Web-sovellusten suorituskykyoptimointi

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Toukokuu 2025
Juho Suomela

TURUN YLIOPISTO
Tietotekniikan laitos

Juho Suomela: Web-sovellusten suorituskykyoptimointi

TkK-tutkielma, 22 s.
Tietotekniikka
Toukokuu 2025

Web-sovellukset ovat tärkeä osa yritysten toimintastrategiaa ja ne mahdollistavat kaikenkokoisten yritysten liiketoiminnan skaalautuvuuden. Web-sovellusten suorituskyky on tärkeä tekijä, joka vaikuttaa sekä käyttäjien kokemukseen että yritysten liiketoimintaan. Nykypäivänä käyttäjät odottavat yrityksiltä nopeasti latautuvia ja responsiivisia web-sovelluksia. Puutteet web-sovelluksen suorituskyvyssä, kuten pitkät latausajat ja hitaat toiminnot voivat johtaa tyytymättömyyteen ja palveluiden hylkäämiseen.

Tässä tutkielmassa selvitetään miten web-sovellusten suorituskykyä voidaan mitata ja optimoida. Tutkielmassa kootaan alan tieteellisestä kirjallisuudesta löydetyt menetelmät ja käsitellään ne kirjallisuuskatsauksen muodossa. Web-sovellusten suorituskyvyn mittaamiselle löydettiin neljä mittaria, jotka mahdollistavat web-sovellusten suorituskyvyn mittaamisen konkreettisilla indekseillä. Web-sovellusten optimointimenetelmät jaettiin viiteen kategoriaan ja jokaisen kategorian yleisimmät menetelmät käsitellään tutkielman kirjallisuuskatsauksessa.

Asiasanat: verkko-ohjelmointi, suorituskykyoptimointi, JavaScript, web-sovellus

Sisällys

1 Johdanto

Viime vuosien aikana yritykset ovat alkaneet siirtyä perinteisistä työpöytäsovelluksista joustavampiin web-sovelluksiin. Nykyään verkkosivut ovat lähes elinehto yritykselle, joka haluaa ylläpitää ja kasvattaa toimintaansa. Verkkosivujen käyttäjät odottavat nopeita ja responsiivisia sivustoja, joissa sivut latautuvat nopeasti ja käyttöliittymä toimii saumattomasti.

Web-kehittäjien on tärkeää ymmärtää, mitkä tekijät vaikuttavat sovellusten suorituskykyyn ja millä tavoin sovellusten eri osa-alueiden suorituskykyä pystytään mittaamaan ja parantamaan. Aihealueet kuten esimerkiksi sivuston latausnopeuden parantaminen, renderöinnin optimointi sekä resurssien tehokas hallinta ovat kaikki merkityksellinen osa käyttäjälle välittyvän verkkosivun suorituskykyä. Tässä tutkielmassa selvitetään, miten web-sovellusten suorituskykyä voidaan mitata ja määritellä sekä selvitetään tapoja parantaa web-sovellusten suorituskykyä. Tutkielmassa vastataan seuraaviin tutkimuskysymyksiin:

TK1: Miten web-sovellusten suorituskykyä voidaan mitata?

TK2: Millä tavoin web-sovellusten suorituskykyä voidaan optimoida?

Tutkimuskysymyksiin vastataan kirjallisuuskatsauksena, jossa perehdytään tieteelliseen kirjallisuuteen web-sovellusten eri osa-alueiden suorituskyvyn parantamisesta. Tutkielman kirjoitushetkellä JavaScript-ohjelmointikieli ja sen ympärille rakentunut ekosysteemi on vahvassa johtoasemassa web-kehityksessä käytettävien teknologioiden joukossa. JavaScriptiä pystytään soveltamaan tehokkaasti sekä käyttö-

liittymä-, että palvelinpuolella ja tästä syystä tutkielmassa keskitytäänkin pääsääntöisesti JavaScript-pohjaisten web-sovellusten kehitykseen ja kyseistä ohjelmointikieltä ja siihen liittyviä teknologioita pohjustetaan taustaluvuissa yleisellä tasolla.

Tiedonhaku lähdeaineiston löytämiseksi tehtiin pääosin Google Scholarissa ja IEEE Xploressa. Lopulliseen tiedonhakuun käytettiin hakulausetta: *("web application") AND ("performance optimization" OR "performance enhancement")*. Tiedonhakua tuettiin myös Keenious-nimisellä tieteellisen kirjallisuuden hakuun tarkoitetulla tekoälymallilla. Löydettyjä hakutuloksia rajattiin julkaisuvuoden perusteella ja jäljelle jääneistä hakutuloksista valittiin parhaiten tutkielman aiheeseen sopivat tutkimukset tiivistelmätekstien perusteella.

Tutkielma koostuu rakenteeltaan taustaluvuista ja kirjallisuuskatsauksesta. Luvussa 2 käsitellään yleisimmät web-sovellusten kehittämiseen käytettävät teknologiat, esitetään yksinkertainen malli web-sovellusten rakenteelle ja vastataan tutkimuskysymykseen TK1: *Miten web-sovellusten suorituskykyä voidaan mitata?* Luvussa 3 vastataan tutkimuskysymykseen TK2: *Millä tavoin web-sovellusten suorituskykyä voidaan optimoida?* jakamalla web-sovellusten suorituskykyoptimointi pienempiin osa-alueisiin, joissa esitellään aiheeseen liittyvästä kirjallisuudesta löydetyt optimointimenetelmät kullekin osa-alueelle.

2 Tausta

Web-sovellusten optimoinnin ja optimointimenetelmien ymmärtämiseksi on ensin tunnettava web-sovellusten rakenne sekä niihin käytettävät teknologiat. Web-sovellusten optimointimenetelmien hyödyntämisen lisäksi on osattava tunnistaa suorituskyvyn ongelmat web-sovelluksissa ja pystyttävä mittaamaan niitä tarkoilla mittareilla. Tehtyjen mittausten perusteella mahdollistetaan oikeiden optimointimenetelmien valitseminen suorituskyvyssä ilmenneiden ongelmien ratkomiseksi.

2.1 Web-sovellukset

Web-sovellukset ovat vakiinnuttaneet paikkansa muiden sovellustyyppien kuten esimerkiksi työpöytäsovellusten rinnalla. Nykypäivänä niitä käytetään laajasti erilaisiin tarkoituksiin kuten verkkokauppoihin, viestintään, viihteeseen ja yritysten sisäisiin järjestelmiin. Perinteisistä käyttäjän laitteelle asennettavista sovelluksista eroten web-sovellukset toimivat laitteen verkkoselaimessa ja hyödyntävät internet-yhteyttä palvelujen tarjoamiseen. Web-sovellusten kehittäminen tuo mukanaan omia haasteitaan erilaisten laiteympäristöjen hallintaan ja verkkoinfrastruktuurin tarpeeseen liittyen, mutta mahdollistaa skaalautuvien ja helposti saavutettavien sovellusten luomisen. Tässä alaluvussa käsitellään yleisimpiä web-sovellusten kehittämiseen käytettäviä teknologioita ja perehdytään siihen, millaisia web-sovellukset ovat rakenteeltaan.

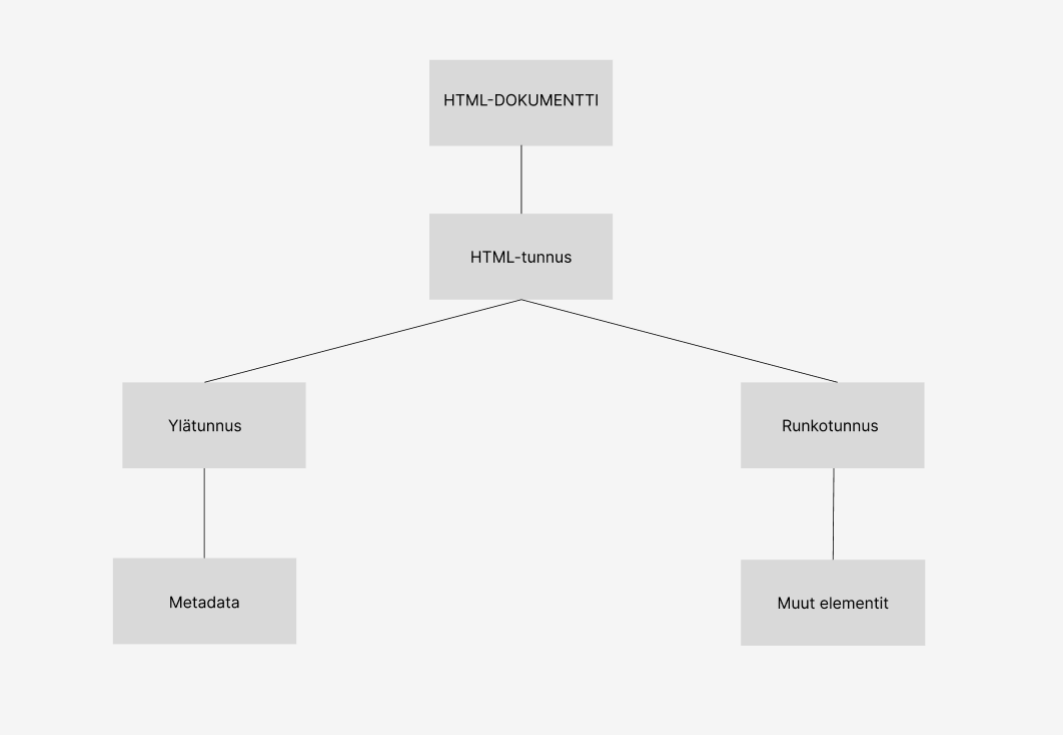
JavaScript, HTML ja CSS

Web-kehitys perustuu useiden keskeisten teknologioiden yhdistämiseen. HTML (HyperText Markup Language), CSS (Cascading Style Sheets) ja JavaScript muodostavat perustan modernille web-sovellusten kehitykselle. HTML toimii web-sovellusten rakenteen määrittäjänä, CSS vastaa niiden visuaalisesta ilmeestä sekä tyylittelystä ja JavaScript on puolestaan vastuussa sovelluksen toiminnallisuudesta ja vuorovaikutteisuudesta. Tässä taustaluvussa käsitellään näitä kolmea teknologiaa pintapuolisesti keskittyen niiden perusominaisuuksiin ja rooliin osana web-sovellusten toimintaa.

JavaScript on yksi web-kehityksen historian tärkeimmistä teknologioista, jonka avulla staattista tekstiä sisältävistä verkkosivuista muotoutui interaktiivisia web-sovelluksia. JavaScript on siis vastuussa web-sovellusten toiminnallisuudesta. JavaScriptin pystyy muovaamaan sovellusten sisältöä ja visuaalista ilmettä tehden sivuista dynaamisia ja interaktiivisia. JavaScriptiä pystytään myös käyttämään palvelinpuolen logiikan määrittelyyn. Tämä tarkoittaa sitä, että web-sovelluksia on mahdollista rakentaa käyttämällä JavaScriptiä ainoana logiikasta vastaavana ohjelmointikielenä. [1]

HTML on merkintäkieli (engl. markup language), jota selaimet käyttävät esimerkiksi tekstin, äänen kuvien ja videoiden kokoamiseen verkkosivuiksi. HTML-dokumentti muistuttaa kuvan 2.1 mukaista puumaista rakennetta, joka muodostuu erilaisista tunnuksista (engl. tag). HTML-dokumentin juurena toimii HTML-tunnus. Kaksi HTML-dokumentin pääelementtiä ovat ylätunnus (engl. head tag) sekä runkotunnus (engl. body tag). Ylätunnus ei pidä sisällään käyttäjälle näytettäviä asioita vaan toimii säiliönä metadatalle kuten esimerkiksi dokumentin otsikolle ja tyylitykselle. Runkotunnus puolestaan sisältää sivulle renderöitävän (engl. render) sisällön. Runkotunnuksen alla voi olla useita eri elementtejä, joilla on omat tunnuksensa sisällöstä riippuen. CSS on kieli, jolla voidaan lisätä tyylejä eli tyylittää (engl. Styling)

HTML-dokumentin määrittelemä sisältö eli tehdä siitä miellyttävämmän näköinen käyttäjälle. [1]



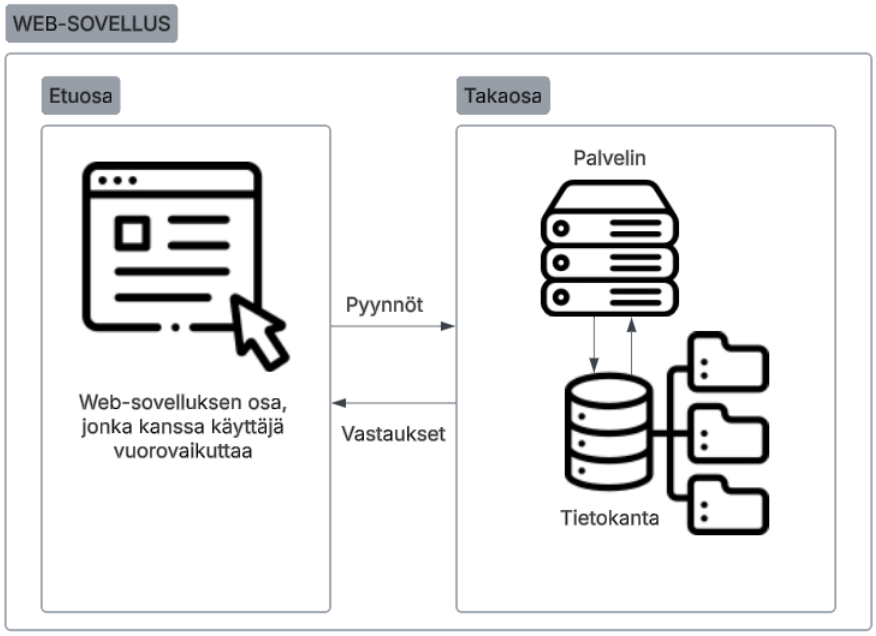
Kuva 2.1: HTML-dokumentin rakenne.

Web-sovellusten rakenne

Suurin osa web-sovelluksista seuraavat kuvassa 2.2 mallinnettua niin sanottua asiakas-palvelin (engl. client-server) -mallia, jossa käyttöliittymä ja tiedon tallennus erotellaan toisistaan. Tämän mallin mukaisesti käyttäjä on vuorovaikutuksessa web-selaimen kautta niin sanotun sovelluksen etuosan (engl. front end) kanssa ja front end puolestaan vuorovaikuttaa selaimen avulla sovelluksen logiikan eli takaosan kanssa (engl. back end). Sovelluksen etuosa on siis HTML:llä, CSS:llä, sekä JavaScriptillä kirjoitettu käyttöliittymä, jonka avulla käyttäjä on vuorovaikutuksessa palvelinsovelluksen kanssa. [2]

Sovelluksen takaosa yksinkertaistettuna sisältää palvelinohjelmia (engl. server) sekä tietokantoja. Palvelimen tehtävä on käsitellä etuosasta lähetetyt pyynnöt ja

välittää tietokannasta haettu data sovelluksen etuosaan, jossa se näytetään käyttäjälle. Tietokanta puolestaan säilöö web-sovelluksen toiminnalle oleellisen datan. Web-sovelluksen kasvaessa voidaan ottaa käyttöön useampia palvelinohjelmia sekä tietokantoja jakamaan vastuuta sovelluksen toiminnasta, mikä mahdollistaakin web-sovelluksien skaalautuvuuden. Tietokantoihin kohdistuvien kyselyiden määrän kasvaessa kuormitusta kevennetään usein tallentamalla osa aiemmin haetusta datasta välimuistiin. Välimuistin tehtävä on säilöä käyttäjän usein tarvitsemaa dataa, joko selaimen välimuistiin tai erilliseen välimuistijärjestelmään sovelluksen takaosassa. Välimuistiin tallennettua dataa voidaan käyttää uudelleen ilman, että sitä tarvitsee hakea jälleen tietokannasta. [2]



Kuva 2.2: Yksinkertaisen web-sovelluksen rakenne

2.2 Suorituskyvyn määritteleminen

Suorituskyky on yksi keskeisimmistä tekijöistä, mikä määrittää ohjelmistojen ja erityisesti web-sovellusten käytettävyyden ja käyttäjäkokemuksen laatua. Se kuvaa kykyä vastata käyttäjien tarpeisiin, kuten sivun lataamiseen ja toimintojen suorittamiseen mahdollisimman nopeasti. Nopeus ei kuitenkaan ole itseisarvo suorituskykyä mitatessa. Myös käyttäjälle näytettävän sekä käyttäjän palvelimelle lähettämän datan tulee pysyä eheänä sen kulkiessa sovelluksen eri osien läpi. Web-kehityksessä suorituskyky ei ole pelkästään tekninen kysymys, vaan sillä on myös suoria vaikutuksia sekä liiketoimintaan että käyttäjätyytyväisyyteen.

Erityisesti nykypäivänä, käyttäjien odottaessa nopeita ja saumattomia käyttökokemuksia, on suorituskyvyn merkitys kasvanut entisestään. Hitaat latausajat, viiveet ja palvelukatkot voivat johtaa käyttäjien turhautumiseen ja saattavat pahimmillaan saada asiakkaat hylkäämään palvelun kokonaan. Suorituskyvyn parantaminen ei siis ole pelkästään kehittäjien tekninen tavoite, vaan myös strateginen keino parantaa asiakaskokemusta ja kasvattaa liiketoimintaa.

Tässä luvussa tarkastellaan ensin mitä suorituskyky web-kehityksessä tarkoittaa sekä miten ja millaisilla mittareilla sitä voidaan mitata. Tämän jälkeen syvennytään siihen millaiset tekijät vaikuttavat web-sovellusten suorituskykyyn.

Suorituskyvyn määritteleminen ja mittarit

Web-sovelluksille ennen kaikkea tärkein suorituskyvyn mittari on asiakastyytyväisyys. Sovellusten suorituskykyä täytyy kuitenkin pystyä mittaamaan konkreettisilla indekseillä. Tärkeimpiin web-sovellusten suorituskykyä mittaaviin indekseihin kuuluu muun muassa

- **Vastausaika** (engl. corresponding time) eli aika, joka kestää käyttäjän tekemästä toiminnosta siihen, että toiminto on suoritettu ja palautettu data on käyttäjän nähtävillä.
- **Kuormituskapasiteetti** (engl. load capacity) eli suurin mahdollinen määrä pyyntöjä, jonka järjestelmä pystyy käsittelemään samanaikaisesti ilman suorituskyvyn heikkenemistä.
- **Läpisyöttö** (engl. throughput) eli järjestelmän käsittelemien pyyntöjen määrä tietyssä aikayksikössä.
- **Suorituskyvyn laskurit** nämä laskurit mittaavat palvelimen tai käyttöjärjestelmän suorituskykyä useilla eri indekseillä kuten esimerkiksi järjestelmän kuormitus, olio ja säie määrä, muistin käyttöaste, prosessorin käyttöaste sekä levyn ja tietoverkon tulo ja lähtö. [3]

Ymmärtääksemme miten web-sovellusten suorituskykyä voidaan parantaa, on ensin perehdyttävä tekijöihin, jotka heikentävät web-sovellusten suorituskykyä. Huonosti rakennetut avainsivut, kuten portti- (engl. gateway page), koti- (engl. homepage) tai aloitussivujen (engl. landing page), joihin on sijoitettu liikaa kuvia ja muita esittelyyn käytettäviä ulkoisia elementtejä voivat esimerkiksi heikentää verkkosivujen suorituskykyä. [4]

Suorituskykyä rajoittavat tekijät löydetään järjestelmän suorituskyvyn testaamisella ja tehtyjen muutosten vaikutuksia tutkitaan vertailuanalyysillä (engl. benchmarking). Profiloinnilla tarkoitetaan puolestaan prosessia, jossa tehtyjen testien sekä

vertailuanalyysin perusteella päätellään, mikä tarkalleen ottaen aiheuttaa sovelluksen tämän hetkiset suorituskyvyn ongelmat [5]. Web sovellusten rakenteessa ja suunnittelussa saattaa myös olla käytetty ratkaisuja, jotka eivät tällä hetkellä vaikuta suorituskykyyn huomattavasti, mutta saattavat tuottaa ongelmia tulevaisuudessa. Tällaisia tulevaisuudessa ilmeneviä ongelmia pystytään välttämään oikein toteutetulla suunnittelu- ja kehitysprosessilla. Tällaisiin ongelmiin kuuluvat mm. jatkuvan sekä iteratiivisen kehitysprosessin puute, sovelluksen suorituskyvyn puutteellinen testaus sekä kunnollisen sovelluksen kapasiteetin suunnittelun puute [6].

3 Suorituskyvyn optimointimenetelmät

Web-sovellusten suorituskyvyn parantaminen koostuu monista pienistä tekijöistä sekä käyttäjä-, että palvelinpuolella tapahtuvien prosessien muutoksista ja optimoinneista. Web-sovellusten suorituskyvyn parantaminen voidaan jakaa Shailesh ja Sureshin [6] mallin mukaan useaan tarkempaan osa-alueeseen seuraavasti:

- Pyyntöputken prosessointijärjestelmien optimointi
- Käyttäjäpuolen optimointi
- Palvelinpuolen optimointi
- Staattisten resurssien optimointi
- Suunnittelu- ja suoritusajan optimointi

Mallissa esitetty pyyntöputken (engl. request pipeline) prosessointijärjestelmien optimointi osa-alueena sisältää kaikki pyyntöjen käsittelyyn kuuluvat järjestelmät kuten esimerkiksi selainohjelmiston, tietokannat sekä palvelinsovelluksen. [6] Nykyaikaisia web-sovelluksia käyttäessä suurin osa sivun vastausajasta kuluu sisällön toimitukseen sekä tietokantoihin kohdistuvien kyselyiden suorittamiseen. Prosentuaalisesti tähän saattaa kulua jopa noin 80 % sivun latausajasta, mikä tekeekin sisällön

toimituksista vastaavista pyyntöputken prosessointijärjestelmistä usein hedelmällisen optimoinnin kohteen. [5] Pyyntöputken prosessointijärjestelmien optimointiin kuuluvat menetelmät ovat usein suoraan kytköksissä muiden mallissa esitettyjen osa-alueiden optimointimenetelmiin, joten osa-aluetta ei käsitellä omassa alaluvussaan, vaan siihen liittyviä optimointimenetelmiä käsitellään epäsuorasti muiden osa-alueiden alaluvuissa.

Alan kirjallisuudesta löydetyt optimointimenetelmät on koottu taulukkoon 3.1. Löydetyt optimointimenetelmät on jaettu aiemmin esitetyn mallin mukaisesti neljään kategoriaan poislukien pyyntöpuolen prosessijärjestelmien optimointi. Taulukossa esitetyt lyhenteet juontuvat optimoitujen sovellusten nimistä, jotka on avattu taulukon alaosassa.

Taulukko 3.1: Optimointimenetelmien käyttö kuuden sovelluksen suorituskyvyn parantamiseen neljässä tutkimuksessa

	RT [3]	MS [7]	EC [8]	PO [5]	CO [5]	OS [5]
KÄYTTÄJÄPUOLEN OPTIMOINTI						
Resurssien latausjärjestyksen muutokset	×		×	×	×	×
Avoimen lähdekoodin hyödyntäminen		×				
Resurssien latauksen viivästäminen				×	×	×
Nimipalvelukyselyiden vähentäminen			×	×	×	×
Sisällön asynchroninen lataus	×					
Evästeiden vähentäminen tiedonsiirrossa	×					
Selaimen välimuistin hyödyntäminen			×			
PALVELINPUOLEN OPTIMOINTI						
Palvelinpuolen sivutus (engl. pagination)		×				
Välimuistitus	×	×	×	×	×	×
Luku- ja kirjoitustoimintojen erottelu	×	×				
Tietokannan hienosäätö		×		×	×	×
Palvelimen hienosäätö	×	×	×	×	×	×
Sovelluslogiikan tehostaminen				×	×	×
Kuorman jakaminen usealle palvelimelle	×	×				
Massamuistin päivitys puolijohdelevyihin	×					
Uudelleenohjausten vähentäminen			×	×	×	×
Sisällönjakelu käyttäjän sijainnin mukaan			×			
STAATTISTEN RESURSSIEN OPTIMOINTI						
Kuvien optimointi			×	×	×	×
HTTP-pyyntöjen määrän ja koon vähentäminen	×	×	×	×	×	×
Pakkaaminen	×		×	×	×	×
Resurssien prosessointi päivittämättä sivua		×				
SUUNNITTELU- JA SUORITUSAJAN OPTIMOINTI						
Koodiin tehdävät parannukset	×	×	×			

RT Roadtransport MS Online mall system EC E-commerce PO Portal (Joomla)

CO Community (PhpBB) OS E-commerce (OsCommerce)

3.1 Käyttäjäpuolen optimointi

Käyttäjäpuolen optimointi keskittyy siihen, miltä sovelluksen käyttäminen tuntuu käyttäjälle. Tähän osa-alueeseen kuuluvat sovelluksen nopeuttamisen lisäksi myös sellaiset muutokset, jotka saavat käyttäjän mieltämään sovelluksen nopeampana kuin mitä se todellisuudessa on. [6] [7] Käyttäjäpuolen optimointiin kuuluu esimerkiksi kuvien ja muiden resurssien lataaminen vain tarvittaessa ja asynkronisesti, (engl. asynchronous) eli samanaikaisesti sovelluksen muiden toimintojen kanssa [6]. Avoimen lähdekoodin käyttöliittymän kehysten (engl. framework) jatkuvan kehittymisen myötä natiivisivujen korvaaminen tällaisella kehyksellä on myös varteenotettava tapa parantaa koodin selkeyttä ja vähentää koodin tarpeetonta toistoa [7]. Tällä hetkellä kolme käytetyintä käyttöliittymän kehystä ovat React, Angular ja Vue.js [9] [10].

Taulukosta 3.1 nähdään, että lähdemateriaalissa tarkasteltavien sovellusten optimointiin tehtiin kaikkiin käyttäjäpuolen optimointia jossain muodossa. Yleisimpänä käyttäjäpuolen optimointimenetelmänä oli resurssien latausjärjestyksen muutos, jota käytettiin viiden sovelluksen optimointiin kuudesta tutkimuksissa mukana olleesta sovelluksesta. Web-sivun latausprosessi alkaa sillä että selain lataa ensin pelkän HTML-dokumentin, joka ei vielä itsessään sisällä kuvia, tyylitystä tai JavaScriptillä luotua logiikkaa. Tämän jälkeen selain jäsentää ja lataa näkyviin HTML-dokumentin aloittaen ylätunnuksen jäsentämisestä ja pysäyttää latausprosessin pyytääkseen mahdolliset vastaan tulevat staattiset resurssit palvelimelta. Ylätunnuksen analysoinnin jälkeen selain toistaa saman prosessin runkotunnukselle. Latausprosessin pysäyttäminen resurssien pyytämistä varten saa verkkosivun latautumisen vaikuttamaan hitaammalta käyttäjälle. Näin ollen suorituskyvyn parantaminen resurssien latausjärjestystä muuttamalla perustuukin siihen, mihin viittaukset resursseihin on sijoitettu HTML-dokumentissa. [3] [8] Resurssien latausjärjestystä muuttamalla pystytään helposti vaikuttamaan siihen, miten käyttäjä mieltää sovelluksen

latausnopeuden. Esimerkiksi siirtämällä CSS-tiedostojen lataus HTML-dokumentin pääelementtiin sekä JavaScript-tiedostot latausjärjestyksen pohjalle, saadaan sivun latautuminen vaikuttamaan todellisuutta nopeammalta, vaikka se kestäisikin todennäköisesti yhtä pitkään [5] [6]. Näkyvien osien lataamisen priorisointi onkin yksi käyttäjäpuolen optimoinnin perusperiaatteista.

Toiseksi suosituin käyttäjäpuolen optimointimenetelmä oli ylimääräisten nimipalvelukyselyiden (engl. domain name system lookup) vähentäminen. Käyttäjän kirjoittaessa haluamansa verkkosivun osoitteen (esimerkiksi www.turku.fi) selaimeen, ei osoite itsessään vielä tarkoita mitään. Osoite tulee ensin kääntää niin kutsutuksi IP- eli internet protokollaosoitteeksi (engl. internet protocol address), jonka avulla selain pystyy kommunikoimaan web-sovellusta tukevan palvelimen kanssa [11]. IP-osoitteet haetaan nimipalvelukyselyillä nimipalvelujärjestelmästä ja jokainen tällainen kysely kestää tavallisesti n. 20-120 millisekuntia. Ennen nimipalvelukyselyn suorittamista selain ei pysty aloittamaan sivun näyttämiseen tarvittavien resurssien lataamista palvelimelta. Näin ollen tarpeettomien nimipalvelukyselyiden määrää rajoittamalla pystytään nopeuttamaan sivun latausaikoja. Nimipalvelukyselyiden määrää pystytään vähentämään vähentämällä uniikkien isäntänimien (engl. hostname) määrää. [8]

Jugo, Kermek ja Meštrović käyttivät tutkimuksessaan [5] kaikkien sovellustensa optimointiin sisällön lataamisen viivästämistä. Tutkimuksessa ei anneta tarkempaa kuvausta menetelmälle, mutta yksi yleisistä tekniikoista viivästää sisällön lataamista on nimeltään laiska lataaminen (engl. lazy loading). Laiska lataaminen perustuu siihen, että kaikkea sisältöä ei pyydetä palvelimelta sivun avaamisen yhteydessä vaan sen sijaan ei-välttämättömän sisällön lataaminen suoritetaan vain tarvittaessa, joka näin ollen nopeuttaa sivun latausaikaa. Laiskan latauksen käyttäminen nopeuttaa sivun alustavaa latausaikaa, mutta saattaa aiheuttaa kooltaan suurille sivuille viivettä käyttäjän siirtyessä sivun eri osioilta toisille. [12]

3.2 Palvelinpuolen optimointi

Suuri osa nykyaikaisten web-sovellusten suorituskykyongelmista liittyy kasvaneeseen samanaikaisuuteen (engl. concurrency), jolla viitataan siihen kuinka monta pyyntöä sovellusta tukevat järjestelmät käsittelevät samanaikaisesti [3] [7]. Palvelinpuolen optimointiin kuuluu mm. palvelinkomponenttien hienosäätöä sekä palvelimen kokoon ja kokoonpanoon tehtäviä muutoksia [6]. Palvelinpuolen laitteiston päivitykset saavat yleisesti ottaen aikaan huomattavimmat vaikutukset samanaikaisuuden aiheuttamiin ongelmiin, mutta ovat myös usein yksi kalleimmista optimoinnin osa-alueista [7].

Laitteistoon tehtävien muutosten lisäksi palvelinpuolen toimintaa pystytään parantamaan myös ohjelmallisilla ja arkkitehtuurisilla ratkaisuilla, joiden avulla pystytään parantamaan suorituskykyä ilman merkittäviä muutoksia laitteistoon. Kuten taulukosta 3.1 käy ilmi, palvelinpuoleen kohdistuvia menetelmiä löytyi eniten ja niitä käytettiin myös eri tutkimusten välillä ristiin enemmän kuin käyttäjäpuolen optimointimenetelmiä. Välimuistin (engl. caching) hyödyntäminen oli palvelinpuolen optimointimenetelmistä suosituin ja sitä käytettiinkin eri muodossa osana kaikkien tutkimusten suorituskykyoptimointia [3] [5] [7] [8]. Esimerkiksi tietokantoihin lähetettyjen kyselyiden vastausten tallentaminen välimuistiin mahdollistaa vastausten uudelleenkäyttämisen ja näin ollen vähentää tietokantoihin kohdistuvaa kuormaa samanaikaisuuden kasvaessa [4]. Pyyntöpuolen prosessointijärjestelmiin kohdistuvassa osiossa mainittiin suurimman osan sivun vastausajasta koostuvan sisällön toimituksesta ja tietokantoihin kohdistuvista kyselyistä. Näin ollen varsinkin tietokantojen kuormaa vähentävät optimointimenetelmät ovat myös hyvä tapa nopeuttaa web-sovelluksen vastausaikaa. Tutkimuksissa Yao ja Xia lähestyivät välimuistin käyttämistä kaikkien staattisten luokkien pakkaamisella ja tallentamisella välimuistiin sekä Memcached-nimisen välimuistitusratkaisun käyttöönotolla. Memcached on monikäyttöinen jaettu välimuistitusjärjestelmä, jolla pystytään säilömään haluttua

dataa väliaikaisesti eri palvelimien vapaaseen muistiin [13]. Memcached on yksi yleisimmin käytetyistä välimuistitusjärjestelmistä ja sitä käyttävät myös esimerkiksi Wikipedia, Facebook ja X (entinen Twitter) [13]. Lähdemateriaalissa Memcachedia käytettiin kahdessa tutkimuksessa [3] [5]. Yao ja Xia käyttivät Memcachedia langattomien yhteyspisteiden (engl. hotspot) datan välimuistittamiseen ja Jugo, Kermek ja Meštrović käyttivät sitä kaikissa kolmessa tutkimukseen käytetyssä sovelluksessa tietokantoihin kohdistuneiden kyselyiden vastausten välimuistittamiseen.

Memcachedille usein vaihtoehtoisena välimuistitusjärjestelmää Redisiä käytettiin yhdessä sovelluksessa [7]. Redisiä käytettiin verkkokauppasovelluksen ostotoimintojen varmistamiseen tallentamalla tuotteiden osto ja ostoskoriin lisäämis- sekä poistotapahtumat ensin välimuistiin, jotta vältytään mahdollisesti turhalta tai haitalliselta tallentamiselta pysyvään tallennustilaan. Ostettujen tuotteiden määrää seuraavat laskurit lisättiin Redis-välimuistiin, minkä tarkoituksena on seurata ostettujen tuotteiden määrää ja estää saldon ylittävät ostotapahtumat. [7] Redis soveltuu tällaisiin varsinaisen tietokannan eheyttä suojeleviin toimintoihin, sillä sen operaatiot ovat atomisia (engl. atomic) eli ne suoritetaan aina kokonaisuudessaan ja operaation keskeytyessä ennenaikaisesti, mitään muutoksista ei tallenneta [7] [14]. Kolmas löytynyt tapa hyödyntää välimuistia suorituskyvyn parantamisessa oli staattisten tiedostojen kuten esimerkiksi kuvien, skriptien ja tyyliohjeiden (engl. stylesheet) tallentaminen selaimen välimuistiin [5] [8]. Tallentamisen jälkeen seuraavilla käyttökerroilla välimuistiin tallennettuja tiedostoja ei tarvitse ladata uudestaan vaan sovellus voi lähettää pyyntöjen mukana *If-Modified-Since* -otsikon (engl. header), jonka avulla palvelin pystyy tarkistamaan, onko kyseinen tiedosto muuttunut edellisen latauskerran jälkeen [8]. Mikäli tiedosto ei ole muuttunut edellisen latauskerran jälkeen, pystyy palvelin palauttamaan pelkän varmistuksen siitä, että tiedosto ei ole muuttunut kokonaisen tiedoston uudelleenlähettämisen sijaan, mikä vähentää ladattavan datan määrää [8]. Vaihtoehtoinen tapa varmistaa välimuistiin

tallennettujen tiedostojen eheys on lähettää tiedostot sisältävien vastausten mukana Expires-otsikko, joka kertoo ajan, jonka jälkeen kyseinen tiedosto on ladattava palvelimelta uudelleen [8] [5].

Palvelimen hienosäätö eri tavoilla osoittautui myös yhdeksi yleisimmistä palvelinpuolen optimointimenetelmistä. Palvelinohjelmien kuten Apachen ja Nginxin asetuksia muuttamalla optimointia tehtiin jossain muodossa kaikille tutkimuksissa optimoiduille sovelluksille. [3] [5] [7] [8] Palvelinohjelmien asetuksiin tehtävät muutokset ovat sovelluskohtaisia ja riippuvat käytössä olevasta ohjelmasta. Apacheen tehtäviä muutoksia olivat esimerkiksi Apachen tarjoamien moniprosessointimoduulien (engl. multi-processing module) käyttöönotto ja hienosäätö, joilla pystytään parantamaan palvelimen kuormituskapasiteettia [3]. Ngixiä käytettiin palvelinohjelmana Cai, Li ja Zhang tutkimuksessa verkkokauppasovellukselle. Tuetun samanaikaisuuden sekä kuormituskapasiteetin lisäämiseksi Nginxin asetuksia muutettiin sallimaan vain prosessorin ydinten määrän verran samanaikaisia prosesseja, sillä yksittäinen ydin pystyy käsittelemään vain yhtä prosessia kerrallaan. Liian suuren prosessien määrän salliminen johtaa suureen määrään säikeistämistä, joka puolestaan hidastaa prosessorin toimintaa säikeiden välillä tapahtuvaan vaihteluun kuluvan ajan vuoksi.

3.3 Staattisten resurssien optimointi

Staattisilla resursseilla tarkoitetaan ikään kuin web-sovelluksen rakennuspalasia. HTML, kuvat, JavaScript ja CSS luokitellaan kaikki staattisiksi resursseiksi. [6] Keskiarvolta vain 20-30 % web-sovelluksen sisällöstä koostuu HTML-sisällöstä, loput 70-80 % koostuu puolestaan staattisista tiedostoista kuten kuvista sekä JavaScript- ja CSS-tiedostoista [6]. Tämä tekeekin nimenomaan näiden staattisten tiedostojen optimoinnista erittäin tärkeän osan web-sovellusten suorituskykyoptimointia.

Staattisten resurssien optimoinnissa keskitytään yleisesti ottaen tiedostojen määrän ja koon pienentämiseen. Staattisten resurssien optimointi on suoraan kytkök-

sissä pyyntöputken prosessointijärjestelmien optimointiin, sillä usein lähdemateriaalissa toistuva tapa parantaa web-sovellusten suorituskykyä staattisten resurssien optimoinnilla on vähentää lähtevien pyyntöjen määrää ja vastausten kokoa, mikä puolestaan vähentää pyyntöputken prosessointijärjestelmien tekemän työn määrää [4] [5] [6] [7]. Lähtevien HTTP-pyyntöjen määrän minimoinnilla vähennetään käytetyn kaistan (engl. bandwidth) sekä siirretyn datan määrää ja näin ollen parannetaan myös sivun vastausaikaa [5]. Staattisien tiedostojen yhdistäminen on tärkeä osa HTTP-pyyntöjen vähentämistä. Staattisia tiedostoja kuten JavaScript- ja CSS-tiedostoja yhdistämällä saadaan vähennettyä ladattavien tiedostojen ja näin ollen myös lähtevien pyyntöjen määrää. Lähtevien pyyntöjen määrää pystyy myös vähentämään haravoimalla ja poistamalla sovelluksen rakenteesta säännöllisesti käyttämättömiä tiedostoja sekä virheellisiä HTTP-pyyntöjä [6]. Datan lähettäminen optimaalisissa tietosäiliöissä (engl. data container) keventää myös pyyntöputken prosessointijärjestelmille kohdistuvaa kuormaa [6]. JSON (JavaScript object notation) on hyvä esimerkki kevyestä ja helposti luettavasta tietosäiliöstä, jolla pystytään siirtämään tekstimuotoista dataa HTTP-pyyntöjen välityksellä. JSONia voidaan käyttää korvaamaan muita raskaampia tietosäiliöitä [6].

Taulukkoa 3.1 tarkasteltaessa huomataan HTTP-pyyntöjen määrän ja koon vähentämisen olevan staattisten resurssien optimoinnin yleisin osa-alue. Pyyntöjen määrää pystytään vähentämään eri tavoilla sekä käyttäjä-, että palvelinpuolella. Käyttäjäpuolella staattisiin resursseihin liittyvien pyyntöjen määrää vähennettiin vähentämällä kyseisten resurssien esiintymistä käyttäjäpuolen logiikassa [7]. Toinen tapa optimoida staattisia resursseja pienentämällä siirrettävän datan määrää on JavaScript-tiedostojen minifiointi (engl. minification) ja obfuskointi (engl. obfuscation). Minifioinnilla tarkoitetaan prosessia, jossa koodista poistetaan kaikki turhat merkit kuten esimerkiksi välilyönnit, rivinvaihdot ja kommentit [8]. Minifioinnin tarkoituksena on pienentää JavaScript-tiedostojen kokoa ja nopeuttaa nii-

den lataamista käyttäjälle. Minifiointia voidaan tehdä JavaScript-, CSS-, HTML- ja PHP-tiedostoille [8]. Obfuskaatio on puolestaan prosessi, jossa koodista muutetaan vaikealukuista ilman, että sen toiminta muuttuu. Obfuskaation tavoitteena on suojata lähdekoodia luvattomalta kopioinnilta ja analysoinnilta, eikä se sinänsä vaikuta merkittävästi sovelluksen suorituskykyyn, mutta tästä huolimatta Jugo, Kermek ja Meštrović listasivat sen yhtenä käytetyistä optimointimenetelmistä. JavaScriptin minifiointia käytettiin osana kahden tutkimuksen suorituskykyoptimointia [5] [8]. Jugo, Kermek ja Meštrović [5] ilmoittivat käyttäneensä minifiointia vain JavaScript tiedostoille, kun taas Bada kertoo minifioineensa kaikki CSS-, HTML-, PHP- ja JavaScript-tiedostot.

Palvelinpuolella pakkaaminen osoittautui yleisimmäksi optimointimenetelmäksi ja kuten taulukosta 3.1 huomataan, sitä tehtiin kolmessa eri tutkimuksessa [3] [5] [8]. Staattiset tiedostot pakkaamalla saadaan vähennettyä siirrettävän datan määrää ja näin tehdään niiden siirtämisestä nopeampaa, mikä puolestaan nopeuttaa sivun latausaikaa [8]. Pakatut tiedostot tulee usein purkaa selaimen toimesta. Huomioon otettavaa on, että jotkin vanhat selaimet eivät tue pakattujen tiedostojen purkamista, vaikka suurin osa uusista selaimista siihen pystyykin. [8]

3.4 Suunnittelu- ja suoritusajan optimointi

Suunnitteluajan optimointiin kuuluu kaikki sovelluksen tuotantoversion ulkopuolella tapahtuva testaaminen ja optimointi. Tähän osa-alueeseen sisältyy esimerkiksi koodin arviointi, kuormituksen mallintaminen (eng. workload modeling) sekä sovelluksen kehitysversion testaaminen. Suoritusajan optimointiin puolestaan kuuluu esimerkiksi reaaliaikainen suorituskyvyn monitorointi. [4] [5] [6]

Web-sovelluksen kehitysvaiheessa säännöllinen ja iteratiivinen testaaminen vähentää myöhemmin tehtävän työn määrää ja mahdollistaa suorituskyvyn pullonkaulojen löytämisen jo kehityksen aikaisissa vaiheissa. Tämä ei kuitenkaan poista

testaamisen tarvetta tuotannossa olevalta versiolta. Kehitysversioon tehtävien muutosten siirtäminen tuotantoversioon tulisi tehdä säännöllisesti pienissä osissa, jotta kehitysversiossa mahdollisesti huomaamatta jääneet ongelmat on helpompi paikantaa tuotantoversiosta [6].

Vaikka suorituskykyoptimointi ohjelmistokehityksessä usein yhdistetään koodin parantamiseen, on lähdemateriaali ristiriitainen aiheeseen liittyen. Esimerkiksi Shailesh ja Suresh listaa tutkimuksessaan useita koodin parantamiseen ja tarkistamiseen liittyviä menetelmiä, kuten esimeriksi automaattisia koodin analysoijia, koodin vertaisarviointia sekä omien laatustandardien luomista ja noudattamista. Shailesh ja Suresh pitivät muiden joukossa edellä mainittuja menetelmiä tärkeänä osana suorituskykyyn koituvien haittojen ennaltaehkäisyä, kun taas Jugo, Kermek ja Meštrović tekemä tutkimus pitää koodin parantamista epäoptimaalisena tapana parantaa web-sovelluksen suorituskykyä, sillä oikein toimiessaan koodin suorittaminen vie vain n. 5-15 % web-sovelluksen vastausajasta. Koodin tarkastamista ja sen laatuun keskittymistä voidaankin pitää ennaltaehkäisevänä tapana välttyä suorituskykyä haittaavilta ongelmilta tulevaisuudessa, kun taas suorituskykyä parannettaessa on usein tehokkaampaa keskittyä muihin osa-alueisiin, joilla on suurempi vaikutus web-sovellusten vastausaikaan.

4 Yhteenveto

Web-sovellusten suorituskyky on keskeinen tekijä sekä käyttäjäkokemuksen, että liiketoiminnan menestyksen kannalta. Nopeat ja optimaalisesti toimivat sovellukset parantavat asiakastyytyväisyyttä ja ovat osa pitkäaikaisten asiakassuhteiden luomista. Tämän tutkielman tavoitteena oli selvittää, miten web-sovellusten suorituskykyä voidaan mitata sekä selvittää millaisilla metodeilla suorituskykyä voidaan parantaa.

Tutkimuskysymyksenä TK1 esitettiin "*Miten web-sovellusten suorituskykyä voidaan mitata?*". Web-sovellusten suorituskyvyn mittaamiseen löydettiin neljä indeksiä, jotka esiteltiin tutkielman luvussa 2.2. Löydetyt indeksit mittaavat web-sovellusten vasteaikaa, kuormituskapasiteettia, läpisyöttöä ja järjestelmän suorituskyvyn laskureita kuten esimerkiksi muistin ja prosessorin käyttöastetta. Löydettyjen indeksien avulla voidaan arvioida, miten hyvin sovellus vastaa käyttäjien odotuksiin ja missä mahdolliset ongelmat sovelluksen suorituskyvyssä sijaitsevat.

Puolestaan tutkimuskysymys TK2 oli "*Millä tavoin web-sovellusten suorituskykyä voidaan optimoida?*". Web-sovellusten suorituskyvyn optimoinnille löydettiin viisi kategoriaa, jotka jakoivat suorituskyvyn optimoinnin selkeämpiin osiin kokonaisvaltaisen tarkastelun sijaan. Pyyntöpuolen prosessointijärjestelmien optimointi keskittyy HTTP-pyyntöjä käsittelevien järjestelmien, kuten selainohjelmiston, tietokantojen sekä palvelinohjelman tehostamiseen. Käyttäjäpuolen optimointi sisältää menetelmiä kuten resurssien latausjärjestyksen optimointi sekä nimipalvelukyselyiden vähentäminen. Palvelinpuolen optimointi puolestaan keskittyy samanaikais-

ten käyttäjien määrän aiheuttamien ongelmien ratkomiseen esimeriksi implementoimalla välimuistitusjärjestelmiä keventämään tietokantoihin kohdistunutta kuormaa. Staattisten resurssien optimointi muodostaa myös oman kategoriansa, jossa pyritään vähentämään tiedostojen määrää ja kokoa. Tämä saavutetaan esimerkiksi yhdistämällä JavaScript- ja CSS-tiedostoja sekä pakkaamalla staattiset tiedostot tiedonsiirron tehostamiseksi. Lopuksi suunnittelu- ja suoritusajan optimointi keskittyy ennakoivaan suorituskyvyn huomioimiseen jo kehitysvaiheessa.

Löydettyjen tulosten perusteella voidaan todeta, että web-sovellusten suorituskyvyn optimointi on moniulotteinen prosessi, joka täytyy muotoilla optimoitavan sovelluksen sekä käyttäjien tarpeiden mukaan. Oikeanlaisella suorituskyvyn mittaamisella pystytään tarkentamaan suorituskykyyn vaikuttavat ongelmatekijät ja valitsemaan niitä parhaiten ratkovat optimointimenetelmät.

Web-sovellusten suorituskyky aiheena koskettaa sekä yrityksiä, että käyttäjiä. Aiheeseen liittyvää tieteellistä tutkimusta oli tästä huolimatta yllättävän vaikea löytää. Aiheen tarkentaminen tiettyyn tai tietyn kategorian menetelmiin olisi saattanut helpottaa lähdeaineiston löytämistä tutkielman suunnitteluvaiheessa. Tehty tutkielma onnistuu kokoamaan yleisimpiä suorituskyvyn optimointimenetelmiä ja helpottaa yksittäisiin optimointimenetelmiin ja niiden vaikutuksiin keskittyvän jatkotutkimuksen tekemisen. Jatkokysymyksenä tutkielmalle voitaisiin lähteä pohtimaan yksittäisten optimointimenetelmien vaikutuksia erilaisten web-sovellusten toimintaan.

Lähdeluettelo

[1] A. Ranjan, A. Sinha ja R. Battewad, *JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript*. Bpb Publications, 2020, ISBN: 9789389328721.

[2] R. T. Fielding ja R. N. Taylor, "Principled design of the modern Web architecture", *ACM Trans. Internet Technol.*, vol. 2, nro 2, s. 115–150, toukokuu 2002. DOI: 10.1145/514183.514185.

[3] Y. Yao ja J. Xia, "Analysis and research on the performance optimization of Web application system in high concurrency environment", teoksessa *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, 2016, s. 321–326. DOI: 10.1109/ITNEC.2016.7560374.

[4] S. Shivakumar ja P. Suresh, "A survey and analysis of techniques and tools for web performance optimization", *Journal of Information Organization*, vol. 8, nro 2, s. 31, 2018. DOI: 10.6025/jio/2018/8/2/31-57.

[5] I. Jugo, D. Kermek ja A. Meštrović, "Analysis and Evaluation of Web Application Performance Enhancement Techniques", teoksessa *Web Engineering*, S. Casteleyn, G. Rossi ja M. Winckler, toim., Cham: Springer International Publishing, 2014, s. 40–56, ISBN: 978-3-319-08245-5. DOI: 10.1007/978-3-319-08245-5_3.

[6] K. Shailesh ja P. Suresh, ”An analysis of techniques and quality assessment for Web performance optimization”, *Indian Journal of Computer Science and Engineering (IJCSE)*, vol. 8, s. 61–69, 2017, ISSN: 0976-5166.

[7] Z. Cai, J. Li ja J. Zhang, ”Research on Performance Optimization of Web Application System based on JAVA EE”, *Journal of Physics: Conference Series*, vol. 1437, nro 1, tammikuu 2020. DOI: 10.1088/1742-6596/1437/1/012039.

[8] A. Bada, ”Performance Optimization of Web-based Application”, *International Journal of Computer Science Engineering*, vol. 10, s. 39–45, huhtikuu 2021. DOI: 10.21817/ijcsenet/2021/v10i2/211002005.

[9] R. Vyas, ”Comparative analysis on front-end frameworks for web applications”, *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, nro 7, s. 298–307, 2022. DOI: 10.22214/ijraset.2022.45260.

[10] S. Rathinam, ”Analysis and Comparison of Different Frontend Frameworks”, teoksessa *International Conference on Applications and Techniques in Information Security*, Springer, 2022, s. 243–257. DOI: 10.1007/978-981-99-2264-2_19.

[11] R. Aitchison, ”An Introduction to DNS”, teoksessa *Pro DNS and BIND*. Berkeley, CA: Apress, 2005, s. 3–19, ISBN: 978-1-4302-0050-5. DOI: 10.1007/978-1-4302-0050-5_1.

[12] R.-M. Bâra, C. A. Boiangiu ja C. Tudose, ”Analysing the performance impacts of lazy loading in web applications”, *Journal of Information Systems & Operations Management*, vol. 18, nro 1, s. 1–15, 2024, ISSN: 1843-4711.

[13] D. Carra ja P. Michiardi, ”Memory partitioning in Memcached: An experimental performance analysis”, teoksessa *2014 IEEE International Conference*

on Communications (ICC), 2014, s. 1154–1159. DOI: 10.1109/ICC.2014.6883477.

[14] M. T. Faridi, K. Singh, K. Soni ja S. Negi, ”Memcached vs Redis Caching Optimization Comparison using Machine Learning”, teoksessa *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, 2023, s. 1153–1159. DOI: 10.1109/ICACRS58579.2023.10404339.