Ajastettu ajo hakee tietokannasta asiakkaat, jakaa asiakkaat ryppäisiin ja käynnistää erilliseen prosessiin hakuindeksin luonnin eri dokumenttityypeille. Erillisiin prosesseihin käynnistetyt hakuindeksin luonnit hakevat annetun parametrin perusteella tietoja tietokannasta käyttäen tietojen käyttökerrosluokan metodeja. Lisäksi prosessit luovat haetuista tiedoista asiakaskohtaisen hakuindeksin Redikseen hyödyntäen aikaisemmin luotuja metodeja indeksin luontiin liittyen. Asiakaskohtaisen hakuindeksin luokka esikäsittelee tietokannasta haetut tiedot, luo niistä dokumentit ja tallentaa ne määritellyssä muodossa Redikseen. Hakuindeksi tallennetaan Rediksen lajiteltuun sarjaan, josta tietoja voidaan hakea nopeasti, kunhan tiedot on esikäsitelty oikein. Ennen tietojen tallentamista hakuindeksiin muutettiin kaikkien merkkijonojen kirjaimet pieniksi kirjaimiksi. Hakuindeksin merkkijonojen ollessa vain pieniä kirjaimia, voidaan muokata käyttöliittymältä tuleva hakuehto myös pienille kirjaimille, jolloin haku tehdään merkkikokoriippumattomasti. Rediksen lajitellun sarjan luontiin ja hakemiseen käytettiin Predis-luokkakirjaston tarjoamia komentoja ZADD ja ZRANGEBYLEX. ZADD-komennolla lisätään tietoja lajiteltuun sarjaan ja tallennetulle tiedolle annetaan pisteet, joiden perusteella Redis lajittelee tiedot lajitellussa sarjassa. Tiedoille asetettiin samat pisteet, koska silloin hakuindeksi on mahdollista rakentaa helpommin, Rediksen pitäessä huolta lisättyjen tietojen järjestyksestä, jolloin tietoja voidaan hakea nopeasti niiden ollessa jo järjestyksessä.

Haettava tieto	Erotinmerkki	Dokumentin yksilöllinen tunniste	Erotinmerkki	Avaimen nimi	Haettavan tiedon yhtenäinen muoto hakuindeksissä
ropo capital oy	::	12345	::	yritys	ropo capital oy::12345::yritys

Kuva 6. Hakuindeksin dokumentin rakenne. (Kupari, 2020)

```
127.0.0.1:6379> ZADD AsiakasIndeksi::54321::Asiakkaat 0 "ropo capital oy::12345::yritys"
(integer) 1
```

Kuva 7. Jäsenen lisäys lajiteltuun sarjaan. (Kupari, 2020)

Hakuindeksiä luodessa tiedot tallennettiin lajiteltuun sarjaan kuvan (Kuva 6) mukaisesti. Haettava tieto ensimmäisenä, toisena dokumentin yksilöllinen tunniste ja viimeisenä arvona avaimen nimi. Arvot eroteltiin toisistaan käyttämällä erotinmerkkiä hakualueen rajaamisen helpottamiseksi. Rediksen ZADD-komento lisää lajiteltuun sarjaan jäsenen (Kuva 7), kuvassa 6 olevalle dokumentille asetetaan liukuluku arvo 0 kuvan 7 mukaisesti.

Rediksen lajitellusta sarjasta - tässä tapauksessa luodusta hakuindeksistä - haetaan tietoja käyttäen komentoa ZRANGEBYLEX. ZRANGEBYLEX-komennon pakolliset parametrit ovat lajitellun sarjan avain, hakualueen aloituskohta ja hakualueen lopetuskohta. Optionaalisina parametreina voidaan ZRANGEBYLEX-komennolle antaa palautettavien tietojen lukumäärä sekä määrittää kuinka monta riviä määritellystä alkukohdasta eteenpäin otetaan mukaan haettuun tietosisältöön.

```
127.0.0.1:6379> ZRANGEBYLEX AsiakasIndeksi::54321::Asiakkaat "[ropo" "[ropo\xff::]"
1) "ropo capital oy::12345::yritys"
```

Kuva 8. Jäsenen hakeminen lajitellusta sarjasta. (Kupari, 2020)

Kuvassa 8 olevalla ZRANGEBYLEX-komennolla haetaan kuvassa 7 luotu jäsen lajitellusta sarjasta hakusanalla ”ropo”. Asetetaan hakualueen aloituskohdaksi ”[ropo” ja lopetuskohdaksi ”[ropo\xff::]”, missä aloituskohdan ”[ropo” kertoo hakualueen olevan poissulkeva kaikille muille merkkijonon arvoille kuin ”ropo”. Lopetuskohdan hakualue ”[ropo\xff::]” määrittää myös poissulkevana hakualueen lopetuskohdan ja ”\xff” ollessa suurin merkkijono muokattuna binaarimuotoon, sitä käytetään yhdessä haettavan merkkijonon kanssa saadakseen kaikki merkkijonot asetetusta hakualueesta. Rajaamalla vielä lopetuskohdan hakualuetta pois sulkevasti ”::]” avulla pienennetään hakualuetta vielä seuraavaan erotinmerkkiin, jolloin haut ovat nopeampia lajitellusta sajasta. ZRANGEBYLEX-komennon avulla löydetyt tiedot ovat tietotyypiltään merkkijonoja, jotka sisältävät tässä tapauksessa dokumenttien tiedot toisistaan eroteltuna välimerkillä. Jos haettu tieto on merkkijono, sitä on haastava muokata oikeaan muotoon käyttöliittymälle. Ongelman ratkaisemiseksi luotiin PHP-luokka, joka luo objekteja merkkijonoista, jolloin niiden tietoja voidaan viedä käyttöliittymälle yhtenäisessä muodossa.

Viimeinen sovellukselle annettu vaatimus oli määritellä GraphQL:än skeematiedostoa käyttäen tyypit haettaville tiedoille hakuindeksistä. GraphQL:än tyyppien määrittelyn jälkeen luotiin erilliset kyselyt eri tyypeille ja määriteltiin sekä luotiin hakuehtotyyppi. Kyselyjen sekä tyyppien määrittelyn jälkeen luotiin tyypeille omat GraphQL-resolverinsa (Resolver), joille välitettiin määritetty hakuehto. GraphQL-resolverin tulee hakea tietoja automaattisen täydennyksen hakuindeksistä ja palauttaa löydetyt tiedot perustuen kyselyssä välitettyyn hakuehtoon. Resolverit käyttivät luotuja luokkia, joiden avulla tiedon hakeminen hakuindeksistä oli yksinkertaista ja nopeaa tietojen palautuessa valmiiksi halutussa muodossa luokkien metodeja käyttäen.

Automaattisen täydennyksen hakuindeksin luonnissa oli tärkeässä osassa myös indeksoitavien tietojen muistinkäytön optimointi. Optimointina suoritettiin indeksoitavien tietojen avaimien konvertoiminen mahdollisimman lyhyiksi merkkijonoiksi. Avaimien lyhentämisen vuoksi jouduttiin tekemään tietojen haun jälkeen linkitys alkuperäisiin avaimiin käyttäen luokkien linkitysfunktioita. Hakuindeksin luonnissa otettiin huomioon, että indeksoitavien tietojen tulisi olla mahdollisimman harvoin muuttuvia tietoja, esimerkiksi tehtävän yksilöllinen tunniste. (Orfin, Antoni)

7.2 Front-end

Käyttöliittymän eli front-endin rakentaminen aloitettiin käyttöliittymän rautalankamalleja (wireframe) tutkimalla. Rautalankamallien tutkimisen jälkeen käyttöliittymän ulkoasun työstäminen alkoi käyttäen SCSS-tyylitiedostoja. Reactin avulla käyttöliittymän kehitys oli nopeaa, koska tiedostojen tallentamisen jälkeen React tunnistaa muutokset tiedoissa ja päivittää automaattisesti selaimessa auki olevan käyttöliittymän. Käyttöliittymän kehityksessä on käytössä Material-Ui:n tarjoamia valmiita käyttöliittymäkomponentteja, joiden tyylittelyt muokattiin rautalankamallin mukaisiksi. Käyttöliittymän tulee

olla responsiivinen ja helppokäyttöinen myös mobiililaitteilla, joten kehittäminen sekä luonnostelu aloitettiin pienimmästä näytön koosta kohti suurempia näyttökokoja.

Tyylittelyjen jälkeen kirjoitettiin GraphQL kyselyt, joita käyttämällä haetaan tietoja hakuindeksistä. Kyselyitä tuli kolme, jokaiselle tiedolle omansa. Koska GraphQL mahdollistaa erillisten kyselyjen yhdistämisen yhdeksi isommaksi kyselyksi, yhdistin aikasemmin luomani kyselyt yhdeksi isommaksi kyselyksi. Kyselyjen yhdistäminen nopeutti front-endin ja back-endin välistä yhteyttä ja selkeytti front-endin kehitystä, koska kaikki hakuindeksistä löytyvät tiedot palautuivat yhdessä vastauksessa. Tietojen hakemisen jälkeen määrittelin jokaiselle haetulle tiedolle rajapintaluokat käyttäen TypeScriptiä. Rajapintaluokkien avulla sain tehtyä vastauksessa palautuville tiedoille omat muuttujansa, mitkä ovat vahvasti tyypitettyjä. Tehtyjen rajapintaluokkien avulla käyttöliittymän ylläpitäminen sekä kehittäminen on nopeampaa, koska kehittäjän ei tarvitse katsoa rajapintadokumentaatiota selvittääkseen palautuvien tietojen tyyppejä. Rajapintaluokkien luonnin jälkeen lisäsin GraphQL kyselyssä tulleiden kielikäännösten ja näytettävien hakutulosten välille linkityksen toisiinsa, jonka jälkeen tiedot olivat valmiina näytettäväksi käyttöliittymällä asiakkaan omalle kielelle käännettynä.

Material-Ui:n käyttöliittymäkomponenteista valittiin käytettäväksi AutoComplete-komponentti. AutoComplete-komponentti renderöi listan sille välitetyistä tiedoista. Listan renderöitävät tiedot haettiin käyttäen aikasemmin luotua GraphQL-kyselyä. Listalle haettiin tiedot hakuindeksistä asiakkaan kirjoittaessa vähintään kolmen merkin mittainen hakulause syöttökenttään. Asiakkaan kirjoittaessa tehtiin aina uusi GraphQL-kysely hakuindeksiin uusien hakutulosten löytämiseksi. Back-endin ja front-endin välisen tiedonsiirron määrän vähentämiseksi luotiin React-koukku, joka lähettää GraphQL kyselyn vasta puolen sekunnin päästä siitä, kun asiakas on lopettanut hakulauseensa kirjoittamisen. Hakutuloksia näyttävän listan responsiivisuus sekä tyylittelyt vaativat AutoComplete-komponentin tyylien ylikirjoittamista. AutoComplete-komponentin listassa olevat tiedot korostettiin perustuen kirjoitettuun hakulauseeseen korostus funktion avulla. Hakukomponentin tulee näyttää tiedot lähes reaaliaikaisesti, jotta käytettävyys sekä käyttökokemus olisivat hakua käytettäessä ensiluokkaiset.

8 JATKOKEHITYS

Back-endin jatkokehitys jatkuu uuden hakuindeksin luontiin käytettävän tietovaraston tutkimisella sellaisessa tapauksessa, kun luodun hakuindeksin tietomäärä on massiivinen. Hakuindeksin tietojen määrä tulee kasvamaan, eikä tietoja voida enää säilöä RAM-muistiin kustannussyistä. Vaihtoehtoja Rediksen korvaajaksi löytyy, mutta useissa tapauksissa joudutaan tekemään kompromissejä hakujen nopeuden tai hakuindeksin luonnin keston kanssa. Hakuindeksin luontiin ja hakuun käytetyt luokat tukevat sovitinsuunnittelumallia mahdollistaen tietovaraston vaihtamisen uuteen vaivattomasti.

Ennen uuden tietovaraston käyttöönottoa tulee laskea datan vievän tilan lisäksi myös indeksien viemä tila. Dokumenttipohjainen tietokanta sopisi käytettäväksi hakuindeksin kanssa, sillä haetut tiedot muokataan dokumentteihin jo hakuindeksia luotaessa. Front-endin osalta käyttöliittymä vastaa rautalankamalleja, joten näin ollen tyylittelyihin liittyen ei ole jatkokehitystarpeita. Mahdollisena lisäkehityksenä voisi olla siirtymä pikahausta asiakkaan valitseman tiedon perusteella uuteen dialogiin, esimerkiksi hallintasivulle.

9 POHDINTA

Opinnäytetyön tarkoituksena oli luoda asiakkaan käyttöliittymän kojelautaan pikahakukomponentti, joka on käytettävyydeltään yksinkertainen ja suorituskyvyltään erinomainen. Pikahaulle asetettuihin vaatimuksiin vastaamisen kannalta hakuindeksin luonti oli olennainen osa. Indeksiin kohdistuvat haut testattiinkin huolellisesti manuaali- sekä automaattitesteillä. Tavoitteessa onnistuttiin hyvin, lopputulos on laadultaan erinomainen ja toiminnallisuudeltaan täysin käyttöön sopiva. Työn tilaajan puolen ohjaaja antoi työstä kehuja. Koodi oli hänen mukaansa laadukasta, monikäyttöistä sekä OOP-mallia oli hyödynnetty taidokkaasti. Työssä käytettävät tekniikat ovat moderneja ja käyttökelpoisia yritysmaailmassa.

Työn toteutusta ja etenemistä uhkasi tiukka aikataulu kokoaikaisen työn ohessa. Tarkan suunnitelman ja huolellisen riskienhallinnan avulla aikataulu oli mahdollinen, päämäärään päästiin ilman isompia ongelmia.

Opinnäytetyö tehtiin yritykselle, jossa olen ollut töissä lähes kahden vuoden ajan. Työskentely Rediksen parissa loi työyhteisölle huomattavaa tietotaitoa tarkan ja seikkaperäisen dokumentoinnin ansiosta. Välimuistituksella säästetään arvokasta aikaa lyhentyneiden latausaikojen avulla. Operatiivisen relaatiotietokannan kuormaa vähennettin myös hakujen kohdistuessa Rediksen välimuistiin relaatiokannan sijasta.

LÄHTEET

2Kmediat. [Viitattu 2020-03-20] Saatavissa: https://www.2kmediat.com/php/johdanto.asp

AddSearch. 2020. What is a search index and how does it work? [Viitattu 2020-03-29] Saatavissa: https://www.addsearch.com/blog/search-index/

Amazon. 2012. Redis - Fast, open source in-memory data store for use as a database, cache, message broker, and queue. [Viitattu 2019-12-17]. Saatavissa: https://aws.amazon.com/redis/

AppWorks, S. F. Redis vs MongoDB: Which One Should You Choose? [Viitattu 2020-03-23] Saatavissa: https://www.sfappworks.com/blogs/redis-vs-mongodb-which-one-should-you-choose/

Arden Dertat. How to Implement a Search Engine Part 1: Create Index. [Viitattu 2020-04-22] Saatavissa: http://www.ardendertat.com/2011/05/30/how-to-implement-a-search-engine-part-1-create-index/

Arguments. GraphQL. [Viitattu 2020-04-18] Saatavissa: https://graphql.org/learn/queries/#arguments

Bhavikp19. 2011-10-10. Linked representation of Deque [digikuva] Sijainti: https://upload.wikimedia.org/wikipedia/commons/d/df/Link_list.png

Building Prefixy. [Viitattu 2020-03-20] Saatavissa: https://prefixy.github.io/#data-structures--algorithms

Byron, Lee. 2015. GraphQL: A data query language. [Viitattu 2020-04-18] Saatavissa: https://engineering.fb.com/core-data/graphql-a-data-query-language/

Code Institute. What is PHP Programming? [Viitattu 2020-03-24] Saatavissa: https://codeinstitute.net/blog/what-is-php-programming/

Facebook. 2020. React – A JavaScript library for building user interfaces. [Viitattu 2020-04-17] Saatavissa: https://reactjs.org/

Giroux, Marc-André. 2019. Where we Come From: An Honest Introduction to GraphQL. [Viitattu 2020-04-18] Saatavissa: https://medium.com/@__xuorig__/where-we-come-from-an-honest-introduction-to-graphql-4a2ef6124488

Hámori, Ferenc. 2018. The History of React.js on a Timeline. [Viitattu 2020-04-17] Saatavissa: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

Jain, Saurav. 2019. Inverted Index. [Viitattu 2020-03-20] Saatavissa: https://www.geeksforgeeks.org/inverted-index/

Jöch, David. 2018. Functional vs Class-Components in React. [Viitattu 2020-04-17] Saatavissa: https://medium.com/@Zwenza/functional-vs-class-components-in-react-231e3fbd7108

Kentico 8 Documentation. Creating search indexes. [Viitattu 2020-04-22] Saatavissa: https://docs.kentico.com/k8/configuring-kentico/setting-up-search-on-your-website/creating-search-indexes

Kupari, Severi. 2020. Hakuindeksin dokumentin rakenne.

Kupari, Severi. 2020. Jäsenen hakeminen lajitellusta sarjasta.

Kupari, Severi. 2020. Jäsenen lisäys lajiteltuun sarjaan.

Kupari, Severi. 2020. Tyyppiluokkien luokkakaavio.

Lofhjelm, Janne. Osa 2 - Versionhallinta: Git ja Github. tkt-lapio. [Viitattu 2020-03-26] Saatavissa: https://tkt-lapio.github.io/git/

Majerczyk, M. 2019. Database Compare — SQL vs. NoSQL (MySQL vs PostgreSQL vs Redis vs MongoDB). [Viitattu 2020-03-29] Saatavissa: https://medium.com/profil-software-blog/database-compare-sql-vs-nosql-mysql-vs-postgresql-vs-redis-vs-mongodb-3da5f41c31b5

Marsden, Sam. 2018. Search Engine Indexing. [Viitattu 2020-04-22] Saatavissa: https://www.deepcrawl.com/knowledge/technical-seo-library/search-engine-indexing/

Meyer, Dave. 2018. 8 steps to a definition of done in Jira. Atlassian. [Viitattu 2020-03-26] Saatavissa: https://www.atlassian.com/blog/jira-software/8-steps-to-a-definition-of-done-in-jira

Multiple fields in mutations. GraphQL. [Viitattu 2020-04-18] Saatavissa: https://graphql.org/learn/queries/#multiple-fields-in-mutations

Orfin, Antoni. How we cut down memory usage by 82%. Octivi. [Viitattu 2020-03-26] Saatavissa: https://web.archive.org/web/20180421151235/http://labs.octivi.com/how-we-cut-down-memory-usage-by-82/

Payravi Kevin. 2014-06-24. Scrum Diagram. [digikuva] Sijainti: https://upload.wikimedia.org/wikipedia/commons/b/b1/Scrum_diagram_%28labelled%29.png

PhpStorm. 2020. TypeScript - Help | PhpStorm. [Viitattu 2020-04-16] Saatavissa: https://www.jetbrains.com/help/phpstorm/typescript-support.html

Queries and Mutations. GraphQL. [Viitattu 2020-04-18] Saatavissa: https://graphql.org/learn/queries/

Redis Labs. 1.1.2 Other Features. [Viitattu 2020-04-22] Saatavissa: https://redislabs.com/ebook/part-1-getting-started/chapter-1-getting-to-know-redis/1-1-what-is-redis/1-1-2-other-features/

Redis Labs. 1.2 What Redis data structures look like. [Viitattu 2020-04-22] Saatavissa: https://redislabs.com/ebook/part-1-getting-started/chapter-1-getting-to-know-redis/1-2-what-redis-data-structures-look-like/

Redis Labs. 1.2.5 Sorted sets in Redis. [Viitattu 2020-04-22] Saatavissa: https://redislabs.com/ebook/part-1-getting-started/chapter-1-getting-to-know-redis/1-2-what-redis-data-structures-look-like/1-2-5-sorted-sets-in-redis

Redis Labs. 6.1.1 Autocomplete for recent contacts. [Viitattu 2019-12-17]. Saatavissa: https://redislabs.com/ebook/part-2-core-concepts/chapter-6-application-components-in-redis/6-1autocomplete/6-1-1-autocomplete-for-recent-contacts/

Redis Labs. 7.1.1 Basic search theory. [Viitattu 2020-01-22] Saatavissa: https://redislabs.com/ebook/part-2-core-concepts/chapter-7-search-based-applications/7-1-searching-in-redis/7-11-basic-search-theory/

Redis Labs. Full Text Search. [Viitattu 2020-04-22] Saatavissa: https://redislabs.com/redis-best-practices/indexing-patterns/full-text-search/

Redis. Data types. [Viitattu 2020-04-22] Saatavissa: https://redis.io/topics/data-types

Redis. EVAL script. [Viitattu 2020-03-12] Saatavissa: https://redis.io/commands/eval

Redis. EXPIRE key seconds. [Viitattu 2020-03-12] Saatavissa: https://redis.io/commands/expire

Redis. Introduction to Redis. [Viitattu 2020-04-22] Saatavissa: https://redis.io/topics/introduction

Redis. Redis Persistence. [Viitattu 2020-03-29] Saatavissa: https://redis.io/topics/persistence

Redis. Secondary indexing with Redis. [Viitattu 2020-04-22] Saatavissa: https://redis.io/topics/indexes

Ropo Capital. [Viitattu 2020-03-12] Saatavissa: https://www.ropocapital.fi/fi/yritys/

Ropo Capital. Arvot ja toimintatapamme. [Viitattu 2020-03-12] Saatavissa: https://www.ropocapital.fi/fi/yritys/arvot-visio-strategia/#miksi

Swiftype. Search Concept: Search index. [Viitattu 2020-04-22] Saatavissa: https://swiftype.com/search-concepts/search-index

Tarvainen, Jani. 2016. Mikä on GraphQL ja miten se eroaa REST-rajapinnoista (API)? [Viitattu 2020-04-18] Saatavissa: https://symfony.fi/artikkeli/mika-on-graphql-ja-miten-se-eroaa-rest-rajapinnoista-api

Tarvainen, Jani. 2016. Mikä on TypeScript. [Viitattu 2020-04-16] Saatavissa: https://symfony.fi/artikkeli/mika-on-typescript

Typescriptlang. TypeScript in 5 minutes. Type annotations. [Viitattu 2020-04-16] Saatavissa: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html#type-annotations

Xarawn 2015-11-4 Lifecycle of the Test-Driven Development method [digikuva] Sijainti: https://commons.wikimedia.org/wiki/File:TDD_Global_Lifecycle.png