Opinnäytetyö (AMK)

Tietojenkäsittely

2024

Elias Westerling

# Valmistuneiden projektien arkistointilogiikan suunnittelu ja toteutus

## algoritmien vertailu ja haasteiden ratkaisu

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tietojenkäsittely ja viestintä

2024 | 42

Elias Westerling

# Valmistuneiden projektien arkistointilogiikan suunnittelu ja toteutus

- algoritmien vertailu ja haasteiden ratkaisu

Digitalisoituneessa yhteiskunnassa toiminnanohjausjärjestelmän rooli kasvaa organisaatioissa. Erityisesti tiedonhallinnan ja arkistoinnin tarpeet korostuvat, kun tietoa tuotetaan ja käsitellään yhä enemmän. Opinnäytetyön tavoitteena oli löytää ratkaisu TPE Turun Pelti ja Eristys Oy:n arkistointitarpeisiin toiminnanohjausjärjestelmään. Arkisto on toimeksiantajalle välttämätön, sillä valmistuneiden projektien, positioiden ja työnumeroiden tietoja on säilytettävä pitkään niiden valmistumisen jälkeen. Arkistoitujen tietojen on oltava nopeasti ja vaivattomasti saatavilla. Tavoitteena oli luoda suorituskykyinen ja helposti ylläpidettävä arkistointilogiikka, joka kykenisi suodattamaan projekteja ja projektien pienempiä osia käyttäjän syöttämän hakusanan perusteella. Ratkaisun tulisi olla kestävä, eli tiedonkäsittelyn tulisi onnistua myös suurissa datamäärissä.

Opinnäytetyön tutkimusmenetelminä käytettiin tiedonhaussa kirjallisuuskatsausta ja kehittämismenetelmistä pragmatismia ja konstruktiivista lähestymistapaa. Kehittämismenetelmät ilmenivät toiminnallisena testaamisena ja soveltavana raportointina. Teoreettisissa testeissä käsiteltiin kolmea hakualgoritmia tiedon hakemiseksi tietokannasta ja kolmea suodatuslogiikkaa, jotka perustuivat käyttäjän syöttämään hakusanaan. Teoreettisia testejä suoritettiin pienellä datamäärällä, minkä jälkeen yhdistettiin parhaaksi todettu haku- ja suodatinalgoritmi. Yhdistettyä algoritmia kokeiltiin toiminnallisissa testeissä. Toiminnallisen testaamisen tarkoituksena oli kokeilla logiikan kestävyyttä generoimalla tietokantaan suuri määrä tietoa. Testien perusteella tutkitut logiikat toimivat hyvin pienissä tietokannan tietomäärissä. Sen sijaan tutkittuja toimintamalleja ei voitu hyödyntää arkistointilogiikan toteutuksessa, koska suurilla tietomäärillä Node.js:n palvelinympäristön oletusmuistiraja ylittyi ja johti ohjelmistovirheeseen.

Asiasanat: algoritmi, Prisma, toiminnanohjausjärjestelmä, arkisto, ORM-järjestelmät, arkistointilogiikka

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and communication technology

2024 | 42

Elias Westerling

# Design and Implementation of Archiving Logic

– system comparison and solution to challenges

The aim of this thesis was to find a solution for the archiving needs of TPE Turun Pelti ja Eristys Oy in their Enterprise Resource Planning (ERP) system. Archiving is essential for the company, as information about completed projects, positions, and tasks must be retained for an extended period after completion. The archived data must be readily and easily accessible. The objective was to create a performant and easily maintainable archiving logic capable of filtering projects and their subcomponents based on a search term given by the user. The solution should be durable, meaning that data processing should be possible even with large data volumes.

This thesis used a literature review for information gathering, pragmatism for practical application, and a constructive approach for development. Methods included functional testing to evaluate performance and applied reporting to document the design process and results. The theoretical tests focused on three search algorithms for retrieving data from the database and three filtering logics based on search terms given by the user. Theoretical tests were conducted using small datasets, after which the best-performing search and filter algorithms were combined. The combined algorithm was then tested in functional tests. The purpose of the functional testing was to assess the robustness of the logic by generating large amounts of data in the database. Based on the tests, the researched logic worked well with small datasets. However, the tested models could not be used for the implementation of the archiving logic, as large data volumes exceeded the default memory limit of the Node.js server environment.

Keywords: algorithms, Prisma, Enterprise Resource Planning, archive, ORM-systems, archive logic

# Sisällys_Toc184650749

# Taulukot

# Kuvat

# Käytetyt lyhenteet ja sanasto

Algoritmi:

Säännöstö, jonka avulla voidaan suorittaa laskutoimitus tai muu toiminto. Käytetään yleisesti tietojenkäsittelyssä ja matematiikassa. (Suomisanakirja.fi, 2024)

Arkistointi:

Arkistointi viittaa olennaisen tiedon tallentamiseen ja sen pitkäaikaiseen säilyttämiseen ja tiedon säilymiseen muuttumattomana. Arkistointiin kuuluu myös arkistoidun tiedon saatavuuden, käytettävyyden ja turvallisuuden ylläpitäminen. (Kansallisarkisto)

Arkistointilogiikka:

Järjestelmä tai menetelmä, jonka avulla arkistoitu data organisoidaan, säilytetään, haetaan ja hallitaan tehokkaasti.

CLI:

Command Line Interface, komentokehote. Tapa järjestää ihmisen ja tietokoneen välinen kommunikointi. (Hassan, 2023)

Digitalisaatio:

Digitalisaatio viittaa merkittävään yhteiskunnalliseen muutokseen, jossa perinteiset fyysiset toiminnot korvataan digitaalisilla teknologioilla. (Opetushallitus)

Elinkaarimalli:

Ohjelmistokehityksessä ja talousalalla käytettävä malli, jota käytetään kuvaamaan tuotteen tai järjestelmän vaiheita sen kehityksessä ja käytössä. (Kolehmainen, 2005)

ERP:

Enterprise Resource Planning, ohjelmistojärjestelmä joka auttaa organisaatioita virtaviivaistamaan liiketoimintaprosessejaan. (SAP)

EU:

Euroopan Unioni

GDPR:

General Data Protection Regulation, yleinen tietosuojalaki

ID:

Tietojenkäsittelyssä annettava yksittäinen tunniste. (Schott, 2023)

ORM:

Objective Relational Mapping, ORM-järjestelmä.

ORM-järjestelmä:

Ohjelmistotekniikka, joka helpottaa olio-ohjelmointikielien ja relaatiotietokantojen välistä vuorovaikutusta. Tekniikan tarkoituksena on helpottaa kehittämistyötä vuorovaikutuksen parantamisen avulla. (Abba, 2022)

Positio:

Projektille kuuluva pienempi kokonaisuus. Projektilla voi olla useampi positio ja positioilla voi olla vielä pienempiä töitä, eli työnumeroita.

Prisma:

Prisma on avoimen lähdekoodin ORM-järjestelmä. (Prisma Data, Inc., 2024)

Sekventiaalinen:

Tietyssä järjestyksessä tapahtuva suorittaminen. (Suomisanakirja.fi, 2024)

SQL:

Structured Query Language, rakenteinen kyselykieli jota käytetään tietojen käsittelemiseen ja hakemiseen tietokannasta. (2Kmediat, 2024)

Tietoturva:

Tietoturvalla tarkoitetaan tiedon saatavuuden, eheyden ja luottamuksellisuuden turvaamista. (Jyväskylän yliopisto)

Toiminnonohjausjärjestelmä:

Toiminnanohjausjärjestelmä on ohjelmistojärjestelmä, joka auttaa organisaatioita virtaviivaistamaan keskeisiä liiketoimintaprosessejaan. (Fikuro, 2024)

TPE:

Turun Pelti ja Eristys Oy

# 1 Johdanto

Rakennusalan yrityksessä on tärkeää ylläpitää tietoja valmistuneista projekteista, positioista ja työnumeroista. Arkistoinnin tarkoituksena on toimia luotettavana tietovarastona, joka mahdollistaa historiatietojen tallentamisen ja tarvittaessa niiden tehokkaan hakemisen. Tämä ei pelkästään helpota organisaation sisäistä toimintaa ja päätöksentekoa, vaan myös parantaa projektien ja sille kuuluvien positioiden ja työnumeroiden jälkiseurantaa sekä tulevien vastaavien hankkeiden suunnittelun laatua. Järjestelmässä käsitellään suuria määriä projekteja sekä niille kuuluvia positioita ja työnumeroita. Niiden arkistointi parantaa tuotannon näkyvyyttä ja pitää sen siistinä ja ajankohtaisena. Tämä puolestaan kehittää huomattavasti käyttäjäkokemusta ja mahdollistaa arkistoitujen ja arkistoimattomien töiden erottamisen omiin sektioihinsa.

Projektien, positioiden ja työnumeroiden määrä voi kasvaa huomattavaksi, joten on tärkeää miettiä haun suorituksen kestoa ja minimoimaan prosessorin työkuormaa. On harkittava, montako hakua suoritetaan ja millä suodattimilla minkäkin haun yhteydessä.

Opinnäytetyön tavoitteena on löytää mahdollisimman hyvä ratkaisu TPE Turun Pelti ja Eristys Oy:n (myöhemmin TPE) toiminnanohjausjärjestelmän valmistuneiden projektien arkistointilogiikalle. Tavoitteen saavuttamiseksi arkistointilogiikka suunnitellaan huolellisesti ja toteutetaan käyttäen parhaiksi todettuja menetelmiä. Ennen suunnittelua suoritetaan tiedonhaku, joka edesauttaa suunnittelun ja varsinaisen logiikan toteuttamista. Aiheen valinta perustuu sen ajankohtaisuuteen toiminnanohjausjärjestelmän kehittämisen näkökulmasta ja sen kriittiseen merkitykseen yrityksen toiminnan kannalta. Digitaalinen aikakausi tarjoaa merkittäviä mahdollisuuksia organisaatioille, jotka omaksuvat ja hyödyntävät teknologisia innovaatioita liiketoimintaprosesseissaan. Tämä kehitys tuo toimeksiantajalle huomattavan kilpailuedun verrattuna muihin kilpaileviin yrityksiin. Digitalisaatio mahdollistaa tehokkaammat toimintatavat, paremman asiakaskokemuksen sekä skaalautuvuutta liiketoiminnan kasvuun (Upwork, 2024).

Tässä opinnäytetyössä käytettäviä tutkimusmenetelmiä ovat tiedonhaussa kirjallisuuskatsaus ja kehittämismenetelmät, joista toteutuvat käytännönläheisinä ja kehittämiseen soveltuvina pragmatistinen ja konstruktivistinen lähestymistapa (Salonen, 2017). Kehittämistyön toteutuksessa käytetään toiminnallista ohjelmointia ja testaamista.

Työssä tarkastellaan Prisma ORM -järjestelmän tarjoamia mahdollisuuksia ja ominaisuuksia sekä analysoidaan sen suorituskykyä ja käytännöllisyyttä arkistointilogiikan toteuttamista varten. Tutkimuksen tavoitteena on selvittää, millä menetelmillä saadaan toteutettua parhaiten toimiva arkistointilogiikka edellä mainittuihin tarpeisiin.

Opinnäytetyön tavoitteena on tutkia vaihtoehtoja arkistointilogiikan toteuttamiseksi sekä verrata niiden käytännöllisyyttä ja tehokkuutta muihin vastaaviin algoritmeihin. Tutkimusten ja arkistointilogiikan suunnittelun jälkeen pyritään toteuttamaan valittu ratkaisu osaksi toiminnanohjausjärjestelmän lähdekoodia.

Tarkoituksena on löytää käyttötarkoitukseen mahdollisimman tehokas algoritmi ja suunnitella hakukyselyt siten, että datan hakeminen pysyy mahdollisimman kevyenä. Tällöin ohjelma säilyttää käytännöllisyyden käyttäjän näkökulmasta ja on kykenevä hakemaan suurienkin datamäärien joukosta tehokkaasti tietoa. Tämä edellyttää muun muassa tietokantahakujen hajautusta: yhden laajan haun sijaan toteutetaan useita pienempiä hakuja kerrallaan.

Opinnäytetyö käsittelee tiedonhaun arkistointilogiikkaa, algoritmeja, suorituskykyä, yhteensopivuutta sekä käytettävyyttä. Tietoa haetaan yleisesti ottaen arkistosta, arkistointilogiikan teoriasta, ORM-järjestelmistä sekä Prismasta.

Yleisen tiedonhaun ja tavoitteiden määrittämisen jälkeen alkaa toteutus, jossa suunnitellaan ja toteutetaan käytettävä arkistointilogiikka. Arkistointilogiikka on myös testattava yksityiskohtaisesti toiminnallisuuden kannalta. Testauksessa esiintyvät haasteet ja ongelmakohdat dokumentoidaan.

Lopuksi käsitellään testien ja analyysien tuloksia sekä käsitellään valmiin arkistointilogiikan toimivuutta ja sen hyödyllisyyttä käyttäjän näkökulmasta. Tämän jälkeen pohditaan opinnäytetyön keskeisimpiä seikkoja ja lisäksi käydään läpi tutkimuskysymysten vastaukset ja ideat jatkokehitykseen.

# 2 Teoreettinen tausta ja kirjallisuuskatsaus

Tässä luvussa keskitytään tiedonhakuun yleisesti arkistosta, arkistointilogiikoista ja niihin liittyvistä huomioitavista seikoista, ORM-järjestelmistä ja niiden eduista ja heikkouksista suoriin SQL-kyselyihin verrattuna sekä Prisma ORM-järjestelmästä ja sen tarjoamista työkaluista ja ominaisuuksista.

## 2.1 Arkistointi

Arkistointi on hyvin keskeinen osa organisaatioiden tiedonhallintaprosesseja. Arkistoinnin pääasiallinen tarkoitus on varastoida tietoa, jolle ei nykyhetkenä ole tarvetta, mutta joka voi olla tarpeellista tulevaisuudessa. Arkistoinnin avulla organisaatiot voivat hallita dataa tehokkaasti ja varmistaa sen nopean saatavuuden, sekä optimoida aktiivisesti käytössä olevan tiedon määrää. Arkistoinnin suurimpia etuja ovat mm. aiemmin tallennetun tiedon nopea ja tehokas saatavuus sekä käyttäjäkokemuksen parantaminen. (Patrina Corporation, 2023.)

Arkisto voi sisältää monenlaista tietoa, kuten yrityksen sisäisiä töitä, henkilötietoja, rahoitukseen liittyviä asiakirjoja, dokumentteja ja muita tietoja, joita voidaan tarvita myöhemmin (Data Science Society, 2023). Arkiston avulla käyttäjällä on saatavilla vain se informaatio, joka tukee hänen välittömiä tarpeitaan ja toimintaansa. Esimerkiksi meneillään olevan projektin osalta käyttäjä voi tarvita vain tiettyjä asiakirjoja tai raportteja, eikä hänen tarvitse käydä läpi kaikkea saatavilla olevaa tietoa.

Laadukas arkistointi tuo yritykselle mukanaan merkittäviäkin hyötyjä, kuten käyttökustannuksien vähenemisen, liiketoiminnan jatkuvuuden edistämisen, saatavilla olevan tiedon keskityksen ja tehostamisen, laki- ja vaatimusseikkojen tuen, riskinhallinnan edistämisen, vakauden säilyttämisen viestintä -muutosten aikana sekä organisaation luotettavuuden parantamisen. Näiden hyötyjen vuoksi useimmat yritykset pyrkivät investoimaan arkistoihinsa ja niiden toimivuuteen. (Lucidea Technologies Corp.)

Arkistointi mahdollistaa vanhan tiedon siirtämisen ja tallentamisen järjestelmällisesti. Arkiston etuna on sen kyky palauttaa aiemmin tallennettua tietoa nopeasti, mikä voi olla tärkeää esimerkiksi historiatiedon tai vanhojen tapahtumien tarkastelussa. Tämä tekee arkistoinnista tärkeän osan organisaation tietohallintoa. (Iron Mountain, 2024.)

Arkistointi tukee tiedonhallintaa monella tavalla. Ensinnäkin arkistointi edesauttaa optimoimaan aktiivisessa käytössä olevan tiedon määrää, mikä selkeyttää ohjelmaa ja parantaa käyttäjäkokemusta. Käyttäjän työ tehostuu, kun hän voi keskittyä vain tarvitsemaansa tietoon (Porkka, 2023). Tämän myötä keskittyminen olennaiseen asiaan helpottuu huomattavasti. Tiedon säilyttäminen järjestelmällisesti arkistossa

vähentää myös tietotulvaa, joka voi johtaa käyttäjän puolelta virheellisiin päätöksiin tai aikarajoitteisiin. (Iron Mountain, 2024.)

Toiseksi arkistointi mahdollistaa tietojen ja dokumenttien luokittelun ja kategorisoinnin, mikä tekee tiedon löytämisestä ja hallinnasta entistä helpompaa. Hyvin organisoitu arkisto voi sisältää eri kategorioita, suodatintoimintoja ja tunnisteita, joiden avulla käyttäjät voivat hakea ja löytää tarvitsemansa tiedon nopeasti. Tämä vähentää tiedon etsimiseen kuluvaa aikaa ja samalla parantaa organisaation tehokkuutta. (Iron Mountain, 2024.)

Arkistointilogiikka on keskeinen osa tietohallintoa. Arkistointilogiikka pitää sisällään monia näkökohtia, kuten mm. tietojen tallennuskäytännöt, tiedon hallinnoinnin, tiedon suojaamisen, tietoturvan, tiedon eheyden ja pysyvyyden, sekä elinkaaren hallinnan ja tiedon hävittämisen. Tehokas arkistointilogiikka mahdollistaa organisaatioille suurienkin tietomäärien hallinnan samalla varmistaen, että tiedot pysyvät saatavilla ja muuttumattomina prosessin aikana. Näin voidaan varmistua siitä, että arkistoitu tieto säilyy suojattuna ja eheänä. (Torro, 2022.)

Arkistointilogiikan pääsääntöinen tehtävä on määrittää, miten ja minne tieto tallennetaan sekä miten tietoon päästään helposti käsiksi. Tiedon tallennuksessa käytetään usein numeerisia tunnisteita, kuten ID-tunnisteita, jotka helpottavat tiedon hallintaa ja käsittelyä (Schott, 2023). Tiedon käsittelyyn käytettävät ID-tunnisteet ovat uniikkeja, ja niitä ei voida suoraan liittää arkaluontoiseen tietoon, mikä puolestaan parantaa tietoturvaa. Tietokannat on suunniteltu optimoimaan suoritusta numeeristen hakujen avulla, mikä myös tukee ID: n käyttöä sen tehokkuuden takia. Tällä tavoin organisaatiot voivat skaalata tietojärjestelmiään helpommin ja ylläpitää niitä pitkäaikaisesti. (TEK-Tools, 2023.)

Yleisiä arkistointilogiikan vertailukriteereitä ovat mm. suorituskyky, skaalautuvuus, tarkkuus, monimutkaisuus ja ylläpidettävyys (AltexSoft, 2023). Näiden lisäksi tietorakenneyhteensopivuus ja soveltuvuus tiettyihin käyttötarkoituksiin ovat merkittäviä tekijöitä. Näitä ovat esimerkiksi suuren datamäärän käsittely ja reaaliaikainen haku. (Algorithm Examples)

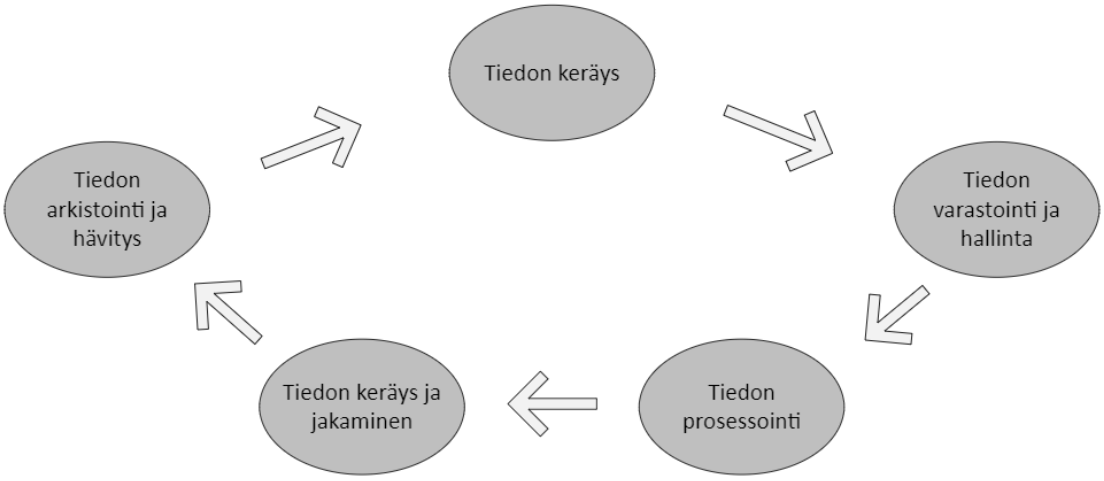
## 2.2 Tietoturva ja tiedon elinkaarimalli

Arkistointilogiikan kannalta tietoturva on yksi tärkeimmistä osa-alueista. Arkistoitavien tietojen suojaaminen on välttämätöntä, jotta organisaatiot voivat välttää tietovuotoja ja muita turvallisuusriskejä (Patrina Corporation, 2023). Suomessa noudatetaan tiukasti tietosuojalakia, joka perustuu EU:n GDPR:ään (Tietosuojalaki 5.2.2018/1050). GDPR:n mukaan arkistoitavat henkilötiedot on minimoitava vain välttämättömään ja tiedon säilyttämiseen liittyvät käytännöt on määriteltävä tarkasti. Tiedon minimointiperiaate tarkoittaa, että vain myöhemmin mahdollisesti tarvittavat tiedot tulisi arkistoida ja niiden säilyttämisessä tulee

huomioida, kuinka kauan tietoa tarvitaan, varsinkin jos arkistoidut tiedot pitävät sisällään henkilötietoja. Yleisen tietosuojalain mukaan henkilöllä on oikeus vaatia omien tietojensa poistoa arkistosta. Tällöin asianomaisen tiedot tulee poistaa, ellei tietojen säilyttämiselle ole oikeutettua perustetta.

Arkistointilogiikan on myös varmistettava tiedon eheys ja pysyvyys. Tämä tarkoittaa, että arkistoitu tieto ei saa muuttua tai hävitä ilman asianmukaista prosessia. Eheyden varmistaminen voidaan saavuttaa käyttämällä versionhallintaa (IT Business Edge, 2011) ja muita tiedon suojaustekniikoita, kuten esimerkiksi ohjelman toiminnallisuudella, jossa tieto tuodaan käyttöliittymään vain lukuoikeutena. Näin tarvittava tieto voidaan tuoda ongelmitta käyttöliittymään tarkasteltavaksi, mutta käyttäjä ei vahingossakaan voi muokata arkistoitua tietoa.

Tiedon elinkaaren hallinta on hyvin keskeinen ja tärkeä osa arkistointilogiikkaa. Tiedon elinkaarimallilla tarkoitetaan prosessia, jossa tieto kulkee eri tiedon elinkaaren vaiheiden läpi (Kuva 1).




Kuva 1. Tiedon elinkaarta havainnollistava kuva (Chia, 2023).

Tiedon elinkaaressa jokainen kohta on tärkeä. Tämä tarkoittaa, että organisaatioiden vastuulla on suunnitella huolellisesti tiedon arkistointi ja hävitys. Arkistoinnin on huolehdittava siitä, että tietoa säilytetään vain niin kauan kuin on tarpeellista ja sen jälkeen kaikki arkistoituun kohteeseen liittyvä tieto tulee poistaa tietokannasta. Hävittämisprosessin on oltava hyvin dokumentoitu ja turvallinen, jotta pystytään varmistamaan, että tietokannasta hävitettävät tiedot eivät pääse vääriin käsiin. Tiedon hävityksen tarve korostuu, kun arkistoiduissa tiedoissa on arkaluontoisia tietoja, kuten henkilötietoja. (Chia, 2023.)

Johtopäätöksenä voidaan todeta, että arkistointilogiikka on monipuolinen ja kompleksinen alue, joka kattaa useita keskeisiä osa-alueita tietohallinnossa. Tiedon hallinnoinnilla ja tallennuskäytännöillä on oma tärkeä merkityksensä toimivan, turvallisen ja tehokkaan arkistointilogiikan luomisessa. Arkistointilogiikan

suunnittelussa ja toteutuksessa on hyvin tärkeää noudattaa lainsäädäntöön (GDPR) liittyviä seikkoja (5.2.2018/1050). Lainsäädäntöön liittyvät seikat tekevät siitä vieläkin tärkeämmän organisaatioille, jotka pyrkivät suojaamaan tietoa ja noudattamaan lainsäädäntöä. Tehokas arkistointilogiikka ei ainoastaan paranna tiedon saatavuutta ja käyttökokemusta, vaan myös suojaa ja hallitsee tietoa, mikä on nykypäivän tietoyhteiskunnassa tärkeää.

## 2.3 ORM-järjestelmät

Object-Relational Mapper (ORM) -järjestelmät ovat tärkeitä työkaluja nykypäivän ohjelmistokehityksessä niiden tuomien etujen ja helpotuksien vuoksi. Ne pystyvät tarjoamaan kehittäjille tehokkaan ja modernin tavan hallita tietoa ohjelmien ja relaatiotietokantojen välillä ilman monimutkaisten SQL-kyselyiden luomista. Tämä mahdollistaa objektiorientoituneen lähestymistavan tiedonkäsittelyyn, mikä helpottaa datan hallintaa sekä ohjelman ja tietokannan välistä vuorovaikutusta. Kehittäjät voivat ORM-järjestelmien avulla keskittyä sovelluksen kehittämiseen sen sijaan, että heidän pitäisi käyttää huomattavasti aikaa tai huolehtia tietokantakyselyistä ja tietokantamuunnoksista. ORM-järjestelmät mahdollistavat tiedonkäsittelyn objektipohjaisesti. Objektipohjainen tiedonkäsittely helpottaa huomattavasti datan hallintaa ja vuorovaikutusta relaatiotietokantojen kanssa. Objektipohjaisuus tekee siitä myös kehittäjille helpommin ymmärrettävää. (Tina, 2020.)

ORM-järjestelmiä käytetään eri järjestelmissä niiden helppokäyttöisyyden vuoksi (Awati, 2023). Näiden järjestelmien avulla kehittäjät pystyvät lukemaan, luomaan, päivittämään sekä poistamaan tietoa ilman syvällistä tietämystä SQL-kyselykielestä (Tawfik, 2024). ORM-järjestelmiä käyttäviä tunnettuja yrityksiä ovat esimerkiksi GitLab (GitLab, 2024) ja Airbnb (LinkedIn, 2024).

Esimerkkejä suosituista ORM-järjestelmistä ovat Prisma, Drizzle ja TypeORM. Nämä työkalut tarjoavat kehittäjille laajan valikoiman ominaisuuksia, jotka tekevät datan hallinnasta ja sen yhdistämistä sovelluksen logiikkaan helpompaa (Twena-Stern, 2024). Ominaisuudet mahdollistavat myös automaattiset tietomuutokset molempiin suuntiin, mikä vähentää manuaalista työtä ja parantaa kehityksen tehokkuutta (Takeo, 2019).

ORM-järjestelmien käyttöön liittyy monia etuja. Nämä edut tekevät niistä erinomaisen vaihtoehdon ohjelmistokehityksessä. Ensinnäkin kehityksen nopeus kasvaa huomattavasti, sillä kehittäjät voivat keskittyä logiikan toteuttamiseen ilman tarvetta luoda monimutkaisia SQL-kyselyitä. Tämä parantaa myös koodin luettavuutta, koska ORM-kirjastot tarjoavat usein selkeämmän ja helpommin ymmärrettävän syntaksin verrattuna suoriin SQL-kyselyihin. (Adequate, 2023.)

Toiseksi ORM-järjestelmät vähentävät virheiden esiintymistä. Virheitä esiintyy vähemmän, koska kehittäjiltä ei vaadita suoran SQL-koodia luomista. Tämän vuoksi

käyttäjävirheiden riski pienenee huomattavasti. ORM-järjestelmät huolehtivat automaattisesti datan muunnoksista sekä tyypityksestä ja varmistavat, että tiedot pysyvät synkronoituna tietokannan ja sovelluksen välillä. Tämä vähentää myös ylläpidon aikaa ja vaivannäköä, koska ohjelmakoodin ylläpitäminen on yleisesti ottaen helpompaa ja virhetilanteita esiintyy vähemmän. (Adequate, 2023.)

Vaikka ORM-järjestelmät tarjoavat monia etuja, niillä on myös joitakin heikkouksia, jotka kehittäjien on otettava huomioon. Suorituskyky on yksi suurimmista huolenaiheista, koska ORM-järjestelmät voivat olla hitaampia kuin suorat SQL-kyselyt erityisesti suurten tietomäärien käsittelyssä. ORM-tasolla tapahtuvat abstraktiot voivat aiheuttaa ylimääräisiä viiveitä. ORM-järjestelmissä esiintyviä virheitä on vaikeampi havaita. Kehittäjien on tärkeää harkita tarkasti, milloin ORM on oikea valinta ja milloin suora SQL-kysely olisi parempi ratkaisu. Kehittäjien on myös tärkeä ottaa huomioon, että ORM-järjestelmät poikkeavat toisistaan, joten niiden käytössä voi esiintyä eroavaisuuksia. Kehittäjät törmäävät oppimiskäyrään ORM-järjestelmien käyttöönoton yhteydessä. (Pellegrini, 2023.)

ORM-järjestelmät voivat myös rajoittaa kehittäjien mahdollisuuksia toteuttaa monimutkaisia kyselyitä, joita olisi mahdollista suorittaa suoran SQL-kyselyn avulla. (Adequate, 2023.)

Prisma ORM on moderni objektirelaatiomäpperi (ORM), joka on suunniteltu käytettäväksi erityisesti TypeScript- ja JavaScript-sovelluksiin (Mysliwiec). Esimerkkinä suomalaisesta yrityksestä, joka käyttää Prismaa järjestelmissään on S-Pankki (TheirStack, 2024). ORM-järjestelmät pystyvät mahdollistamaan kehittäjien työskentelyn korkeamman tason tietomallien avulla, muuntamalla tietokantakyselyt automaattisesti ohjelmakoodin ymmärrettäväksi rakenteeksi. Prisma on saavuttanut laajaa suosiota viime vuosina erityisesti siksi, että ohjelmistoarkkitehtuurit ovat kehittyneet monimutkaisemmiksi ja mikroarkkitehtuurit (esim. mikropalvelut) ovat yleistyneet. (Minotti, 2023.)

ORM-järjestelmissä suorituskyvyn rooli on huomattava. Prisma ORM on optimoitu modernien tietokantajärjestelmien hyödyntämiseksi, kuten PostgreSQL, MySQL ja SQLite (Amplication, 2024). Prisman suorituskyky heijastuu erinomaisesti etenkin hyvin ja huolellisesti suunnitelluissa tietokantaskeemoissa. Prisma suoriutuu hyvin erityisesti kyselyjen automatisoinnissa, sillä se kykenee generoimaan tehokkaita SQL-kyselyitä. Tämän ansiosta kehittäjät voivat luoda monimutkaisia tietokantakyselyitä ilman tarvetta optimoida SQL-koodia manuaalisesti ja ilman laajaa osaamista SQL-kyselyistä entuudestaan. Prisma tarjoaa myös automaattisen täydennyksen, joka helpottaa ohjelmakoodin tuottamista (Ahmed, 2024).

Prisman käyttämä "query engine" -moottori käsittelee kyselyt tehokkaasti, vaikka tietokannassa olisi suuriakin tietomääriä. Prisman hakujen käyttämä moottori toimii niin, että ne paljastavat matalan tason rajapinnan, jota ylemmän tason rajapinnat hyödyntävät. Moottorit on toteutettu Rustia käyttäen (Prisma Data, Inc., 2024). Kyselyiden optimoidun luonteen ansiosta Prisma pystyy saavuttamaan kohtalaisen

suoritustehokkuuden verrattuna perinteisiin ORM-järjestelmiin, kuten Sequelizeen ja TypeORM:iin. (Ahmed, 2024.)

Prisman suorituskykyä kuitenkin rajoittaa se, että jokainen kysely luo uuden yhteyden tietokantaan. Tämä voi suurilla tietomäärillä hidastaa toimintaa, kun suorituskykyä verrataan täysin manuaalisiin kyselyihin. Pienemmissä kyselyissä edellä mainittu rajoitus tekee Prismasta tehokkaamman, mutta raskaammissa tietojenkäsittelyissä suorituskyky saattaa heikentyä. (Prisma Data, Inc., 2024.)

Prisma ORM:in suurin hyöty on sen modernius, joka tuo kehittäjälle positiivisen kehityskokemuksen sekä selkeä syntaksi, joka mahdollistaa intuitiivisen ja tehokkaan kehitystyön. Prisma hyödyntää dynaamista tietokantamallinnusta, jonka toiminta perustuu suoraan tietokannan rakenteeseen. Tämä ominaisuus helpottaa tietokantamallien hallintaa, sillä Prisma on kykeneväinen synkronoimaan automaattisesti tietomallit tietokantakaavioiden kanssa. Prismaa hyödyntävät kehittäjät voivat myös käyttää Prisman integrointia TypeScriptin kanssa, mikä parantaa tietoturvaa ja vähentää virheiden riskiä ajon aikana. (Prisma Data, Inc., 2024.)

Lisäksi Prisma tarjoaa tehokkaan työkalupaketin, joka tukee esimerkiksi migraatioiden hallintaa (Prisma Data, Inc., 2024) ja tietokannan visualisointia (Prisma Data, Inc., 2024). Tämän myötä kehittäjille mahdollistuu keino hallita tietokannan rakenteen muutoksia helposti ja systemaattisesti. Prisma Client, joka on osa Prisma-ekosysteemiä, tarjoaa TypeScript-pohjaisen asiakasrajapinnan tietokantakyselyiden tekemiseen (Prisma Data, Inc., 2024). Näin ollen Prisma Client mahdollistaa helpon integroitavuuden sovelluslogiikkaan. Tämä tekee Prisma ORM:sta erittäin käyttäjäystävällisen työkalun erityisesti TypeScript-sovelluksille, jossa tietoturva ja tiedon eheys ovat keskeisessä roolissa.

Prisman käyttö vaatii jonkin verran konfigurointia. Prisman konfigurointi suoritetaan pääsääntöisesti Skeema-tiedostossa, jossa määritellään malli, kentät, tietotyypit sekä yhteydet (Kuva 2). (Prisma Dart, 2024.)

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url = env("DATABASE_URL")
}

model customer {
  id       Int      @id @default(autoincrement())
  username String   @unique
  email    String   @unique

  posts    Post[]
}

model post {
  id       Int       @id @default(autoincrement())
  title    String
  content  String?   @db.VarChar(255)
  authorId Int
  author   Customer? @relation(fields: [authorId], references: [id])
}

enum Role {
  USER
  ADMIN
}
```

Kuva 2. Koodiesimerkki Prisman skeematiedostosta (Prisma Data, Inc., 2024).

Vaikka Prisma tarjoaa hyvin selkeän syntaksin, monimutkaisten tietokantarakenteiden hallinta voi kuitenkin vaatia perusteellista perehtymistä Prisman dokumentaatioon ja käytännön opettelua. Prisman tietomalli on myös vahvasti sidoksissa tietokannan skeemaan, mikä tekee siitä vähemmän joustavan verrattuna muihin ORM-järjestelmiin, jotka mahdollistavat suuremman määrän kustomoituja rakenteita. Tämä saattaa rajoittaa sen käyttöä erityisesti organisaatioissa, joissa on usein vaihtuvia ja monimutkaisia tietomalleja. (Htwe, 2023.)

Yksi Prisma ORM:n vahvimmista ominaisuuksista on sen käytettävyys. Kehittäjien arvostama Prisman ominaisuus on erityisesti sen intuitiivinen käyttöliittymää ja sen tarjoama TypeScript-tuki, jotka tekevät koodista helposti ymmärrettävää ja ylläpidettävää. Prisma luo suoraan tietokannan skeemasta TypeScript-tyypit, mikä auttaa kehittäjiä välttämään virheitä ja samalla se tehostaa sovellusten kehitysprosessia (Pellegrini, 2023). Tämän lisäksi Prisma tarjoaa helppokäyttöisen CLI-työkalun, jonka avulla kehittäjän on mahdollista hallita migraatioita ja skeemaa suoraan terminaalista. Migraatioiden ja skeeman hallinta suoraan terminaalista tehostaa kehitystä (Obielum, 2022).

Toinen merkittävä käytettävyyttä parantava tekijä on Prisma Studio, joka tarjoaa visuaalisen työkalun tietokannan hallintaan. Prisma Studion avulla kehittäjät voivat tarkastella ja muokata tietoja suoraan käyttöliittymän kautta ilman, että heidän tarvitsee suorittaa SQL-komentoja. Tämä helpottaa erityisesti aloittelevia kehittäjiä ja nopeuttaa virheiden selvitystä (Prisma Data, Inc., 2024). Prisma studio myös mahdollistaa pienten tietokantakorjausten tekemisen ja samalla edesauttaa ohjelman toimivuuden kokeilua. (Rahman, 2024.)

Yhteenvetona, Prisma ORM tarjoaa kehittäjille modernin ja tehokkaan ratkaisun tietokantojen hallintaan TypeScript- ja JavaScript-sovelluksissa. Sen suorituskyky on optimoitu pienille ja keskisuurille kyselyille, joissa automatisoidut SQL-kyselyt ja numeeriset ID-tunnisteet tuovat merkittäviä etuja. Prisman suurimpia hyötyjä ovat sen käyttäjäystävällisyys, tehokas TypeScript-integraatio ja työkalut, kuten Prisma Studio, jotka parantavat käyttäjän kehityskokemusta.

Heikkouksistaan huolimatta Prisma pystyy tarjoamaan merkittäviä etuja kehittäjille ja organisaatioille, jotka etsivät tehokasta tapaa hallita tietokantojaan. Yhteensopivuusrajoitukset ja joustamattomuus tietyissä skenaarioissa saattavat rajoittaa sen käyttöä, mutta sen hyödylliset ominaisuudet tekevät siitä yhden markkinoiden johtavista ORM-ratkaisuista modernille kehitykselle.

# 3 Arkiston suunnittelu ja menetelmät

## 3.1 Tutkimuksen tavoite

Kehittämistoiminta voidaan rinnastaa toimintatutkimuksen tekemiseen, jolloin tavoitteena on tutkimuksen lisäksi kehittäminen (Heikkinen, 2007). Kehittämistyön tavoitteena on luoda arkisto, joka kattaa projektit, positiot ja työnumerot sekä niihin liittyvät arkistointia vaativat tiedot. Arkiston on oltava TPE:n tiedonhallinnan tehokkuuden takaaja. Arkistointilogiikkaan sisältyvien haku– ja suodatinlogiikoiden on pystyttävä käsittelemään tietoa niin tehokkaasti ja nopeasti, ettei käyttäjäkokemus kärsi. Tämän lisäksi on varmistettava, ettei haettu tieto pääse muuttumaan tietojen palautuksen yhteydessä tietokannasta. Tavoitteiden saavuttamiseksi on määriteltävä vaatimukset tiedon elinkaaren hallinnalle. Hakukyselyn on oltava kestävä, sillä arkistoidun tiedon määrä tulee kasvamaan ajan saatossa. Kestävyyden takaamiseksi hakulogiikan on pystyttävä käsittelemään suuriakin määriä tietoa.

Ennen arkiston kehittämistä tutkitaan, onko Prisma teoriassa hyvä työkalu arkistoinnin toteutukseen. Tarkasteltaessa Prisman ongelmakohtia, arkistointilogiikankannalta tärkeitä seikkoja herää useita kysymyksiä, kuten

1. ovatko nämä ongelmat kriittisiä toiminnanohjausjärjestelmän algoritmilogiikalle?
2. tuleeko ohjelma koskaan hyödyntämään niin suurta datamäärää tai niin monimutkaisia kyselyitä, että ohjelman käyttö hidastuisi merkittävästi tai Prisman tarjoaman tuen rajat tulisivat vastaan?
3. onko Prisma soveltuva työkalu tämän arkistointilogiikan toteuttamiseen?
4. miten arkistointilogiikka voidaan toteuttaa mahdollisimman tehokkaasti käyttäen mahdollisimman kevyitä hakuja ja kyselyitä?

## 3.2 Arkiston suunnittelu

Arkiston suunnittelun yhteydessä määritellään tavoitteet, säilytysaika ja tekniset ratkaisut. Suunnitteluvaiheen jälkeen suoritetaan teoreettisia testejä eri algoritmilogiikoilla ja parhaaksi todettu arkistointilogiikka valitaan toteutukseen. Arkistointilogiikan toteutuksen jälkeen suoritetaan toiminnallinen testaus. Toiminnallisen testauksen tavoitteena on testata logiikan toimintaa ääriolosuhteissa. Toiminnallisen testauksen yhteydessä tietokantaan generoidaan suuri määrä tietoa ja tämä tieto haetaan valitulla logiikalla. Näin varmistutaan siitä, että arkistointilogiikka saavuttaa määritellyt vaatimukset ja tarjoaa käyttäjälle nopean pääsyn arkistoituun tietoon riippumatta tiedon määrästä tietokannassa.

Arkistointia vaativia tietoja ovat projektien tiedot, positiot, jotka kuuluvat projektille sekä positiolle kuuluvat työnumerot. Arkiston tavoitteena on toimia mahdollisimman käyttäjäystävällisesti, jotta tietojen tuominen arkistosta sujuu sujuvasti ja nopeasti. Tämän kriteerin pohjalta voidaan todeta, että arkiston suorittamien hakujen on oltava mahdollisimman kevyitä. Haut suoritetaan käyttäen uniikkia numeroa, joka on ID. Jokaisella projektilla, positiolla ja työnumerolla on oma ID-tunnisteensa. Näin toimitaan, jotta voidaan optimoida hakunopeus tietokannasta ja vältytään virhetilanteilta. Hakukyselyn suorituskyvyn optimoimiseksi vältetään myös kaikkien tietojen hakemista yhdellä kerralla. Projekteille, positioille ja työnumeroille suoritetaan omat kevyemmät haut. Toiminnallisuus tulisi toteuttaa niin, että arkiston logiikka olisi kehittäjille mahdollisimman ylläpidettävä ja helposti luettava. Näin minimoidaan haun kuormaa ja pystytään suorittaa haku mahdollisimman nopeasti. Arkiston toteuttamiseksi suoritetaan kolme erillistä päähakua: yksi projektitiedoille, yksi positiotiedoille ja yksi työnumeron tiedoille. Haut suoritetaan siten, että projektihaku käynnistyy, kun arkistointitoiminto käynnistetään. Tällöin palvelimelta palautuu arkistoituihin projekteihin sopivat kriteerit. Käyttäjälle annetaan lista arkistoiduista projekteista ja he pystyvät valitsemaan projektin, jolloin ohjelma hakee serveriltä projektille kuuluvat positiot. Positiot renderöidään listamuotoiseksi käyttöliittymään, josta käyttäjä voi valita positiolle kuuluvan työnumeron. Työnumerot haetaan palvelinpuolelta vastaavaan tapaan ja renderöidään käyttöliittymään. Tässä opinnäytetyössä keskitytään kuitenkin vain palvelinpuolen tapahtumiin eli haku- ja suodatinlogiikan testaukseen ja parhaan vaihtoehdon löytämiseen.

Arkistointilogiikan vertailukriteereitä ovat suorituskyky, skaalautuvuus, tarkkuus, monimutkaisuus sekä ylläpidettävyys. Kriteerit vastaavat suoraan tavoitteisiin ja käytännön vaatimuksiin.

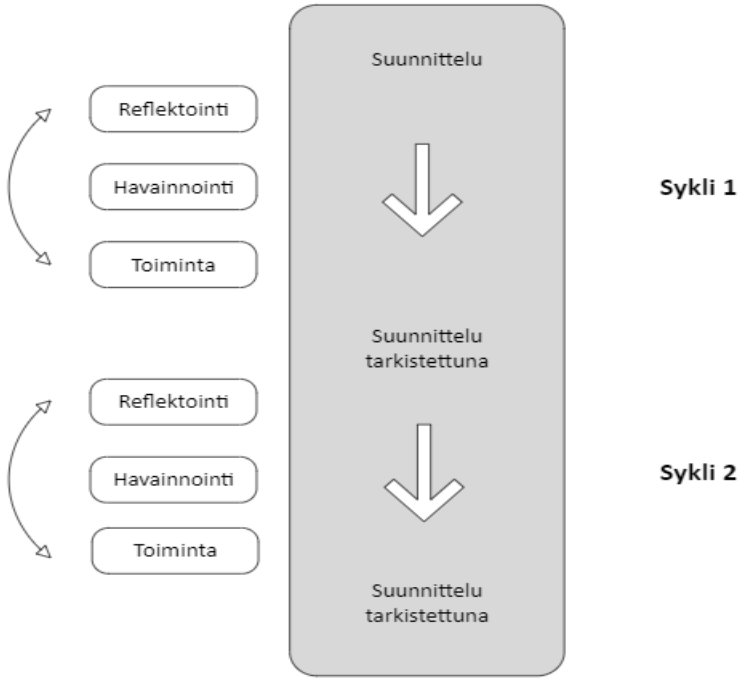
Ennen arkiston toteutusta suoritetaan teoreettisia testejä, jossa mietitään Prisman ominaisuuksien ja suorituskyvyn puolelta parasta vaihtoehtoa arkistointilogiikan projektihaun ja suodatintoiminnon toteuttamiseen. Teoreettisen testien jälkeen valitaan parhaaksi todettu menetelmä arkiston toteutukseen. Arkistointilogiikan valmistuttua tehdään vielä toiminnalliset testit, joissa testataan arkiston toiminnallisuutta ääritapauksissa ja tuodaan esiin hakuihin liittyvät tiedot.

Arkistoidun tiedon hävittämiselle tietokannasta ei tulla luomaan logiikkaa, sillä se ei sisällä arkaluonteisia tietoja, kuten henkilötietoja. Tiedot säilytetään arkistossa myös siksi, että niiden käyttö voi olla tarpeen pitkänkin ajan kuluttua.

## 3.3 Kehittämismenetelmä

Arkistointilogiikan luomisessa ja testaamisessa edetään kehittämismenetelmin. Kehittämistoiminnassa voidaan erottaa viisi erilaista lähestymistapaa: positivismi, interpretatismi, pragmatismi, konstruktivismi ja realismi (Hyötyläinen, 2007). Näistä lähestymistavoista arkistointilogiikan kehittämiseen soveltuvat parhaiten pragmatismi

ja konstruktivismi. Pragmatistinen lähestymistapa perustuu käytännönläheisyyteen, jossa pyritään ratkaisemaan työelämässä esiintyvä haaste tiedon ja työskentelyn avulla (Salonen, 2017). Kehittämistoiminnassa tuotettu tieto palvelee osallistujia eli työyhteisöä hyödyn ja toimivuuden näkökulmista. Pragmatismissa painottuvat toimivuus, käytännöllisyys ja yhteisöllisyys. Pragmatismiin sisältyy myös inhimillisen erehtymisen mahdollisuus, koska kehittämistoiminnan perustaksi ei voida tuottaa täysin varmaa ja erehtymätöntä tietoa (Laitinen, 2014). Konstruktivistinen lähestymistapa painottaa kehitystoiminnan syklisyyttä (kuva) ja kehittämisprosessia, jossa työyhteisön rooli kehittämistyössä ja ongelmanratkaisussa on merkittävä (Salonen, 2017). Kehittämistoimintaan sisältyä reflektio vaatii henkilöitä, joiden esittämät kysymykset ja yhteistyö tuovat lisää näkökulmia kehittämisprosessiin (Syrjälä, 1994). Demokraattisen dialogin tarkoituksena on ohjata, suunnata ja tarkentaa kehittämisen kohteita ja tavoitteita kehityssyklien kautta (Kuva 3) (Salonen, 2017).




Kuva 3. Kuva havainnollistaa kehittämistoiminnan syklisyyttä (Salonen, 2017).

Konstruktivismi edellyttää osallistujilta selkeitä tavoitteita sekä yhteistä kieltä ja toimintatapoja (Salonen, 2017). Kuitenkin vasta itse kehittäminen yrityksen ja erehdyksen kautta tarkentaa lopputulemaa. Konstruktiivinen kehittämisote vaatii tiiviin vuorovaikutussuhteen, johon sisältyy yhteistyö, itseään jatkuvasti korjaava työskentelytapa sekä uuden oppiminen (Hyötyläinen, 2007). Yksilö oppii, kehittyy reflektiivisesti ja kehittää vuorovaikutuksessa työyhteisön kanssa (Laitinen, 2014).

Arkistointilogiikan kehittäminen on toteutettu tiiviissä yhteistyössä työyhteisössä eri osapuolten kanssa. Siihen on sisältynyt opinnäytetyön ja arkistointilogiikan

perusteellinen suunnittelu, tulosten käsittely sekä säännölliset viikkotapaamiset. Tämä yhteistyö on edellyttänyt jatkuvaa ja vuorovaikutteista keskustelua ja tiedottamista, joita on toteutettu hyödyntäen Teams-sovellusta ja suullista keskustelua. Tiedonkulun ja yhteistyön sujuvuus ovat mahdollistaneet kehitysprosessin etenemisen systemaattisesti ja tavoitteellisesti, mikä on ollut keskeistä arkistointilogiikan rakentamisessa.

# 4 Arkistointilogiikan toteutus ja tulokset

## 4.1 Tutkimuksen konteksti

TPE Turun Pelti ja Eristys Oy on rakennusalalla toimiva perheyritys, joka on perustettu vuonna 1984 ja jonka päätoimipiste sijaitsee Piikkiössä. Yrityksessä työskentelee noin 70–80 henkilöä, ja sen toimintaan kuuluvat teräsrakenteiden, ohutlevytöiden ja julkisivuratkaisujen toimittaminen rakennusliikkeille ja teollisuudelle. TPE tavoitteena on tarjota asiakkaille kokonaisvaltaisia palveluja, jotka kattavat suunnittelun, valmistuksen ja asennuksen eri rakennusprojekteihin. Yrityksessä työskentelee myös IT-osaajia, jotka vastaavat ohjelmoinnista ja tietojärjestelmistä. (TPE Turun Pelti ja Eristys Oy, 2024)

Arkistointi on tärkeä osa yrityksen toimintaa, sillä se tukee ajantasaisen tiedon hallintaa ja mahdollistaa aiempien projektitietojen helpon saatavuuden tarvittaessa. Aiemmin arkistointi on hoidettu paperimuodossa, mikä on työlästä ja vie paljon tilaa.

## 4.2 Prisman soveltuvuus arkistointitarkoituksessa

Prisman soveltuvuutta arkistointitarkoituksessa voidaan tarkastella sen tarjoamien ominaisuuksien perusteella. Prisma tarjoaa vahvan tyypityksen, joka minimoi virheiden esiintymisen ja takaa datan pysymisen eheänä ja luotettavana. Prismassa on dynaaminen kyselyrakenne, joka mahdollistaa myös monimutkaisempien tehokkaan haun. Prisma tarjoaa kehittäjille laajan dokumentaation, josta on helppo löytää ohjeita Prisman toiminnallisuuteen liittyen. Dokumentaation ja käytännön kehittämisen tarjoamat edut ovat tuoneet Prisman yhdeksi suosituimmista ORM-järjestelmistä. (Aiyelabegan, 2024.)

Prisma tarjoaa Prisma Studio -työkalun, joka tarjoaa kehittäjälle visuaalisen työkalun tiedon hallintaan tietokannassa. Prisma studio tuo suuria hyötyjä varsinkin kokemattomille kehittäjille, mutta selkeyttää tietokannan rakennetta huomattavasti myös kokeneempien kehittäjien keskuudessa. (Prisma Data, Inc., 2024.)

Prisma pärjää suorituskyvyssä muille kilpailijoille ja Prisma tarjoaa kehittäjälle vaihtoehdon luoda suora SQL-kysely, joka optimoi hakunopeuden (Prisma Data, Inc., 2024). Prisman objektimuotoinen hakukysely on suorituskyvyltään myös varsin nopea, mutta tämän lisäksi se on helppolukuinen ja ylläpidettävä. Objektimuotoinen hakukysely nopeuttaa hakujen luomista huomattavasti, joka puolestaan vauhdittaa kehittämistyötä.

Prisman omaa dokumentaatiota (Prisma Data, Inc., 2024) tarkastellessa voidaan todeta, että se tarjoaa sekä etuja että haasteita verrattuna kilpailijoihinsa. Prisma erottuu erityisesti modernisuutensa ja turvallisuutensa ansiosta, mutta sen käytössä

on myös otettava huomioon ongelmakohtia, kuten suorituskyvyn heikkeneminen erityisesti monimutkaisten tietorakenteiden tai suurten datamäärien käsittelyn yhteydessä. Ominaisuuksiensa puolesta Prisma ORM on erinomaisesti soveltuva työkalu arkiston toteuttamiseksi.

Prisman käyttöönottaminen voi olla haasteellista niille, joilla on aiempaa kokemusta perustason SQL-kyselyistä. Tämä johtuu siitä, että Prisma tarjoaa korkeamman tason abstraktion SQL:ään verrattuna, mikä edellyttää uudenlaisten käsitteiden ja toimintatapojen omaksumista. Näin ollen käyttäjät saattavat kohdata jyrkän oppimiskäyrän siirtyessään perinteisestä SQL:sta Prisman monimutkaisempaan rakenteeseen (Pellegrini, 2023).

### 4.3 Algoritmilogiikan testaus ja arkistoinnin toteutus

Arkistointia vaativat tiedot kattavat laajan valikoiman elementtejä, kuten projektitiedot, positiot sekä positioiden työnumerot. Arkiston hakujen tehokkuuden varmistamiseksi jokaiselle projektille, positioille ja työnumeroille on määritetty uniikki ID-tunniste. Tämä mahdollistaa hakujen optimoinnin ja virhetilanteiden välttämisen, sillä hakutoimintojen suorituskyky paranee, kun tietoja haetaan pienemmissä erissä uniikilla tunnisteella (Schott, 2023). Projekteille, positioille ja työnumeroille toteutetaan erilliset kevyet haut, mikä vähentää hakukyselyjen kuormitusta ja parantaa haun suorituskykyä.

Arkiston toiminnallisuuden tuli olla kehittäjille helposti ylläpidettävä ja ymmärrettävä. Haun yhteydessä oli tärkeää rajoittaa haettavien tietojen määrää vain käyttöliittymässä tarvittaviin tietoihin, mikä minimoi haun kuormaa ja nopeuttaa prosessia (Schott, 2023).

Arkistologiikka toteutettiin suorittamalla vertailua kolmella eri tietokantahakulogiikalla, jonka jälkeen kehiteltiin suodatinlogiikka. Suodatinlogiikka oli tarpeellinen, koska käyttäjällä tulee olla mahdollisuus hakea projektia hakusanalla. Ensimmäinen projektihaku suoritettiin arkistointitoiminnon alkaessa, jolloin palvelimelta palautui arkistoidut projektit kriteerien mukaan. Käyttäjälle esitettiin lista arkistoiduista projekteista, joista hän pystyi valitsemaan haluamansa projektin. Tämän jälkeen ohjelma haki valitulle projektille kuuluvat positiot palvelimelta ja renderöi ne käyttöliittymään. Vastaavalla tavalla haettiin työnumerot, jotka renderöitiin käyttöliittymään. Tässä opinnäytetyössä keskityttiin kuitenkin ensisijaisesti palvelinpuolen tapahtumiin, erityisesti haku- ja suodatinlogiikan testaukseen ja optimaalisten ratkaisujen löytämiseen.

Ennen arkiston toteutusta suoritettiin testejä haku- ja suodatinlogiikoille, joissa arvioitiin Prisman ominaisuuksia ja suorituskykyä arkistointilogiikan projektihaku- ja suodatintoimintojen toteuttamiseksi. Testien perusteella valittiin paras menetelmä arkiston toteuttamiseen. Kun arkistointilogiikka valmistui, tehtiin vielä toiminnalliset

testit, joissa arvioitiin arkiston toimivuutta äärimmäisissä tilanteissa ja tuotiin esiin hakuihin liittyvät tiedot.

Laajin päähaku oli projektihaku, koska se kattoi kaikki projektit, joihin kuului arkistoituja projekteja tai joissa oli arkistoituja positioita sekä projektit, joiden positioilla oli arkistoituja työnumeroita. Positiohaku puolestaan suoritettiin käyttäjän valitseman projektin tunnisteen perusteella, jolloin hakuun sisällytettiin valitun projektin arkistoidut positiot ja positiot, joilla oli arkistoituja työnumeroita. Työnumerohaku toteutettiin valitun position tunnisteen perusteella, ja se kattoi kaikki positiolle kuuluvat työnumerot.

Testaus keskittyi projektihakuihin, koska positio- ja työnumerohakuihin sovellettiin samanlaista logiikkaa kuin projektihakuun. Projektihakujen jälkeen testattiin myös projektien suodatusominaisuutta, jossa käyttäjä pystyi rajaamaan hakua projektikohtaisen yrityksen nimen, työmaan nimen, katuosoitteen ja lisätietojen perusteella. Suodatuslogiikoita kokeiltiin erilaisilla vaihtoehdoilla, ja käyttöön valittiin parhaiten toimiva ratkaisu.

Jokaisessa testissä käytettiin Benchmarkify-mittaustyökalua. Testikantaan lisättiin 20 projektia, joilla oli yhteensä 79 positiota ja 504 työnumeroa satunnaisilla arvoilla. Testiympäristönä toimi Linux 5.15.153.1-microsoft-standard-WSL2 x64 -käyttöjärjestelmä, ja kokeilut suoritettiin Node.js-ympäristössä versiossa 20.17.0, käyttäen V8 JavaScript -moottoria (versio 11.3.244.8-node.23). Suorittimena toimi Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 12 ytimellä, ja järjestelmässä oli käytössä 15 GB muistia. Testien suoritusajaksi oli määritelty 25 000 ms.

Ensimmäinen projektien hakulogiikan esimerkki toteutti projektihakuprosessin kolmessa erillisessä osassa. Ensimmäisessä osassa haettiin projektit, jotka olivat arkistoituna kokonaisuudessaan. Toisessa osassa etsittiin projektit, joilla oli vähintään yksi arkistoitu positio. Kolmannessa osassa haettiin projektit, joilla oli positioita, joihin liittyi arkistoituja työnumeroita. Kaikki kolme osahaun kyselyä suoritettiin samanaikaisesti Promise-toiminnon avulla, mikä mahdollisti suorituksen rinnakkaisuuden ja paransi hakuprosessin tehokkuutta (Kuva 4).

```
// Fetch projects that are archived
    const archivedProjectsPromise   prisma.projects.findMany({
      where: {
        isArchived: true,
      },
    });

    // Fetch projects that have archived positions
    const archivedPositionsPromise   prisma.projects.findMany({
      where: {
        Positions: {
          some: {
            isArchived: true,
          },
        },
      },
    });

    // Fetch projects that have archived tasks
    const archivedTasksPromise   prisma.projects.findMany({
      where: {
        Positions: {
          some: {
            Tasks: {
              some: {
                isArchived: true,
              },
            },
          },
        },
      },
    });

    const [archivedProjects, projectsWithArchivedPositions,
projectsWithArchivedTasks]
      await Promise.all([archivedProjectsPromise,
archivedPositionsPromise, archivedTasksPromise]);
```

Kuva 4. Koodi hakee Prismaa käyttäen samanaikaisesti arkistoidut projektit ja projektit, joilla on arkistoituja positioita tai työnumeroita. Haku suoritetaan rinnakkain Promise.all() -menetelmällä.

Toisessa hakuesimerkissä haku toteutettiin myös kolmessa osassa, mutta käytettiin sekventiaalista await-logiikkaa. Ensimmäisessä haussa valittiin projektit, jotka oli kokonaisuudessaan arkistoitu. Toisessa vaiheessa haettiin projektit, joilla oli vähintään yksi arkistoitu positio. Kolmannessa vaiheessa etsittiin projektit, joilla oli positioita, joihin liittyi arkistoituja työnumeroita. Tässä lähestymistavassa await-komentoa käytettiin jokaisen haun yhteydessä, jolloin kyselyt suoritettiin synkronisesti peräkkäin. Koodin suoritus odotti aina edellisen kyselyn valmistumista ennen seuraavan kyselyn aloittamista. Tämä takasi sen, että jokainen haku sai valmiiksi tulokset ennen kuin siirtyi seuraavaan hakuun (Kuva 5).

```
// Fetch projects that are archived
    const archivedProjectsPromise   await
prisma.projects.findMany({
      where: {
        isArchived: true,
      },
    });

    // Fetch projects that have archived positions
    const archivedPositionsPromise   await
prisma.projects.findMany({
      where: {
        Positions: {
          some: {
            isArchived: true,
          },
        },
      },
    });

    // Fetch projects that have archived tasks
    const archivedTasksPromise   await
prisma.projects.findMany({
      where: {
        Positions: {
          some: {
            Tasks: {
              some: {
                isArchived: true,
              },
            },
          },
        },
      },
    });
```

Kuva 5. Koodi hakee Prismaa käyttäen peräkkäin arkistoidut projektit ja projektit, joilla on arkistoituja positioita tai työnumeroita.

Kolmas hakutapa koostui yhdestä kyselystä, joka kattoi kaikki hakukriteerit samanaikaisesti. Hakuehtoihin sisältyivät projektit, jotka oli jo itsessään arkistoitu, projektit, joilla oli arkistoituja positioita, sekä projektit, joilla oli positioita, joihin liittyi arkistoituja työnumeroita. Algoritmi rakentui OR-ehdoilla, mikä tarkoitti, että projekti palautettiin tuloksiin heti, kun jokin ehdosta täyttyi (Kuva 6).

```
const qualifiedProjects = await prisma.projects.findMany({
      where: {
        OR: [
          { isArchived: true },
          {
            Positions: {
              some: {
                isArchived: true,
              },
            },
          },
          {
            Positions: {
              some: {
                Tasks: {
                  some: {
                    isArchived: true,
                  },
                },
              },
            },
          },
        ],
      },
    });
```

Kuva 6. Koodi hakee Prismaa käyttäen yhdellä haulla arkistoidut projektit ja projektit, joilla on arkistoituja positioita tai työnumeroita.

Kaikissa kolmessa hakutavassa tuloksena saadut projektit olivat samoja. Suorituskyvyssä ilmeni kuitenkin merkittäviä eroja. Testitulosten (Taulukko 1) mukaan ensimmäinen hakutapa saavutti 410 operaatiota sekunnissa, toinen 209 operaatiota sekunnissa ja kolmas eli yhdistetty haku ylsi suorituskyvyltään nopeimpaan tulokseen, 582 operaatiota sekunnissa. Haut palauttivat saman tulosjoukon, jonka perusteella kaikkien hakujen tarkkuus toimi erinomaisesti. Yksittäisen haun yksinkertaisuuden vuoksi se on hauista skaalautuvin, sillä se ei sisällä mitään ylimääräisiä toimenpiteitä.

Parhaaksi hakulogiikaksi valittiin ”Yksittäinen haku” (Kuva 6) sen suoritustehokkuuden, selkeyden ja ylläpidettävyyden vuoksi.

Taulukko 1. Kolmen eri tietokantahaun ominaisuuksien vertailu.

| Kriteeri | Promise -haut | Synkroniset Await -haut | Yksittäinen haku |
|---|---|---|---|
| Suorituskyky | Hyvä (410 ops/s) | Heikko (209 ops/s) | Erinomainen (582 ops/s) |
| Skaalautuvuus | Hyvä | Heikko | Erinomainen |
| Tarkkuus | Erinomainen | Erinomainen | Erinomainen |
| Monimutkaisuus | Korkea | Keskitaso | Matala |
| Ylläpidettävyys | Heikko | Keskitaso | Erinomainen |

Projektihakulogiikan jälkeen suoritettiin testit projektien suodatuslogiikan toimivuuden arvioimiseksi kolmen eri suodatinlogiikan avulla. Testiajaksi määriteltiin 25 000 millisekuntia. Ensimmäisessä suodatuksessa hakutermi muunnettiin ensin pieniksi

kirjaimiksi, ja tämän jälkeen suodattamiseen käytettiin includes-metodia (Kuva 7). Tämän suodatinlogiikan suorituskyky oli 271 835 operaatiota sekunnissa (Taulukko 2).

```
const filteredProjects   archivedProjectsUnique.filter((project: Projects)    {
    if (project.companyName.toLowerCase().includes(searchKeyword.toLowerCase())) return true;
    if (project.worksiteName.toLowerCase().includes(searchKeyword.toLowerCase())) return true;
    if (project.streetAddress.toLowerCase().includes(searchKeyword.toLowerCase())) return true;
    if (project.additionalInformation.toLowerCase().includes(searchKeyword.toLowerCase()))
      return true;
    return false;
  });
```

Kuva 7. Koodi suodattaa projektit annetun hakusanan mukaan includes-metodilla tarkistamalla, sisältävätkö kenttien arvot hakutermin välittämättä kirjainkoosta.

Toinen testi toteutettiin projektin määriteltyjä ominaisuuksia hyödyntäen, jolloin suodatus perustui yksittäisiin määritettyihin projektiominaisuuksiin (Kuva 8). Tämän testin suoritusteho oli 239 266 operaatiota sekunnissa (Taulukko 2).

```
const filterWithProperties = () => {
  return archivedProjectsUnique.filter((project: Projects) =>
  ['companyName', 'worksiteName', 'streetAddress', 'additionalInformation'].some((prop) =>
  project[prop].toLowerCase().includes(searchTerm.toLowerCase()))
  )
}
```

Kuva 8. Koodi suodattaa projektit niiden ominaisuuksien perusteella. Se tarkistaa, sisältääkö jokin määritellyistä kentistä hakutermin huomioiden kirjainkoon ja includes-metodia käytetään suodatuksessa.

Kolmannessa suodatinlogiikassa käytettiin Regex-suodatusta (Kuva 9), joka mahdollisti joustavamman hakutermin käsittelyn. Tämän suodatusmallin suorituskyky oli 239 486 operaatiota sekunnissa (Taulukko 2).

```
const filterWithRegex = () => {
  const searchRegex = new RegExp(searchTerm.trim(), 'i')
  return archivedProjectsUnique.filter((project: Projects) =>
  Object.values(project).some((value) => typeof value === 'string' && searchRegex.test(value))
  )
}
```

Kuva 9. Koodi suodattaa projektit regexillä etsimällä hakutermin osumia projektien merkkijonokentistä arkistoiduissa projekteissa.

Suodatustestien tulokset osoittivat, että tehokkain suodatinlogiikka saavutti noin 12 % paremman suorituskyvyn kuin toiseksi sijoittunut vaihtoehto. Includes -metodilla suodattaminen on myös kehittäjän kannalta helpoiten ylläpidettävä ja yksinkertaisin. Kaikki testatut suodatinlogiikat palauttavat saman tulosjoukon.

Taulukossa 2 esitetyt testitulokset osoittivat, että ensimmäinen suodatinlogiikka oli suorituskyvyltään paras vaihtoehto. Näiden tulosten perusteella voitiin päätellä, että ensimmäisen testin mallit olivat optimaalisia projektinhallinnan tukemiseksi, sillä ne osoittivat sekä tehokkuutta että luotettavuutta.

Taulukko 2. Kolmen eri suodatuslogiikan ominaisuuksien arviointi.

| Kriteeri | Includes -funktio | Ominaisuus -suodatin | Regex |
|---|---|---|---|
| Suorituskyky | Hyvä (271 835 ops/s) | Kohtalainen (239 266 ops/s) | Kohtalainen (239 486 ops/s) |
| Skaalautuvuus | Kohtalainen | Heikko | Hyvä |
| Tarkkuus | Erinomainen | Erinomainen | Erinomainen |
| Monimutkaisuus | Matala | Keskitaso | Vaikea |
| Ylläpidettävyys | Erinomainen | Hyvä | Kohtalainen |

Suoritettujen testien perusteella parhaiksi vaihtoehdoiksi algoritmilogiikan toteuttamiseen todettiin yksittäisen haun algoritmi (Kuva 6) ja includes-metodilla toimiva suodatin (Kuva 8). Molemmat olivat myös helppolukuisia ja helposti ylläpidettäviä kehittäjän näkökulmasta. Algoritmilogiikka toteutettiin yhdistämällä parhaiksi todetut menetelmät.

## 4.4 Arkiston toiminnallisuuden testaus

Arkiston toiminnallisuuden testauksessa luotiin tietokantaan huomattava määrä projekteja, positioita ja työnumeroita. Tällä pyrittiin varmistamaan, että arkistointi toimii tehokkaasti myös suurten tietomäärien yhteydessä. Testiympäristöön generoitiin yhteensä 2169 projektia, 54 486 positiota ja 788 617 työnumeroa. Testit suoritettiin käyttäen Noden.js-versio 20.17.0. Testimäärä vastasi arviolta noin 10 vuoden käytön tuloksena olevaa tietomäärää tietokannassa. Arvio on saatu ulkoiselta lähteeltä ja vertauskohteena on ollut toinen vastaavanlainen käytössä oleva ohjelma.

Alun perin suunniteltu yhden kyselyn algoritmilogiikka havaittiin testauksessa riittämättömäksi, sillä JavaScriptin käyttämän Node.js:n palvelinympäristön oletuskäyttöraja ylittyi suuren datamäärän käsittelyssä. Kun tietokannasta haettiin lähes miljoona relaatiossa olevaa kohdetta, järjestelmä yritti ladata kaiken datan kerralla muistiin, mikä johti "heap out of memory" -ohjelmistovirheeseen. Tämä ohjelmistovirhe osoittaa, että Node.js:n oletusmuistiraja rajoittaa kyseisen ratkaisun käytettävyyttä. Vastaan tullutta muistirajaa voidaan halutessaan nostaa, mutta tätä lähestymistapaa ei pidetty kestävänä.

Ongelman ratkaisuksi pyrittiin suorittamaan suodatus suoraan tietokantakyselyn yhteydessä (Kuva 10), mikä paransi suorituskykyä ja muistinkulutusta verrattuna aiemmin valittuun ratkaisuun. Suodatus suoraan tietokannassa mahdollisti tehokkaamman datan käsittelyn, mutta suurten datamäärien kanssa ilmeni edelleen sama oletusmuistirajan ylittymisestä aiheutuva virhetilanne. Tämän vuoksi tietokantasuodatusta ei voitu valita lopulliseksi ratkaisuksi, sillä se ei pystynyt estämään oletusmuistirajan ylittymistä suurten tietomäärien käsittelyssä.

```
const qualifiedProjects = await prisma.projects.findMany({
      where: {
        OR: [
          { isArchived: true },
          {
            Positions: {
              some: {
                isArchived: true,
              },
            },
          },
          {
            Positions: {
              some: {
                Tasks: {
                  some: {
                    isArchived: true,
                  },
                },
              },
            },
          },
        ],
        AND: [
          searchKeyword
            ? {
                OR: [
                  { companyName: { contains: searchKeyword,
mode: 'insensitive' } },
                  { worksiteName: { contains: searchKeyword,
mode: 'insensitive' } },
                  { streetAddress: { contains: searchKeyword,
mode: 'insensitive' } },
                  { additionalInformation: { contains:
searchKeyword, mode: 'insensitive' } },
                ],
              }
            : {},
        ],
      },
    });
```

Kuva 10. Koodi hakee arkistoidut projektit sekä projektit, joilla on arkistoituja positioita ja työnumeroita. Hakusanasuodatus suoritetaan haun yhteydessä.

Lopulliseksi ratkaisuksi valittiin pagination-logiikka, jossa tietokannasta haetaan kerrallaan vain rajallinen määrä tietoa (Kaplia, 2023). Tämä menetelmä vähentää kerralla haettavan datan määrää ja mahdollistaa tietojen hakemisen osissa. Tämän takia oletusmuistiraja ei ylity, koska kerrallaan suoritettu haku on kooltaan niin pieni. Pagination-logiikka toimii siten, että käyttäjän liikkuessa projekteissa alaspäin, tietoa haetaan tietyn pisteen jälkeen lisää pienemmissä osissa. Tämä ratkaisu parantaa järjestelmän kestävyyttä suuria tietomääriä käsiteltäessä. Pagination-logiikan toteuttaminen edellyttää kuitenkin merkittäviä muutoksia käyttöliittymäpuolen logiikkaan ja sen implementointi ei kuulu tämän opinnäytetyön piiriin.

# 5 Pohdinta

Prisman käytössä ei esiintynyt merkittäviä heikkouksia, jotka puoltaisivat sen soveltumattomuutta arkistointilogiikan toteutuksessa. Prisma on arkistointitarkoitukseen sopiva työkalu monesta syystä. Prisma tukee tehokasta tietomallinnusta ja tietokantaa, mikä mahdollistaa tietojen vaivattoman hakemisen tietokannasta (Aiyelabegan, 2024). Prisman selkeä syntaksi ja sen intuitiiviset toiminnot minimoivat virheiden esiintymisen riskin. Prisman tarjoama automaattinen migraatio mahdollistaa tietorakenteiden muuttamisen hallitusti (Obielum, 2022), mikä helpottaa kehittäjien ohjelman hallintaa. Tämän lisäksi, Prisman haut ovat helppolukuisia ja koodi on helposti ylläpidettävää. Tarjolla oleva Prisma Studio helpottaa kehittäjän havainnollistaa tietokantaa visuaalisessa muodossa (Prisma Data, Inc., 2024). Suorituskyvyn puolesta Prisma pärjää hyvin verrattuna kilpailijoihinsa. Visuaalinen tietokantatyökalu on erityisesti aloitteleville kehittäjille hyödyllinen väline. Tämä tekee Prismasta luotettavan työkalun pitkäaikaisen ja skaalautuvan tietojen hallinnan tarpeisiin.

Testien tuloksena todettiin, ettei arkistointilogiikan toteutus ole järkevää tutkituilla algoritmeilla esiintyneen Node.js:n palvelinympäristön oletusrajamuistin ylittymisen ja siitä aiheutuneen ohjelmistovirheen vuoksi. Vaikka muistirajaa voidaankin halutessaan nostaa, todettiin sen olevan riittämätön toimintatapa tulevaisuutta ajatellen. Ohjelman käyttö ei tule hidastumaan merkittävästi. Testeistä kuitenkin oli havaittavissa, että suurten tietomäärien käsittelyssä saattaa ilmetä rajoituksia, jotka liittyvät Node.js:n palvelinympäristön oletusmuistirajan ylittymiseen. Ohjelmistovirhe ei kuitenkaan liity itsessään Prismaan, joten tämä ei estä sen käyttöä lopullisen ratkaisun toteuttamiseksi.

Tiedon elinkaaren (Kuntaliitto, 2021) huomioiminen arkistointilogiikassa tuo monia etuja. Kun tiedot siirretään manuaalisesti tietokantaan, ne ovat helposti saatavilla. Näin saavutetaan hyvä tasapaino käyttökelpoisuuden ja arkistoinnin välillä. Tämä järjestely helpottaa aktiivisten tietojen hallintaa, sillä tuotannon näkymä pysyy siistinä ja ajantasaisena, kun vanhat projektit siivotaan aktiivisten joukosta. Tämänkaltaisessa mallissa keskeistä on, että vain oleelliset ja ajankohtaiset tiedot ovat esillä. Tämä vähentää informaation määrää ja parantaa työntekijöiden keskittymistä aktiivisiin tietoihin, kun tarpeettomat tiedot eivät täytä tuotantonäkymää (Porkka, 2023). Arkistoinnin avulla myös tulevat tietotarpeet voidaan huomioida, sillä tietojen säilytys ilman poistomekanismia mahdollistaa niiden hyödyntämisen tarvittaessa pitkällä aikavälillä. Tämä on arvokasta esimerkiksi projektien jälkiseurannassa tai dokumentoinnissa, joissa tietoja voidaan tarvita myös vuosien kuluttua.

Arkistointilogiikan tehokkuuden saavuttamiseksi on tärkeää käyttää menetelmiä, joilla voidaan minimoida tietokannan kuormitus. Arkistointilogiikan pilkkominen

kolmeen erilliseen hakuun auttaa jaottelemaan arkistointilogiikkaa pienempiin osiin ja parantaa tietokantahakujen suorituskykyä kuormituksen vähentyessä.

Manuaalinen syöttöprosessi voi kuitenkin tuoda mukanaan haasteita, kuten tiedonsyöttövirheitä ja työmäärän kasvua (Simply Flows, 2024). Jatkossa tarkistusprosessit tai automatisoidut ratkaisut voisivat vähentää näitä riskejä ja tehdä arkistoinnista entistäkin tehokkaampaa. Toisaalta nykyinen malli tuo selkeyttä ja vakautta arkistointiin sekä parantaa tietojen hallittavuutta ja saavutettavuutta niin tuotannon kuin arkiston puolella.

Lopulliseksi ratkaisuksi valittu Pagination-menetelmä tarjoaa arkistointilogiikassa merkittäviä etuja. Pagination hakee vain rajatun määrän tietoja kerrallaan (Kaplia, 2023), mikä minimoi palvelimen muistinkäytön ja estää muistin ylittymisen. Tämä tekee siitä erityisen soveltuvan pitkäaikaisten arkistojen käsittelyyn, joissa tietomäärät kasvavat ajan myötä. Vaikka tehokkaita algoritmeja on jo tutkittu, niiden soveltaminen vaatii käyttöliittymän logiikan mukauttamista ja pagination-logiikan lisäyksen palvelinpuolen hakukyselyihin. Näin voidaan varmistaa saumaton yhteys tietokannan ja käyttöliittymän välillä sekä parantaa tietojen saavutettavuutta.

Ongelmat, kuten palvelinympäristön oletusmuistirajoitukset, eivät itsessään ole kriittisiä algoritmilogiikalle. Ne voivat kuitenkin rajoittaa tiettyjen algoritmilogiikan osien toimivuutta tietyissä olosuhteissa. Tämä viittaa siihen, että algoritmi toimii hyvin, mutta ympäristön resurssirajoituksien kanssa voi tulla ongelmia.

Käytetyt kehittämismenetelmät soveltuivat arkistointilogiikan kehittämiseen hyvin. Konstruktivismin (Salonen, 2017) mukainen vuorovaikutteisuus toteutui käytännön työssä työyhteisössä. Kehittämistyön syklisyys (Salonen, 2017) toteutui käytännössä todettaessa tutkituilla algoritmeilla testeissä esiintyneen Node.js:n palvelinympäristön oletusrajamuistin ylittyminen ja siitä aiheutunut ohjelmistovirhe. Tämä kuvaa yhtä kehittämistyön sykliä, jonka jälkeen alkaa uusi sykli valitun menetelmän implementoinnin myötä. Kehittämistyössä toteutunut yrityksen ja erehdyksen mukainen kehittämistapa edustaa pragmatistista lähestymistapaa (Salonen, 2017).

Testeissä käytetty Benchmarkify määriteltiin suorittamaan 25 sekunnin ajan operaatioita ja niistä saatu testitulos oli tulosten keskiarvo. Kaikki testit pyrittiin suorittamaan samoissa olosuhteissa, jotta saatiin mahdollisimman vertailukelpoiset tulokset. Algoritmilogiikan testauksessa suoritettujen operaatioiden suuren lukumäärän perusteella voidaan todeta, että testi kuvaa suorituskykyä luotettavasti. On kuitenkin huomioitava, että testiympäristön muuttujat vaikuttavat suorituskykyyn. Näitä muuttujia ovat esimerkiksi netin latenssi eli viive, välimuisti ja yhteyspoolit (Burk, 2024). Toiminnallisuuden testauksessa tietokantaan generoitiin suuri määrä projekteja, positioita ja työnumeroita satunnaisilla, mutta realistisilla tiedoilla. Hakua kokeiltiin käytännössä useamman kerran, jolloin päädyttiin ohjelmistovirheeseen. Testauksen tosielämää vastaavan testijoukon osoittamien tulosten perusteella voidaan todeta, että myös toiminnallisen testauksen lopputulos on luotettava.

Tulevaisuudessa arkistointiin voisi lisäksi lisätä käyttöliittymään visuaalisen indikaattorin, joka merkitsisi arkistoidut työt kuvakkeella, kun ne ovat olleet arkistossa yli kymmenen vuotta. Tämä visuaalinen merkintä helpottaisi vanhojen tietojen tunnistamista, jolloin voitaisiin harkita, onko näiden tietojen säilyttäminen edelleen tarpeellista vai voisiko niitä poistaa tai siirtää muuhun säilytysmuotoon. Tällainen toiminto tukisi pitkäaikaisen arkistoinnin ylläpitoa ja auttaisi optimoimaan tietokannan resurssien käyttöä.

# 6 Johtopäätökset ja yhteenveto

## 6.1 Johtopäätökset

Tässä opinnäytetyössä tutkittiin arkistointilogiikan toteuttamista Prisma ORM-järjestelmällä toimeksiantajan TPE Turun Pelti ja Eristys Oy toiminnonohjausjärjestelmään. Tutkimuksen pääasiallinen tavoite oli löytää tehokas, kestävä ja helposti ylläpidettävä arkistointialgoritmi, joka toimisi suurella datamäärällä. Teoreettisissa testeissä kokeiltiin kolmea erilaista hakualgoritmia ja kolmea erilaista suodatusalgoritmia pienellä määrällä dataa käyttäen. Parhaaksi todetut menetelmät yhdistettiin yhdeksi arkistointialgoritmiksi.

Varsinainen tutkimusluku alkoi luomalla tietokantaan pieni määrä projekteja, positioita ja työnumeroita ja testaamalla kolmea eri hakualgoritmia ja kolmea eri suodatinalgoritmia. Testeissä tutkittiin toimintojen suorituskykyä ja niiden ylläpidettävyyttä kehittäjän näkökulmasta. Parhaaksi todetut menetelmät yhdistettiin yhdeksi arkistointialgoritmiksi. Testien tavoitteena oli löytää paras mahdollinen algoritmi ja suorittaa arkistoinnin toiminnallisuus sillä.

Kun parhaat haku- ja suodatinalgoritmit yhdistettiin arkistointialgoritmiksi, testattiin toteutetun algoritmin kestävyyttä ja toiminnallisuutta. Testaus toteutettiin generoimalla tietokantaan suuri määrä projekteja, positioita ja työnumeroita. Testin tavoitteena oli varmistua siitä, että algoritmi pystyy hakemaan tiedon oikeanlaisena tietokannasta ja pitämään sen eheänä koko tiedonkäsittelyn ajan. Tavoitteena oli myös testata arkistointialgoritmin kestävyyttä pidemmän aikavälin tietomäärän avulla ja varmistua näin logiikan käytettävyydestä. Testauksen tuloksena havaittiin ohjelmistovirhe, eli Node.js:n palvelinympäristön oletusmuistiraja tuli vastaan logiikan seurauksena. Tästä syystä valittua arkistointilogiikkaa ei voitu toteuttaa, vaan algoritmia jouduttiin hienosäätämään. Algoritmia muokattiin siten, että suodatus suoritettiin tietokantakyselyn yhteydessä. Hakukyselyn toiminnallisuudesta havaittiin, että suodatus tietokantakyselyssä toimi paremmin kuin aikaisemmin luotu arkistointilogiikka suorituskyvyn ja muistin puolesta, mutta lopputuloksena oli useamman testin jälkeen sama ohjelmistovirhe, joka syntyi ympäristön oletusmuistirajan ylittymisen vuoksi. Ohjelmistovirheen ilmaantumisen vuoksi todettiin, ettei uutta arkistointilogiikkaa voida käyttää sen kestävyyden puutteen vuoksi. Tutkimuksen tulokset osoittavat, että arkistointilogiikan toteuttaminen tietokantateoreettisilla menetelmillä suureen datamäärään on hyvinkin haastavaa.

## 6.2 Yhteenveto

Yhteenvetona päädyttiin lopputulokseen, että tutkitut keinot ovat käytännöllisiä ja tavoitteet täyttäviä pienissä tietokannan tietomäärissä. Tietomäärän kasvaessa

tutkitut arkistointilogiikat eivät ole enää riittävän kestäviä TPE Turun Pelti ja Eristys Oy:n tarpeeseen. Syynä tähän on todettu ohjelmistovirhe, joka syntyi Node.js:n palvelinympäristön oletusrajamuistin ylittymisestä suuren tietomäärän käsittelyssä tietokantahaussa.

Arkistointialgoritmin jatkokehityksen seuraava kokeilu tullaan toteuttamaan pagination-menetelmällä. Pagination-menetelmä hakee tietokannasta vain pienen osan kerrallaan ja haettava määrä on kehittäjien määriteltävissä. Menetelmän avulla voidaan luoda toiminto, joka hakee tietoa sen mukaan, kun käyttäjä käy jo haettuja tietoja läpi. Pagination menetelmän toteuttamiseen vaaditaan merkittäviä käyttöliittymämuutoksia, joten sitä ei toteuteta tässä opinnäytetyössä. Tässä opinnäytetyössä käytyjä tutkimustuloksia ja algoritmeja voidaan kuitenkin hyödyntää pagination-menetelmän toteutuksessa, sillä logiikan pohja on jo luotu. Palvelinpuolen ohjelmoinnin kannalta menetelmän toteutukseen ei vaadita suuria muutoksia, vain hiukan hienosäätöä. Menetelmässä täytyy määritellä logiikka, jolla haetaan aina seuraava määritelty määrä projekteja tietokannasta.

Opinnäytetyön tekeminen oli vaativaa, sillä aiempaa kokemusta arkistoinnin tekemisestä ei ollut. Tiedonhaun yhteydessä opin paljon uutta asiaa yleisesti ottaen arkistosta ja sen logiikoista sekä ORM-järjestelmistä. Prisma oli jo entuudestaan tuttu, mikä helpotti hakukyselyiden luomista, mutta kyselyiden luomisessa oli myös haettava paljon tietoa, jotta hakukysely saatiin optimoitua mahdollisimman hyvin. Banchmarkifyn näyttäminen oli uutta asiaa ja vaati hetken aikaa ennen kuin sen käyttö alkoi onnistua. Tuloksien seuraaminen oli mielenkiintoista ja testien tulokset olivat joissain määrin poikkeavia oletuksistani. Testit opettivat paljon kokeiltujen toiminnallisuuksien suorituskyvystä ja rakenteesta. Niiden yhdistäminen yhteen tietokantakyselyyn oli toimintatapa, joka ei alun perin käynyt mielessä.

Vaativuudesta huolimatta opinnäytetyö oli opettava. Tiedonhaun ja käytännöllisen toteutuksen seurauksena ymmärtää paremmin yleisesti arkistosta ja sen logiikoista sekä toiminnoista, ORM-järjestelmistä ja erityisesti Prismasta ja sen ominaisuuksista.

# Lähdeluettelo

**TheirStack. 2024.** Companies that use Prisma Cloud. [Online] 2024. [Viitattu: 30. 10 2024.] https://theirstack.com/en/technology/prisma-cloud.

**2Kmediat. 2024.** SQL. [Online] 2024. [Viitattu: 20. 11 2024.] https://www.2kmediat.com/sql/alkeet.asp.

**5.2.2018/1050, Tietosuojalaki.** Tietosuojalaki. [Online] [Viitattu: 4. 12 2024.] https://www.finlex.fi/fi/laki/alkup/2018/20181050.

**Abba, Ihechikara. 2022.** What is an ORM – The Meaning of Object Relational Mapping Database Tools. [Online] 21. 10 2022. [Viitattu: 11. 11 2024.] https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/.

**Adequate, Ritika. 2023.** The pros and cons of using raw SQL versus ORM for database development. [Online] 22. 9 2023. [Viitattu: 30. 10 2024.] https://medium.com/@ritika.adequate/the-pros-and-cons-of-using-raw-sql-versus-orm-for-database-development-e9edde7ee31e.

**Ahmed, Shariq. 2024.** Popular ORMs and Their Difference: Prisma, TypeORM, and Sequelize. [Online] 18. 1 2024. [Viitattu: 11. 11 2024.] https://medium.com/@shariq.ahmed525/popular-orms-and-their-difference-prisma-typeorm-and-sequelize-564a83575eea.

**Aiyelabegan, Taofiq. 2024.** Prisma ORM adoption guide: Overview, examples, and alternatives. [Online] 7. 3 2024. [Viitattu: 10. 11 2024.] https://blog.logrocket.com/prisma-orm-adoption-guide/.

**Algorithm Examples.** Ranking 12 Search Algorithms by Efficiency. [Online] [Viitattu: 4. 12 2024.] https://blog.algorithmexamples.com/search-algorithm/ranking-12-search-algorithms-by-efficiency/.

**AltexSoft. 2023.** Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others. [Online] 23. 5 2023. [Viitattu: 4. 12 2024.] https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/.

**Amplication. 2024.** Top 6 ORMs for Modern Node.js App Development. [Online] 2024. [Viitattu: 30. 10 2024.] https://amplication.com/blog/top-6-orms-for-modern-nodejs-app-development.

**Awati, Rahul. 2023.** Object-relational mapping (ORM). [Online] 2023. [Viitattu: 10. 11 2024.] https://www.theserverside.com/definition/object-relational-mapping-ORM.

**Burk, Nikolas. 2024.** ORM Benchmarks. [Online] 23. 7 2024. [Viitattu: 30. 10 2024.] https://benchmarks.prisma.io/.

**—. 2024.** Performance Benchmarks: Comparing Query Latency across TypeScript ORMs & Databases. [Online] Prisma, 23. 7 2024. [Viitattu: 13. 11 2024.] https://www.prisma.io/blog/performance-benchmarks-comparing-query-latency-across-typescript-orms-and-databases#our-benchmarking-methodology.

**Chia, Austin. 2023.** Data Lifecycle Management: A Complete Guide. [Online] 20. 7 2023. [Viitattu: 30. 10 2024.] https://www.splunk.com/en_us/blog/learn/dlm-data-lifecycle-management.html.

**—. 2023.** Data Lifecycle Management: A Complete Guide. [Online] 20. 7 2023. [Viitattu: 12. 11 2024.] https://www.splunk.com/en_us/blog/learn/dlm-data-lifecycle-management.html.

**Data Science Society. 2023.** What Type of Data Should You Archive and Which Solutions to Use. [Online] 13. 12 2023. [Viitattu: 10. 11 2024.] https://www.datasciencesociety.net/what-type-of-data-should-you-archive-and-which-solutions-to-use/.

**Fikuro. 2024.** Mikä on ERP? [Online] 7. 10 2024. [Viitattu: 11. 11 2024.] https://www.fikuro.fi/blogi/toiminnanohjausjarjestelma.

**GitLab. 2024.** GitLab scalability. [Online] 2024. [Viitattu: 30. 10 2024.] https://docs.gitlab.com/ee/development/scalability.html.

**Hassan, Abdishakur. 2023.** What Is a Command-Line Interface? [Online] 1. 3 2023. [Viitattu: 20. 11 2024.] https://builtin.com/software-engineering-perspectives/command-line-interface.

**Heikkinen, H. 2007.** Toimintatutkimus - Toiminnan ja ajattelun taitoa. [kirjan tekijä] J. & Valli, R. Aaltola. *Ikkunoita tutkimusmetodeihin.* Jyväskylä : PS-kustannus, 2007.

**Htwe, nyilynn. 2023.** Prisma ORM vs. Traditional ORMs: Pros and Cons. [Online] 18. 4 2023. [Viitattu: 10. 11 2024.] https://medium.com/@GopherViergan/prisma-orm-vs-traditional-orms-pros-and-cons-ae450c140f15.

**Hyötyläinen, R. 2007.** Tutkimusavusteisen kehittämisen metodologinen kaksoisluonne. [kirjan tekijä] E. Ramstad ja T. Alasoini. *Työelämän tutkimusavusteinen kehittäminen Suomessa. Lähestymistapoja, menetelmiä, kokemuksia, tulevaisuuden haasteita.* Helsinki : Tykes, 2007.

**Iron Mountain. 2024.** Digitaalisen arkistoinnin edut. [Online] 2024. [Viitattu: 10. 11 2024.] https://www.ironmountain.com/fi-fi/resources/blogs-and-articles/b/7-benefits-of-digital-archiving.

**IT Business Edge. 2011.** Nine Best Practices for Efficient Database Archiving. [Online] 6 2011. [Viitattu: 30. 10 2024.] https://www.itbusinessedge.com/storage/nine-best-practices-for-efficient-database-archiving/.

**Jyväskylän yliopisto.** Mitä on tietoturva? [Online] [Viitattu: 11. 11 2024.] https://www.jyu.fi/fi/yliopistopalvelut/digipalvelut/palvelut/tietoturva/mita-on-tietoturva.

**Kansallisarkisto.** Usein kysyttyjä kysymyksiä. [Online] [Viitattu: 11. 11 2024.] https://kansallisarkisto.fi/usein-kysyttyja-kysymyksia.

**Kaplia, Marharyta. 2023.** Pagination vs. Infinite Scroll vs. Load More Explained. [Online] 11. 7 2023. [Viitattu: 12. 11 2024.] https://crocoblock.com/blog/pagination-vs-infinite-scroll/#pros-of-pagination.

**Kolehmainen, Taru. 2005.** Elinkaarimalli. [Online] Kotimaisten kielten keskus, 29. 9 2005. [Viitattu: 11. 11 2024.] https://www.kotus.fi/nyt/kolumnit_artikkelit_ja_esitelmat/kieli-ikkuna_%281996_2010%29/elinkaarimalli.

**Kuntaliitto. 2021.** Tiedon elinkaari. [Online] 10. 3 2021. [Viitattu: 11. 11 2024.] https://www.kuntaliitto.fi/kuntajohtaminen-ja-digitalisaatio/tiedon-elinkaari.

**Laitinen, I. 2014.** *Se toimii sittenkin. Kohti organisaatiotutkimuksen pragmaattista kompleksisuusteoriaa.* Turku : Turun ammattikorkeakoulu, 2014. Tutkimuksia 42.

**LinkedIn. 2024.** Deciphering the Airbnb Tech Stack: Replicating the Architecture with Python Alternatives. [Online] 2024. [Viitattu: 30. 10 2024.] https://www.linkedin.com/pulse/deciphering-airbnb-tech-stack-replicating-python-fiona-githaiga-6j7cf.

**Lucidea Technologies Corp.** *Lucidea.* [Online] [Viitattu: 30. 10 2024.] https://lucidea.com/blog/records-and-archival-management-within-organizations/.

**Minotti, Mattia. 2023.** Typetta vs Prisma. [Online] 24. 2 2023. [Viitattu: 10. 11 2024.] https://dev.to/minox86/typetta-vs-prisma-15lc.

**Mysliwiec, Kamil.** Prisma. [Online] [Viitattu: 30. 10 2024.] https://docs.nestjs.com/recipes/prisma.

**Obielum, Godson. 2022.** Effortless database schema migration with Prisma. [Online] 22. 7 2022. [Viitattu: 30. 10 2024.] https://blog.logrocket.com/effortless-database-schema-migration-prisma/.

**Opetushallitus.** Datatalousosaamisen perusteita perusopetukseen ja toiselle asteelle. [Online] [Viitattu: 11. 11 2024.] https://www.oph.fi/fi/digiosaaminen/datatalousosaamisen-perusteita-perusopetukseen-ja-toiselle-asteelle/mita-sitten.

**Patrina Corporation. 2023.** What Is Data Archiving: Definition, Benefits, and Best Practices. [Online] 2023. [Viitattu: 10. 11 2024.] https://patrina.com/blog/what-is-data-archiving/.

**Pellegrini, Joseph. 2023.** Why ORMs are not always the way to go. [Online] 31. 7 2023. [Viitattu: 30. 10 2024.] https://fintech.theodo.com/blog-posts/why-orms-are-not-always-the-way-to-go.

**Porkka, Riikka. 2023.** Keskitetty arkistointiratkaisu vapauttaa aikaa tuottavaan työhön. [Online] CGI, 4. 4 2023. [Viitattu: 11. 11 2024.] https://www.cgi.com/fi/fi/blogi/cgi-datacycle360/keskitetty-arkistointiratkaisu-vapauttaa-aikaa-tuottavaan-tyohon.

**Prisma Dart. 2024.** Prisma Schema. [Online] 12. 1 2024. [Viitattu: 30. 10 2024.] https://prisma.pub/getting-started/schema.

**Prisma Data, Inc. 2024.** Engines. [Online] 2024. [Viitattu: 30. 10 2024.] https://www.prisma.io/docs/orm/more/under-the-hood/engines.

**—. 2024.** ORM. [Online] 2024. [Viitattu: 20. 11 2024.] https://www.prisma.io/docs/orm.

**—. 2024.** Overview. [Online] 2024. [Viitattu: 12. 11 2024.] https://www.prisma.io/docs/orm/prisma-schema/overview.

**—. 2024.** Prisma CLI reference. [Online] 2024. [Viitattu: 9. 12 2024.] https://www.prisma.io/docs/orm/reference/prisma-cli-reference.

**—. 2024.** Prisma Client API reference. [Online] 2024. [Viitattu: 9. 12 2024.] https://www.prisma.io/docs/orm/reference/prisma-client-reference.

**—. 2024.** Prisma Studio. [Online] 2024. [Viitattu: 30. 10 2024.] https://www.prisma.io/docs/orm/tools/prisma-studio.

**—. 2024.** Query optimization. [Online] 2024. [Viitattu: 9. 12 2024.] https://www.prisma.io/docs/orm/prisma-client/queries/query-optimization-performance.

**—. 2024.** Raw queries. [Online] 2024. [Viitattu: 12. 11 2024.] https://www.prisma.io/docs/orm/prisma-client/using-raw-sql/raw-queries.

**—. 2024.** What is Prisma ORM? [Online] 2024. [Viitattu: 11. 11 2024.] https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma.

**—. 2024.** Why Prisma ORM? [Online] 2024. [Viitattu: 9. 12 2024.] https://www.prisma.io/docs/orm/overview/introduction/why-prisma.

**Rahman, Md Enayetur. 2024.** Getting Started with Prisma: A Comprehensive Guide. [Online] 25. 10 2024. [Viitattu: 10. 11 2024.] https://medium.com/@enayetflweb/getting-started-with-prisma-a-comprehensive-guide-8edb86a1534d.

**Salonen. 2017.** *Kehittämistoiminta ja kehittämisen menetelmiä ammatillisessa korkeakoulussa.* 2017.

**SAP.** Mikä on ERP? [Online] [Viitattu: 20. 11 2024.] https://www.sap.com/finland/products/erp/what-is-erp.html.

**Schott, Madison. 2023.** 12 SQL query optimization best practices for cloud databases. [Online] 30. 6 2023. [Viitattu: 4. 12 2024.] https://www.thoughtspot.com/data-trends/data-modeling/optimizing-sql-queries.

**Simply Flows. 2024.** Errors from Manual Data Entry and How to Avoid Them. [Online] 2024. [Viitattu: 12. 11 2024.] https://simplyflows.com/errors-from-manual-data-entry/.

**Suomisanakirja.fi. 2024.** algoritmi. [Online] 2024. [Viitattu: 11. 11 2024.] https://www.suomisanakirja.fi/algoritmi.

**—. 2024.** sekventiaalinen. [Online] 2024. [Viitattu: 9. 12 2024.] https://www.suomisanakirja.fi/sekventiaalinen.

**Suullinen tiedonanto, TPE. 2024.** [Online] 2024. [Viitattu: 4. 12 2024.]

**Syrjälä, L. 1994.** Toimintatutkimus ja opettajan ammatillinen kasvu. [kirjan tekijä] L., Ahonen, S., Syrjäläinen, S. & Saari, S Syrjälä. *Laadullisen tutkimuksen työtapoja.* Helsinki : Kirjayhtymä Oy, 1994.

**Takeo. 2019.** How Object-Relational Mapping (ORM) Simplifies Database Integration? [Online] 2019. [Viitattu: 30. 10 2024.] https://www.takeo.ai/insights/how-orm-simplifies-database-integration.

**Tawfik, Mohammed. 2024.** SQL vs. ORM: Choosing the Right Tool for the Job. [Online] 10. 5 2024. [Viitattu: 10. 11 2024.] https://medium.com/@apicraft/sql-vs-orm-choosing-the-right-tool-for-the-job-e0bc8c6fbe62.

**TEK-Tools. 2023.** 8 Best Database Optimization Techniques. [Online] 7. 4 2023. [Viitattu: 10. 11 2024.] https://www.tek-tools.com/database/database-optimization.

**Tina. 2020.** Introduction to Object-relational mapping: the what, why, when and how of ORM. [Online] 19. 11 2020. [Viitattu: 10. 11 2024.] https://dev.to/tinazhouhui/introduction-to-object-relational-mapping-the-what-why-when-and-how-of-orm-nb2.

**Torro, Heidi. 2022.** Tiedon elinkaari julkisessa hallinnossa. s.l. : Tiedonhallintalautakunta, 2022.

**TPE Turun Pelti ja Eristys Oy. 2024.** TPE Turun Pelti ja Eristys Oy. [Online] 2024. [Viitattu: 4. 12 2024.] https://www.tpe.fi/fi/etusivu/.

**Twena-Stern, Tamar. 2024.** Prisma, Drizzle, TypeORM or Sequalize — When Your Focus Is Scale, Which One To Choose? [Online] 6. 5 2024. [Viitattu: 9. 12 2024.] https://medium.com/@tamartwena/node-js-data-access-layer-tools-which-orm-to-choose-40473bc09ad0.

**Upwork. 2024.** What Is Digitalization in Business? Basics and Best Practices. [Online] 2024. [Viitattu: 4. 12 2024.] https://www.upwork.com/resources/digitalization-in-business.