Teemu Lähteinen

LEMONONLINEN KUSTANNUSTEHOKKUU-DEN KEHITTÄMINEN PILVIYMPÄRISTÖSSÄ

Tekniikka
2025

VAASAN AMMATTIKORKEAKOULU
Cloud-based Software Engineering

TIIVISTELMÄ

Tekijä	Teemu Lähteinen
Opinnäytetyön nimi	LemonOnlinen kustannustehokkuuden kehittäminen pilviympäristössä
Vuosi	2025
Kieli	suomi
Sivumäärä	78 + 11 liitettä
Ohjaaja	Harri Lehtinen

Lemonsoft Oyj on ohjelmistoalan yritys, jonka päätuote on Lemonsoft-toiminnanohjausjärjestelmä (ERP). Lemonsoft-ERP:ä käytetään työpöytäsovelluksella, tai LemonOnline-nimisellä verkkoselainsovelluksella. Useimmat asiakkaat käyttävät ohjelmistoa SaaS-mallisena, jossa Lemonsoft hallinnoi järjestelmän palvelimia, tietokantoja ja sovelluspäivityksiä. Lemonsoft aikoo siirtää IT-palvelunsa omasta palvelinympäristöstä Microsoft Azure -pilvipalveluun. Opinnäytetyössä tutkittiin menetelmiä, kuinka LemonOnlinea voidaan kehittää toimimaan kustannustehokkaammin Azure-pilviympäristössä.

Opinnäytetyössä selvitettiin parhaimmat pilvipalvelut LemonOnlinen isännöintiin, sekä niiden kustannusrakenne. Azureen rakennettiin testaamista varten ympäristö, joka käytti mm. Azure App Service, Front Door, SQL Database, Elastic pool ja Cache for Redis -pilvipalveluita. LemonOnlinen suorituskykyä testattiin automatisoidusti Playwright-viitekehyksen avulla. Suorituskykymittareina käytettiin eri pilvipalveluiden resurssien kulutusta, kuten prosessorin käyttöä ja verkkoliikenteen määrää. Testien avulla saatiin käsitys, mitä maksavia resursseja käytettiin eniten.

LemonOnlinen kustannustehokkuuden kehittämismenetelmissä hyödynnettiin tutkimusartikkeleita, jotka käsittelivät verkkosovellusten optimointimenetelmiä ja sovellustason välimuistia. Opinnäytetyön tuloksina syntyi viisi optimointiesimerkkiä, joissa käytettiin sovellustason välimuistia asiakas- ja palvelinsovelluksissa, vähennettiin verkkosivun uudelleenlatauksia, hyödynnettiin NgRx-tilanhallintajärjestelmää, sekä optimoitiin verkkopyyntöjä. Optimointien hyöty todistettiin mittauksilla ja lopuksi tehtiin laskelma, kuinka paljon kustannuksia voitaisiin säästää menetelmiä käyttämällä.

Avainsanat pilvipalvelut, kustannustehokkuus, kehittäminen, ohjelmointi, WWW-sivut

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Cloud-based Software Engineering

ABSTRACT

Author	Teemu Lähteinen
Title	Improving LemonOnline's cost-efficiency in the cloud environment.
Year	2025
Language	Finnish
Pages	78 + 11 Appendices
Name of Supervisor	Harri Lehtinen

Lemonsoft PLC is a company running its business in the software industry. Their main product is an Enterprise Resource Planning (ERP) software called Lemonsoft. Lemonsoft-ERP consists of a desktop application and a web application known as LemonOnline. The software is primarily used as a SaaS-based subscription model, where Lemonsoft manages the servers, databases and software updates. Lemonsoft is planning to move its IT services from on-premises to the public cloud, Microsoft Azure. The goal of the thesis was to find ways to improve LemonOnline's cost-efficiency in the Azure cloud environment.

Thesis started with determining the best cloud services to host LemonOnline, as well as their cost structure. An environment was built to Azure for testing purposes, which used cloud services such as Azure App Service, Front Door, SQL Database, Elastic pool and Cache for Redis. LemonOnline's performance was tested using Playwright, which is an automated end-to-end testing framework. Performance was measured on the different cloud services, using meters such as CPU usage and network request count. Tests helped to gain knowledge which cloud resources were used the most that impact the overall cloud costs.

In the research and development phase, scientific articles were studied regarding web application optimization techniques and application-level caching systems. Based on the findings, five optimization examples were implemented: application-level caching in client-side and backend applications, reducing unnecessary reloading of the web page, using NgRx state management system and optimizing network requests. The optimization benefits could be proven by analyzing the performance meter results. Finally, a calculation was conducted, how much savings could be achieved by utilizing the optimization techniques.

Keywords cloud services, cost effectiveness, development, programming, web pages

SISÄLLYS

TIIVISTELMÄ

ABSTRACT

1 JOHDANTO 9

1.1 Lemonsoft 9

1.2 Lemonsoft-ERP 9

1.3 Opinnäytetyö 10

2 TAUSTA JA TAVOITTEET 11

3 LEMONONLINE 13

3.1 Yleistä 13

3.2 Arkkitehtuuri 13

3.3 Ohjelmointi- ja muut kielet 15

3.3.1 HTML 15

3.3.2 CSS 16

3.3.3 SASS 16

3.3.4 JavaScript 17

3.3.5 TypeScript 17

3.3.6 Visual Basic .NET 17

3.3.7 Transact-SQL 18

3.4 Viitekehykset 18

3.4.1 Angular 18

3.4.2 .NET Framework 19

3.4.3 ASP.NET 19

3.4.4 NgRx 20

3.5 Microsoft SQL Server 20

4 PILVISIIRTYMÄ 22

4.1 Yleistä Azuresta 22

4.2 Azuren palvelut 23

4.2.1 Azure App Service 23

4.2.2 Azure SQL Database ja Elastic pool 24

4.2.3 Azure Front Door 27
4.2.4 Azure Cache for Redis 28
4.2.5 Azure SignalR 30
4.2.6 Azure Monitor 31
4.3 LemonOnline Azuressa 32
4.3.1 Pääkomponenttien siirto pilvipalveluihin 32
4.3.2 Arkkitehtuuri 33
4.3.3 Kustannusrakenne 36
5 OPINNÄYTETYÖPROSESSI 39
5.1 Tutkimuskysymykset 39
5.2 Julkaisut ja teoriat 39
5.3 Tutkimusmenetelmät 44
6 SUORITUSKYVYN TESTAUS JA MITTAUS 46
6.1 Testien valmistelu 46
6.2 Testiympäristön konfigurointi 49
6.3 Testien ajo 50
6.4 Tulokset 51
7 KUSTANNUSTEHOKKUUDEN KEHITTÄMINEN 52
7.1 Välimuistin käyttö asiakassovelluksessa 52
7.2 Verkkosivun uudelleenlataukset 54
7.3 Tilanhallintajärjestelmän hyödyntäminen 57
7.4 Verkkopyyntöjen optimointi 59
7.5 Välimuistin käyttö palvelinpäässä 60
8 TULOKSET 64
9 JOHTOPÄÄTÖKSET 71
9.1 Tutkimusaihe 71
9.2 Testaaminen ja mittarit 71
9.3 Optimoinnit 73
9.4 Tulokset 73
LÄHTEET 75

LIITTEET 79

KUVIO- JA TAULUKKOLUETTELO

Kuvio 1. LemonOnlinen yleisarkkitehtuuri. 14

Kuvio 2. LemonOnlinen kerrosarkkitehtuuri. 15

Kuvio 3. IaaS, PaaS ja SaaS palvelumallien vertailu. (Microsoft, 2025b) 23

Kuvio 4. Azure Front Door -palvelun hinnoittelumalli. (Microsoft, 2023c) 28

Kuvio 5. LemonOnlinen pilviarkkitehtuuri opinnäytetyössä. 34

Kuvio 6. Cache-aside -arkkitehtuurimallin toimintaperiaate. (Microsoft, 2025e) 35

Kuvio 7. Tiedon välivarastoitavuus -vuokaavio. (Mertz & Nunes, 2016, s. 16) 43

Kuvio 8. Opinnäytetyön tutkimuksellisen osan kehitysprosessi. 45

Kuvio 9. LemonOnlinen asiakassovelluksen välimuistin toimintaperiaate. 53

Kuvio 10. Toimintokeskusasetukset LemonOnlinen tilanhallintajärjestelmässä. 58

Kuvio 11. Tietovirta palvelinpään välimuistista asiakassovellukseen. 62

Kuvio 12. App Servicen CPU:n käyttöaste optimointien jälkeen. 66

Taulukko 1. Azure portaalin Dashboardin suorituskykymittarit. 50

Taulukko 2. Mittarit verkkosivun uudelleenlatausten poiston jälkeen. (liite 7) 56

Taulukko 3. Mittarit tilanhallintajärjestelmään käytön jälkeen. (liite 8) 58

Taulukko 4. Mittarit verkkopyyntöjen optimoinnin jälkeen. (liite 9) 60

Taulukko 5. Mittarit palvelinpään välimuistin käytön jälkeen. (liite 10) 62

Taulukko 6. Mittarit, kun kaikki optimointiesimerkit olivat käytössä. (liite 11) 64

Taulukko 7. Optimointien vaikutus kustannuksiin. 67

Taulukko 8. Säästö verkkopyyntöjä vähentämällä. 68

Taulukko 9. Säästö verkkoliikenteen datamäärää vähentämällä. 68

LIITELUETTELO

LIITE 1. Tuotantotyö-testin lähdekoodi

LIITE 2. Ostolasku-testin lähdekoodi

LIITE 3. Myyntitilaus-testin lähdekoodi

LIITE 4. Työaika-testin lähdekoodi

Liitteet poistettu julkisesta versiosta

LIITE 5. Suorituskykymittaukset ilman optimointeja (salainen)

LIITE 6. Suorituskykymittaukset – välimuistin käyttö asiakassovelluksessa (salainen)

LIITE 7. Suorituskykymittaukset – verkkosivun uudelleenlataukset (salainen)

LIITE 8. Suorituskykymittaukset – tilanhallintajärjestelmän hyödyntäminen (salainen)

LIITE 9. Suorituskykymittaukset – verkkopyyntöjen optimointi (salainen)

LIITE 10. Suorituskykymittaukset – välimuistin käyttö palvelinpäässä (salainen)

LIITE 11. Suorituskykymittaukset – esimerkit yhdistettynä (salainen)

1 JOHDANTO

1.1 Lemonsoft

Lemonsoft Oyj on vuonna 2006 perustettu suomalainen ohjelmistotalo, joka tuottaa ohjelmistoratkaisuita asiakasyrityksilleen, kehittääkseen ja kasvattaakseen heidän liiketoimintaansa. Lemonsoftin tuotevalikoimaan ja palveluihin kuuluu mm. seuraavat:

- Lemonsoft – laaja toiminnanohjausjärjestelmä
- LemonOnline – selainpohjainen toiminnanohjausjärjestelmä
- LemonShop – Lemonsoft-toiminnanohjausjärjestelmään integroitu verkkokaupparatkaisu
- Kellokortti – työajanseuranta kaikille toimialoille
- Talosofta – rakennus-, saneeraus- ja talotekniikkaan painottuva toiminnanohjausjärjestelmä
- Finazilla – liiketoimintatiedon raportointi ja ennustaminen
- Finvoicer – laskun elinkaaren hallinta
- Lixani – työajankirjaus ja projektien hallintajärjestelmä erityisesti korjausrakentamiseen
- Metsys – laaja varastonhallintajärjestelmä eri toimialoille

(Lemonsoft, 2025a)

Lemonsoftilla on toimipaikkoja Vaasassa, Tampereella, Helsingissä, Turussa, Joensuussa, Oulussa, Jyväskylässä ja Kouvolassa. Lemonsoftilla oli yli 7000 asiakasta, n. 200 työntekijää ja 22,6 miljoonaa euroa liikevaihtoa vuonna 2022 (Lemonsoft, 2025b). Lemonsoft toimii opinnäytetyön toimeksiantajana.

1.2 Lemonsoft-ERP

Lemonsoft on Lemonsoft Oyj:n kehittämä toiminnanohjausjärjestelmä yritysten liiketoimintojen tehostamiseen. Järjestelmää käytetään työpöytäsovelluksen tai

LemonOnline-nimisen verkkoselainpohjaisen käyttöliittymän avulla. Modernin ulkoasun lisäksi sen käytön etuna on laiteriippumattomuus verrattuna työpöytäkäyttöön. Lemonsoft-ERP:ä käyttävät tuhannet suomalaiset yritykset päivittäisessä liiketoiminnassaan.

Lemonsoft-ERP on mahdollista asentaa asiakkaan omalle palvelimelle, vuokratulle alustalle toiselta palveluntarjoajalta tai Lemonsoft Oyj:n hallinnoimalle alustapalvelulle SaaS (Software as a Service) -mallisena. Suosituin vaihtoehto on SaaS-malli, jossa etuna asiakkaalle on mm. automaattiset ohjelmistopäivitykset, kulujen ennustettavuus, sekä asiakaspalvelun sujuvuus. (Lemonsoft, 2021, s. 8)

1.3 Opinnäytetyö

Julkisten pilvipalveluiden yleistyttyä yritykset haluavat siirtää IT-palveluitaan julkisille pilvialustoille omasta palvelinympäristöstään saavuttaakseen mm. kustannustehokkuutta, tietoturvahyötyjä, toimintavarmuutta ja parempaa skaalautuvuutta. Pilvisiirtymästä koituvat kustannukset ja pilvipalveluiden kulurakenne on hyvä selvittää etukäteen, sekä miten nykyiset IT-palvelut voidaan toteuttaa kustannustehokkaasti uudessa ympäristössä.

Lemonsoft Oyj:n intresseissä on SaaS-mallina tarjottavan Lemonsoft-ERP:n siirtäminen omasta palvelinympäristöstä Microsoft Azure -pilvipalveluun. Opinnäytetyössä tehdään tutkimus ja kehitetään tapoja, joilla LemonOnlinea voidaan muuttaa kustannustehokkaammaksi pilviympäristössä.

2 TAUSTA JA TAVOITTEET

Lemonsoft Oyj:n tarjoamat tuotteet käyttävät nykyisin alustanaan yrityksen omaa palvelinympäristöä. Oman IT-infrastruktuurin ja palveluiden ylläpito vaatii kuitenkin paljon resursseja yritykseltä, jotka voidaan ulkoistaa julkisia pilvipalveluita käyttämällä. Yksi pilvisiirtymän muista hyödyistä on kustannussäästöt, joita saadaan maksamalla vain niistä resursseista mitä käytetään. Resursseilla tarkoitetaan esimerkiksi maksullista laskenta-, muisti- ja tallennuskapasiteettia, joita pilvipalvelut tarjoavat. Kulutetut resurssit vaikuttavat kokonaiskustannuksiin, joten kustannuksia voidaan säästää resurssikulutusta vähentämällä.

Opinnäytetyön päätavoitteena on löytää parhaat keinot, miten LemonOnlinea voidaan kehittää ja optimoida niin, että pilven käytön kustannuksia voidaan pienentää. Sovelluksen käyttö tai ylläpito ei saisi kärsiä muutosten seurauksena. Tuloksena syntyy konkreettisia esimerkkejä, kuinka tämä voidaan tehdä pilvipalveluiden resurssien käyttöä vähentämällä. Resurssien käyttö voi riippua esimerkiksi käytetystä palvelusta Azuressa, sovelluksen koodin tehokkuudesta, arkkitehtuurin ratkaisuista, sekä web-käyttöliittymän ja palvelimen välisestä kommunikaatiosta. Opinnäytetyössä ei tutkita kuinka Azuren palveluita tulisi käyttää mahdollisimman kustannustehokkaasti, sillä niistä on olemassa olevat ja jatkuvasti päivitetyt ohjeet Microsoftin toimesta. Azuren pilvipalveluita käsitellään sen verran kuin tarpeellista, jotta voidaan löytää soveltuvimmat vaihtoehdot LemonOnlinen isännöintiin Azuressa, sekä kartoittaa niiden kustannukset.

Päätavoitteen lisäksi työssä tehdään myös muuta toimeksiantajaa hyödyttävää tutkimusta. Nykyisestä LemonOnlinen arkkitehtuurista muodostuu kokonaiskuva. Azuren tarjoamat palvelut ja niiden kulurakenne hahmottuvat, sekä miten niitä kannattaa hyödyntää järkevästi LemonOnlinen pilvisiirtymässä.

Opinnäytetyön muoto on tutkimuksellinen kehittäminen. Tutkimuksen kohde on LemonOnline-sovellus, työpöytäsovellus pois luettuna, koska LemonOnlinen kehi-

tykseen halutaan panostaa tulevaisuudessa enemmän. Arkkitehtuurillisesti molemmat kuitenkin käyttävät samoja komponentteja, kuten bisneslogiikka, tietokanta ja käyttäjähallinta. Tutkimus rajoittuu LemonOnlinen kustannustehokkuuteen ja sen kehittämistä varten tarvittaviin tietoihin.

3 LEMONONLINE

3.1 Yleistä

Lemonsoft Oyj aloitti vuonna 2006 Lemonsoft-ERP:n kehityksen Windows-työpöytäsovelluksena. LemonOnline kehitettiin sen rinnalle myöhemmin, tuoden ERP:n toiminnot selainkäyttöliittymässä. Ne jakavat saman liiketoimintalogiikan ja tietokannan, eli käyttäjä voi itse päättää kumpaa käyttöliittymää hän haluaa käyttää, käsiteltävien tietojen pysyessä samana taustalla. Selainkäyttöliittymän etuina on laiteriippumattomuus, eli sitä voi käyttää älypuhelimessa, tabletissa ja tietokoneessa. Käyttöliittymä toimii responsiivisesti eri näyttökoolla, mukauttaen automaattisesti sisältönsä näytön kokoon sopivaksi. Responsiivisuuden ansiosta sovelluksen käytettävyys säilyy laitteesta riippumatta. (Lemonsoft, 2025c)

LemonOnlinen avulla käyttäjä voi hallita yrityksensä liiketoimintaprosesseja. Toiminnallisuuksiin kuuluu mm. asiakkuudenhallinta, taloushallinto, logistiikka, tuotannonohjaus, henkilöstöhallinto, raportointi ja projektinhallinta. Yrityksen tietoja tai sovelluksen toimintoja voidaan rajata antamalla käyttöoikeus vain yksittäisille käyttäjille tai tietyn roolin omaaville käyttäjille. Sovelluksen käyttäjät kirjautuvat sovellukseen käyttäjähallinta-mikropalvelun avulla, jossa myös yrityksen pääkäyttäjä voi hallinnoida muiden käyttäjien käyttöoikeuksia. (Lemonsoft, 2025c)

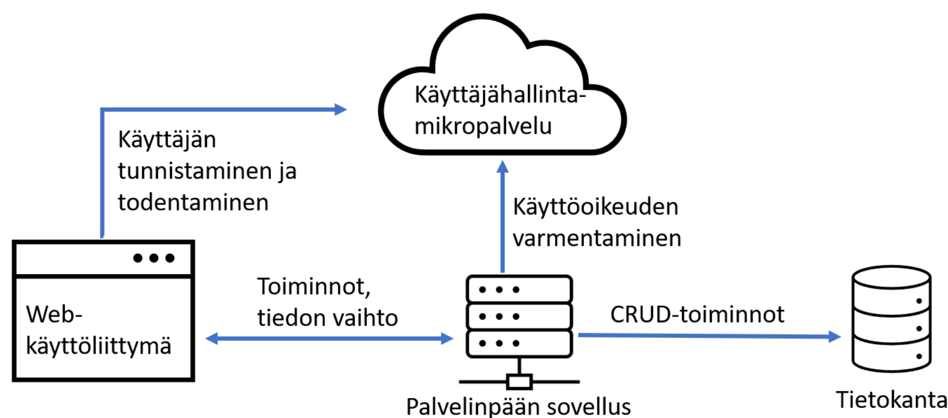
3.2 Arkkitehtuuri

LemonOnlinen arkkitehtuurissa on kolme eri pääkomponenttia: verkkoselainsovellus, palvelinpään sovellus ja tietokanta. LemonOnline hyödyntää myös Azuressa toimivaa Lemonsoftin keskitettyä käyttäjähallinta-mikropalvelua käyttäjän autentikointiin ja käyttöoikeuksien hallintaan.

Verkkoselainsovellus haetaan web-palvelimelta ja se toimii LemonOnlinen käyttöliittymänä. Käyttöliittymä ohjaa käyttäjän kirjautumaan Käyttäjähallinnassa, ennen kuin käyttöliittymä ottaa yhteyden palvelinpään sovellukseen. Käyttöliittymä kommunikoi palvelinpään sovelluksen kanssa lukiessa ja tallentaessa tietoa.

Palvelinpään sovellus todentaa käyttöliittymästä tulevien kutsujen oikeudet Käyttäjähallinnasta saatavien tietojen perusteella ja suorittaa halutut toiminnot. Näitä ovat esimerkiksi CRUD (Create, Read, Update, Delete) -toiminnot tietokantaan. Palvelinpää myös informoi käyttöliittymää palvelinlähtöisesti, jos olennaisia tietoja muuttuu ja käyttöliittymässä näkyvät tiedot pitää päivittää.

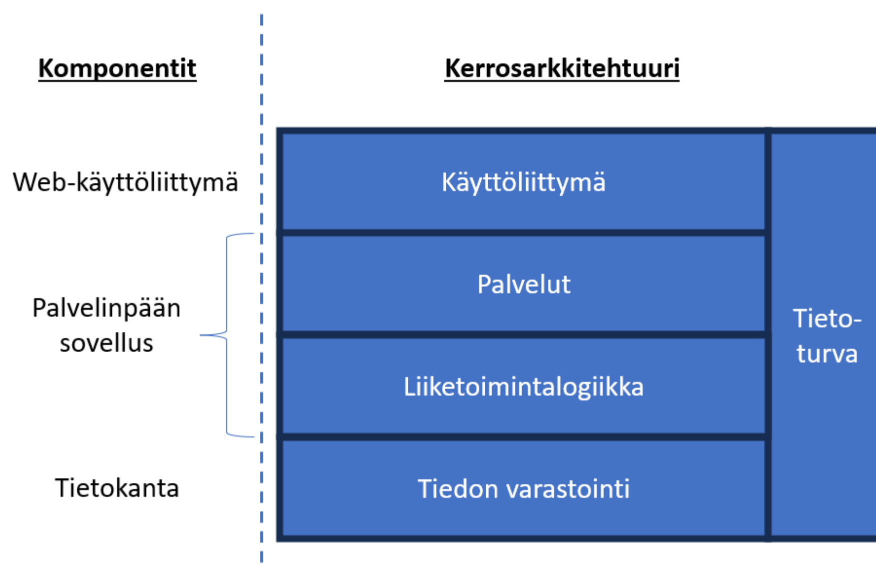
Kuvio 1 esittää LemonOnlinen yleisarkkitehtuurin.



Kuvio 1. LemonOnlinen yleisarkkitehtuuri.

LemonOnline-sovelluksen kerrosarkkitehtuuri koostuu käyttöliittymä-, palvelu-, liiketoiminta- ja tietokantakerroksista. Käyttöliittymä-kerros pitää sisällään LemonOnlinen web-käyttöliittymän. Käyttöliittymä on yhteydessä palvelut-kerrokseen, joka koostuu palvelinpään sovelluksen rajapinnasta. Palvelut-kerros hakee tiedon liiketoimintalogiikka-kerroksesta ja palauttaa sen käyttöliittymä-kerrokseen. Liiketoimintalogiikka-kerros prosessoi tiedon, hakee, lisää, poistaa tai muokkaa tietoa tiedon varastointi -kerroksesta ja palauttaa sen lopulta ylemmille kerroksille. Kaikki kerrokset ovat yhteydessä tietoturva-kerrokseen, joka pitää sisällään tietoturvapalveluita, kuten käyttäjän todentaminen, käyttöoikeuksien tutkiminen, tiedon salaaminen ja väärinkäytön estäminen.

Kuvio 2 esittää kerrosarkkitehtuurin kerrosten liityntää toisiinsa, sekä niiden sijaintia yleisarkkitehtuurin komponenteissa.



Kuvio 2. LemonOnlinen kerrosarkkitehtuuri.

3.3 Ohjelmointi- ja muut kielet

3.3.1 HTML

HTML (HyperText Markup Language) on merkintäkieli, joka määrittää verkkosivun hierarkkisen rakenteen ja sisällön. Merkintäkieleen kuuluu eri tarkoitukseen käytettäviä elementtejä, joiden avulla voidaan esittää esimerkiksi tietoa, kuvia, taulukoita, listoja. Osa elementeistä on toiminnallisuutta varten, kuten painikkeet, linkit ja lomakkeet. HTML-elementit koostuvat aloitus- ja lopetustageista, joilla on uniikki nimi tiettyä tarkoitusta varten. Sisältö asetetaan tagien väliin, kuten teksti, tai muita elementtejä. Elementeille voidaan asettaa attribuutteja tyylittelyjä ja ominaisuuksia varten. (Mozilla, 2024a)

Verkkoselain lataa HTML-dokumentin web-palvelimelta, eli tiedoston .html -päätteellä ja käyttää sitä verkkosivun pohjana. HTML-dokumentti määrittää muut verkkosivun riippuvuudet, kuten JavaScript ja CSS-tiedostot.

3.3.2 CSS

CSS (Cascading Style Sheets) on verkkosivuston sisällön tyylit määrittävä tyylitiedostokieli. Tyylit voidaan määrittää suoraan HTML-dokumentin styles-tagien sisään, tai viitata erilliseen tyylitiedostoon link-elementin avulla, joka tuo tyylitiedoston sisältämät tyylit verkkosivun käyttöön. Tyylit voi myös kirjoittaa suoraan HTML-elementteihin styles-attribuutin arvoksi. Erillinen tyylitiedosto helpottaa ylläpitoa ja vähentää ohjelmointityön määrää. Tyylitiedostot tunnistetaan niiden nimen .css-päätteestä.

Kielen syntaksissa valitaan elementti tai elementit, joita säännöt koskevat tagien nimet kirjoittamalla, sekä ympäröimällä säännöt aaltosulkeilla. Elementeillä on tietyt tyyliominaisuudet, joita voi muokata. Tyylisäännöillä voidaan asettaa esimerkiksi elementin sisältämälle tekstille lihavointi, väri, koko, alleviivauksen tai piirtää elementille reunat. Elementtien sijainti sivustolla myös tehdään tyylisäännöillä. (Mozilla, 2024b)

3.3.3 SASS

SASS (Syntatically Awesome StyleSheet) on CSS-tyylitiedostokielen laajennus, jonka avulla voidaan vähentää tyylisääntöjen kokoa CSS-syntaksin toistoa vähentämällä. Se mahdollistaa erilaisten apuvälineiden kuten sisäkkäisten sääntöjen, tyyliluokkien perinnän, muuttujien, mixin-funktioiden ja import-lausekkeiden avulla. SASS on ilmainen laajennus, sekä se on yhteensopiva kaikkien CSS-versioiden kanssa. (W3Schools, n.d.)

SASS tiedostojen nimet sisältävät .scss-päätteen ja verkkoselaimet eivät pysty tulkitsemaan niitä sellaisenaan. SASS-tiedostot käännetään CSS-tiedostoiksi erillisen kääntäjän avulla. Angular CLI (Command-Line Interface) tukee SASS-tiedostojen kääntämistä, jos Angular-projektin tyylitiedostot sisältää .scss-päätteen, joita myös LemonOnlinen web-sovellus käyttää.

3.3.4 JavaScript

JavaScript on skriptikieli, jota käytetään yleisimmin verkkosivujen ohjelmointikielenä, mutta sitä voidaan ajaa myös muissa ympäristöissä kuin verkkoselaimessa. Skriptikielet tulkataan ohjelman ajon aikaisesti, eikä niitä käännetä etukäteen. JavaScript tukee olio- ja funktiopohjaista ohjelmointityyliä. Kieli on dynaaminen, eli sen ominaisuuksiin kuuluu mm. ajon aikainen olioiden tarkastelu ja rakentaminen, skriptien luonti ja muuttujiin asettaminen, sekä funktioiden muuttuva parametrilistaus. (Mozilla, 2024c)

JavaScript-tiedostot ovat nimetty .js-päätteellä. LemonOnline web-sovelluksen toiminnallisuus perustuu JavaScriptiin.

3.3.5 TypeScript

TypeScript on ohjelmointikieli, jonka avulla voidaan tuoda staattinen tyypitys JavaScriptiin, parantaen sen heikkoa tyypitystä. Staattisuus tulee siitä, että kieltä ei tulkita ajon aikaisesti, kuten JavaScript, vaan se transpiloidaan ennen ajoa JavaScriptiksi, sillä TypeScriptiä ei voida tulkata sellaisenaan. Transpiloinnissa muunnetaan lähdekoodi ohjelmointikielestä toiseen. Tyypitys auttaa havaitsemaan virheitä ohjelman lähdekoodia kirjoittaessa, vähentäen ajon aikaisia ohjelmavirheitä. Ohjelmointiympäristö myös hyötyy tyypityksestä, parantaen ohjelmoijalle annettavia ehdotuksia ohjelmointityön aikana. (TypeScript, 2024)

Angular CLI transpiloi TypeScript-kääntäjän avulla projektin TypeScriptin tiedostot (.ts-pääte) JavaScript-syntaksin mukaiseksi, poistaen niistä tyypityksen. LemonOnline web-sovelluksen lähdekoodi on kirjoitettu pääosin TypeScriptillä.

3.3.6 Visual Basic .NET

Visual Basic .NET (VB.NET) on Microsoftin kehittämä korkean tason ohjelmointikieli, joka on suunniteltu ohjelmoijalle helposti lähestyttäväksi englannin kieltä mukailevan syntaksinsa avulla. Uusien ohjelmoijien on nopea oppia kieltä hyvän

luettavuuden ja selkeytensä ansiosta. Lisäksi siinä on myös kattavat ominaisuudet kokeneempia varten. Kieli voi olla heikosti tai vahvasti tyypitetty, riippuen kirjoitettavan ohjelman käyttökohteesta ja tarpeista. LemonOnlinen palvelinpään sovellus on kirjoitettu VB.NET-kielellä ja siinä on vahva tyypitys, joka ennaltaehkäisee ohjelmavirheitä ja nopeuttaa ajonaikaista suorituskykyä. Kieli on osa .NET viitekehystä. (Microsoft, 2021)

3.3.7 Transact-SQL

Transact-SQL (T-SQL) on Microsoftin kehittämä laajennus Structured Query Language (SQL) ohjelmointikieleen, jonka avulla voidaan käsitellä relaatiotietokantoja. T-SQL perustuu SQL-kielen standardiin, tuoden siihen lisänä muista ohjelmointikielistä tuttuja ominaisuuksia, kuten muuttujia, silmukoita ja ehtolausekkeita. Sen avulla voidaan myös käsitellä merkkijonoja, päivämääriä, kellonaikoja ja tehdä matemaattisia operaatioita. Koodin toistoa voidaan vähentää ja suorituskykyä lisätä proseduurien tai funktioiden avulla. Myös virheidenkäsittely on mahdollista kielen ominaisuuksilla. (T-SQL Tutorial, n.d.)

3.4 Viitekehykset

3.4.1 Angular

Angular on Googlen kehittämä ja ylläpitämä avoimen lähdekoodin web-viitekehys, jonka avulla voidaan tehdä verkkosovelluksia. Angular tarjoaa valmiita rajapintoja, työkaluja ja kirjastoja helpottaakseen ohjelmoijan työtä. Viitekehys toimii pienille ja suurille verkkosovelluksille ja sen komponenttimalli sekä modulaarisuus mahdollistavat ison kehitystiimin työskentelyn projektin parissa. (Angular.dev, 2025a)

Angular tukee SSR (Server-Side Rendering) ja SSG (Static Site Generation) -tekniikoita, mutta niitä ei toistaiseksi hyödynnetä LemonOnlinessa. Viitekehys on kirjoitettu TypeScript-kielellä, joten myös Angular-sovellukset kuten LemonOnline käyttävät sitä. (Angular.dev, 2025a)

3.4.2 .NET Framework

Microsoftin .NET viitekehyksen avulla voidaan luoda Windows sovelluksia ja web palveluita. Sen tavoitteena on tarjota kehitysympäristö, joka toimii samanlailla riippumatta sovelluksen ajoympäristöstä tai tyypistä ja joka voi kommunikoida muiden sovellusten kanssa noudattaen alan standardeja. Viitekehys koostuu luokkakirjastosta ja ajonaikaisesta käyttöympäristöstä (Common Language Runtime, CLR). CLR hallinnoi sovellusta ajon aikana ja tarjoaa sille palveluja kuten säikeiden ja muistinhallinnan. .NET luokkakirjasto sisältää oliopohjaisen kokoelman CLR:n kanssa yhteensopivia luokkia ja tyypityksiä. (Microsoft, 2023a)

3.4.3 ASP.NET

ASP.NET on osa .NET-viitekehystä ja sitä käytetään verkkosivustojen, -sovellusten ja rajapintojen luontiin. ASP.NET tarjoaa kolme kehitysmallia verkkosovellusten luontiin: Web Pages, Web Forms ja MVC (Model-View-Controller). ASP.NET Web API -kehitysmallilla voidaan tarjota HTTP-palveluita käyttöliittymille, kuten REST-rajapinta. (Microsoft, 2022a)

ASP.NET MVC on kehitysmalli, jonka arkkitehtuurissa sovellus jaetaan kolmeen komponenttiin: malliin, näkymään ja käsittelijään. Malli sisältää logiikan, jonka avulla tietoa käsitellään ja varastoidaan tietokantaan. Näkymä vastaa käyttöliittymästä ja määrittelee, miten mallin tarjoama tieto näytetään käyttäjälle. Käsittelijä ottaa vastaan käyttäjän suorittamat toiminnot, toteuttaa ne mallin avulla ja päivittää tiedon näkymään. Arkkitehtuurin hyötynä on eri komponenttien irrallisuus toisistaan, joka helpottaa testattavuutta, selkeyttää rakennetta ja vähentää kompleksisuutta. (Microsoft, 2023b)

LemonOnline käyttää ASP.NET MVC-kehitysmallin näkymänä Angular-sovellusta, jonka alustava rakenne luodaan kehitysmallissa ja tarjotaan resursseineen web-palvelimelta. Teknisesti Angular-sovellus käännetään ja sovelluskoodi kopioidaan paikkaan web-palvelimella, missä siihen voidaan viitata ASP.NET MVC:n näkymän

HTML-dokumentin skripteissä. Malli sisältää sovelluksen liiketoimintalogiikan. ASP.NET Web API toteuttaa rajapinnan ja on kehitysmallissa käsittelijä.

ASP.NET SignalR -kirjaston avulla voidaan tuoda reaaliaikaista toiminnallisuutta web-sovelluksiin. Kirjasto mahdollistaa tapahtuman lähetyksen palvelimelta asiakassovellukseen, esimerkiksi tiedon muuttuessa palvelimen puolella. Näin asiakassovelluksessa ei ole tarpeen tehdä jatkuvaa kyselyä palvelimelta, onko tieto muuttunut. Tämä nopeuttaa tiedon saamista, sekä vähentää kyselyiden määrää. (Microsoft, 2020)

LemonOnline hyödyntää SignalR:ää web-sovelluksessa esimerkiksi sivuilla, joilla voi olla useampia käyttäjiä samanaikaisesti tarkastelemassa tai muokkaamassa samaa tietoa. SignalR:n avulla voidaan lähettää reaaliaikaisesti viesti käyttäjille tiedon muuttuessa.

3.4.4 NgRx

NgRx on viitekehys Angular-sovelluksiin. NgRx tarjoaa kirjastoja erilaisiin tarpeisiin, kuten sovelluksen tilan hallintaan, Angular Routerin integraatioon ja entiteettikokoelmien hallintaan. Lisäksi se tarjoaa kehittäjän työkaluja virheenjäljitystä ja koodin tarkistusta varten. NgRx:n käyttö mahdollistaa siistin sovellusarkkitehtuurin, jossa sovelluksen tila on saatavilla eri komponentteihin keskitetyssä paikassa, helpottaen sen hallintaa. LemonOnlinen web-sovellus hyödyntää NgRx:n kirjastoja sovelluksen tilan hallintaan, kuten kirjautuneen käyttäjän tietojen pitämiseen sovelluksen muistissa. (NgRx.io, 2025a)

3.5 Microsoft SQL Server

Microsoft SQL Server on relaatiotietokannan hallintajärjestelmä (RDBMS), joka voidaan asentaa Windows tai Linux käyttöjärjestelmään. SQL Server tarjoaa palveluita kuten koneoppimista, integraatioita, analyysejä, raportointia ja replikointia. (Microsoft, 2024a)

SQL Serverin ytimenä toimii tietokantamoottori (Database Engine), joka prosessoi, varastoi ja turvaa tietoa. Tietokantamoottoreita voi olla käytössä useampia, jossa yksittäinen instanssi hallinnoi yhtä tai useampaa tietokantaa. Sovellus tai työkalu voi ottaa yhteyden tietokantamoottorin instanssiin tai tietokantaan, jonka avulla suorittaa tietokantaoperaatioita käyttäen T-SQL-ohjelmointikieltä. (Microsoft, 2022b).

LemonOnline käyttää tietokantaratkaisunaan SQL Serveriä ja tukee sen kolmea uusinta versiota.

4 PILVISIIRTYMÄ

4.1 Yleistä Azuresta

Microsoft Azure on julkinen pilvipalvelu, joka tarjoaa useisiin eri käyttötarkoituksiin valikoiman ratkaisuita, kuten alustapalveluita, virtuaalipalvelimia, tallennustilaa, tietokantoja, analytiikkaa ja tekoälyä. Palvelut voidaan jakaa neljään pääkategoriaan, joita ovat IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) ja serverless computing. (Microsoft, 2025a)

IaaS-palvelumallissa pilvipalveluntarjoaja lainaa IT-infrastruktuurin käyttäjän hyödynnettäväksi. Infrastruktuuriin lukeutuu mm. palvelimet, tietoliikenne ja tallennustila. IaaS-palveluiden avulla voidaan ulkoistaa oman palvelinsalin toiminnot, poistaen laitehankinnat, ylläpidon, henkilöstö- ja muut kustannukset. Sen sijaan käyttäjä maksaa vain pilvipalveluista resurssikäytön mukaan. Palveluita voidaan skaalata helposti tarpeen mukaan nostamalla ja laskemalla resurssikapasiteettia. (Microsoft, 2025b)

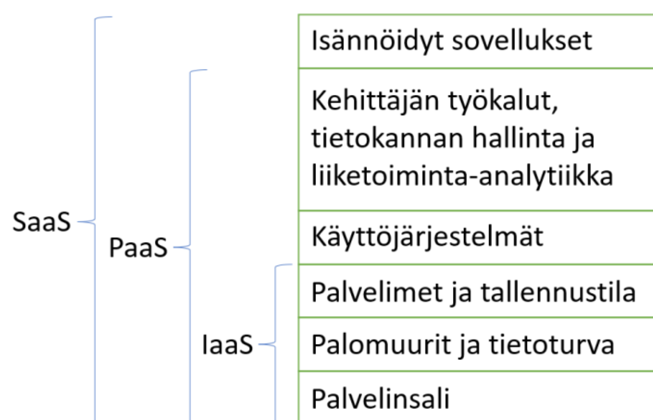
PaaS tarjoaa infrastruktuurin lisäksi käyttöjärjestelmän, kehittäjän työkalut, tietokannan hallintajärjestelmän ja analytiikan. Palvelumalli tukee verkkosovellusten kehityksen koko elinkaarta, kuten lähdekoodin kääntäminen, testaus, käyttöönotto ja päivittäminen. Sen etuina on sovellusten kehittämisen nopeutuminen, hankittavien ja ylläpidettävien ohjelmistolisenssien vähentyminen, sekä kehittyneiden BI- ja analytiikkaratkaisuiden käyttövalmius. (Microsoft, 2025c)

SaaS-palvelumallissa pilvipalveluntarjoaja tuottaa kaiken aiemmin mainitun lisäksi sovelluksen, sen ylläpidon ja päivitykset valmiina pakettina käyttäjälle. Sovellusta käytetään verkkosovelluksena internetin yli erilaisilla käyttäjän päätelaitteilla. (Microsoft, 2025a)

Serverless computing -malli on samankaltainen kuin PaaS, mutta siinä kehittäjälle jää vain sovelluksen lähdekoodin tuottaminen, sillä palvelu tarjoaa lähdekoodin

ajoon vaaditun infrastruktuurin, joka skaalautuu dynaamisesti taustalla resurssikäytön tarpeen muuttuessa. Näin kehittäjä voi keskittyä enemmän liiketoimintalogiikkaan ja arvo tuottamiseen. Palvelumalli mahdollistaa loppukäyttäjille sovelluksen korkean saatavuuden riippumatta käyttäjämääristä, kehittäjien tuottavuuden parantamisen ja tuotteiden nopeamman tuomisen markkinoille. (Microsoft, 2025d)

Kuvio 3 havainnollistaa IaaS, PaaS ja SaaS palvelumallien eron ja niiden suhteen toisiinsa.



Kuvio 3. IaaS, PaaS ja SaaS palvelumallien vertailu. (Microsoft, 2025b)

4.2 Azuren palvelut

Azuresta löytyy useita palveluita, joita voidaan hyödyntää LemonOnlinen pilviarkkitehtuurissa. Kappaleessa käydään läpi opinnäytetyön kannalta niistä merkityksellisimmät.

4.2.1 Azure App Service

Azure App Service on PaaS-mallin pilvipalvelu. Sen avulla voidaan isännöidä verkkosovelluksia ja REST-rajapintoja. Se tukee ensisijaisesti .NET-, .NET Core-, Node.js-, Python-, Java- ja PHP-sovelluksia. Myös taustaprosessien lisääminen on mahdollista Powershell- ja muilla skripteillä, tai suoritettavilla tiedostoilla. Palvelu mahdollistaa sovelluksen kuormituksen lisääntyessä automaattisen käytössä olevien

laskentaresurssien skaalauksen ja kuormituksen tasauksen. Azure App Servicen tarjoama ympäristö voi olla käyttötarpeen mukaan Windows tai Linux -pohjainen, jota palvelu ylläpitää ja päivittää automaattisesti. Palvelu integroituu Visual Studio ja Visual Studio Code -kehitysympäristöihin, mikä suoraviivaistaa sovelluksien luontia, vianetsintää ja käyttöönottoa. Siinä on myös sisäänrakennettu käyttäjien autentikointimahdollisuus Microsoft-, Microsoft Entra ID-, Twitter-, Facebook- tai Google-tiliä käyttäen. (Microsoft, 2024b)

Azure App Servicen kustannukset tulevat palvelun laskentaresurssien käytön mukaisesti. Muita mahdollisia kuluja voi tulla kustomoiduista verkkotunnuksista, SSL (Secure Sockets Layer) -sertifikaateista ja -yhteyksistä. Palvelulle valitaan käyttötarpeen mukaan sopiva ohjelma, sekä palvelutaso. Ohjelmavaihtoehdot ovat Free, Shared, Basic, Premium ja Environment, joista Free ja Shared ovat palvelun kokeilua ja harjoittelua varten. Basic-ohjelma on edullinen, pienille verkkoliikennemäärille suunnattu vaihtoehto, joka ei tarvitse automaattista skaalausta tai verkkoliikenteen hallintaominaisuuksia. Premium-ohjelma tarjoaa runsaasti suorituskykyä, skaalautuvuutta ja luotettavuutta uusimmilla ominaisuuksilla. Environment-ohjelma on ratkaisu erityisiin tarpeisiin, joihin vaaditaan täysin eristetty ympäristö, joka on tietoturvallinen, skaalautuva ja yhdistyy suoraan asiakkaan virtuaaliverkkoon. Basic-, Premium-, ja Environment-ohjelma voidaan valita käytössä olevat laskentaresurssit, kuten suoritinytimien, RAM-muistin ja tallennustilan määrän. Hinta kasvaa sen mukaan, mitä enemmän resursseja on valittu käyttöön. Laskentaresurssien kulutus muodostuu palvelussa isännöidyn sovelluksen tarpeiden ja käytön perusteella, joten LemonOnlinen resurssikäyttöä analysoitaessa tulee kiinnittää huomiota mitkä asiat vaikuttavat kulutukseen. (Microsoft, 2025f)

4.2.2 Azure SQL Database ja Elastic pool

Microsoft tarjoaa SQL Serverin pilvivastineeksi Azure SQL Database -tietokantapalvelua PaaS-mallisena, jossa on uusin vakaa versio SQL Serverin tietokantamoottorista, sekä palvelun hallinnoima infrastruktuuri ja käyttöjärjestelmä. Palvelu lupaa 99.99 % saatavuuden, jota voi parantaa esimerkiksi vikasietoryhmillä (eng. failover

groups), joiden avulla voidaan replikoida tietokantoja toiseen Azuren palvelinsaliin eri maantieteellisellä alueella ja aktivoida kuormantasaus tietokantojen välillä. Verkkoliikenne voidaan tarpeen tullen ohjata toiseen alueeseen, jos esimerkiksi yhteen palvelinsaliin tulee häiriöitä. Palvelu tekee myös automaattiset varmuuskopioinnit tietokannoille, joista voidaan palauttaa aiempi tietokannan tila. (Microsoft, 2024c)

Azure SQL Database -palveluun on sisäänrakennettu älykkäitä toimintoja, kuten Intelligent Insights, joka tarkkailee tietokannan suorituskykyä ja kertoo sen huonontumisesta parannusehdotuksineen. Automaattiviritys (eng. automatic tuning) -ominaisuus tarkkailee kyselyitä, joita suoritetaan tietokantaan ja tekee tietokantaan muutoksia suosituksien perusteella, sekä testaa ja varmentaa muutokset parantaen tietokannan suorituskykyä automaattisesti. Microsoft Defender for SQL tarjoaa tietoturvaominaisuuksia, kuten haavoittuvuusarvioinnin, jonka tulosten ja ehdotusten perusteella voidaan korjata haavoittuvuuksia tietokannassa. Tietokantatapahtumia ja kyselyitä voidaan myös tarkkailla automaattisesti ja hälyttää uhkatilanteista ja saada tietoa, kuinka niihin kannattaa reagoida. (Microsoft, 2024c)

Azure SQL Database tukee yksittäisiä tietokantoja, sekä elastic pool -mallisen joukon tietokantoja, jotka jakavat saman palvelimen resurssit keskenään. Elastic pool sopii tilanteisiin, joissa tietokantoja on useita ja niiden käyttö on vaihtelevaa ja ennalta-arvaamatonta. Jotkut tietokannat eivät käytä juuri lainkaan palvelinresursseja ollessaan tyhjäkäynnillä, kun taas samaan aikaan toista tietokantaa vasten voidaan suorittaa raskaita operaatioita ja palvelinresursseja voidaan hyödyntää siihen. Näin palvelinresursseja, kuten suorittimia ja muistia ei tarvitse varata yksittäisille tietokannoille ja maksaa niistä turhaan tyhjäkäynnin aikana, tuoden kustannussäästöjä. Elastic poolissa oleville tietokannoille voidaan myös asettaa käytettävät minimi- ja maksimiresurssit, ettei yksi tietokanta voi viedä palvelimen kaikkia resursseja, heikentäen muiden palvelimella olevien tietokantojen suorituskykyä. (Microsoft, 2024d)

Elastic pool -palvelun kustannukset riippuvat käytetystä palvelutasosta. Palvelutasoja on saatavilla vCore- ja DTU (Database Transaction Unit) -pohjaisella mallilla. vCore-mallissa voidaan itse valita käytössä olevat resurssit, kuten suoritinten ja muistin lukumäärä. DTU-mallissa käytetään eDTU-yksikköjä (elastic Database Transaction Unit), johon on niputettu tietty määrä eri resursseja: prosessorikäyttöä, muistia ja tietokantaan luku- ja kirjoitusoperaatioita. vCore-mallissa palvelutasoja on General Purpose, Business Critical ja Hyperscale. General Purpose on yleisiin käyttötapauksiin sopiva vaihtoehto, kun kuluissa halutaan säästää. Business Critical -palvelutaso sopii liiketoimintakriittisiin pienen viiveen ja suurien kyselymäärien tapauksiin, jossa myös tietokantojen vikasietoisuus voidaan varmistaa replikoiden avulla. Replikointi on myös mahdollista Hyperscale-palvelutasossa, joka on suunnattu yrityskäyttöön ja on ominaisuuksiltaan erittäin suorituskykyinen vaihtoehto, jossa voidaan säätää varastointi ja laskentaresursseja joustavammin kuin muissa tasoissa. vCore-palvelumallissa on mahdollista ottaa käyttöön myös serverless computing -maksumalli, jossa palvelu säätää käytössä olevia laskentaresursseja automaattisesti kuormituksen muuttuessa. DTU-pohjaisessa mallissa on saatavilla kaksi palvelutasoa. Standard-taso on yleisimpiin tapauksiin ja pieniin budjetteihin tarkoitettu. Premium-taso on suunniteltu suurille tapahtumamäärille ja pienen viiveen tarpeisiin, sekä korkeaa vikasietoisuutta vaativiin tilanteisiin. (Microsoft, 2025g)

Sopiva palvelumalli ja taso tulee valita käyttötarpeen mukaan. Riippumatta valinnoista, yhteistä hinnoittelussa on se, että yleisesti ottaen maksu muodostuu laskentaresurssien ja tallennustilan käytön perusteella. Laskentaresursseja on tarjolla useita, joissa suorituskyky ja hinta vaihtelee. Osa palvelumalleista tukee erilaisia varmuuskopiointimahdollisuuksia. Säästösuunnitelmilla voidaan varata resursseja yhden tai kolmen vuoden ajaksi ja saada alennusta. Jos palvelun ostajalla on käytössään SQL Server lisenssejä, saa niillä alennusta pilvikustannuksista Azure Hybrid Benefit -ohjelmalla. Opinnäytetyössä ei oteta kantaa siihen, kuinka Azuren palveluita tulisi käyttää mahdollisimman kustannustehokkaasti, vaan miten LemonOnlinea voidaan kehittää kustannustehokkaammaksi. Azure SQL Database Elastic

pools kuitenkin soveltuu parhaiten LemonOnlinen käyttöön Azuren tietokantapalveluista. Kustannustehokkuutta ajatellen parhaat keinot tietokantoihin liittyen on löytää keinoja, joilla tietokantojen kokoa voidaan pienentää, sekä suoritin- ja muistinkäyttöä vähentää. (Microsoft, 2025g)

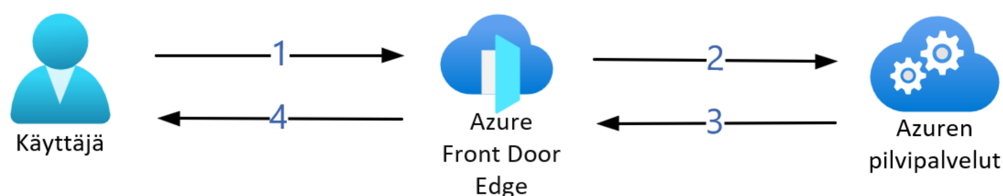
4.2.3 Azure Front Door

Azure Front Door -palvelu on CDN (Content Delivery Network), jonka avulla voidaan tuoda dynaaminen ja staattinen verkkosisältö käyttäjän lähelle, hyödyntäen Microsoftin globaalia reunaverkkoa (eng. edge network). Verkkoon kuuluu satoja PoP (Point of Presence) -asemia globaalisti, joista käyttäjää lähimpänä olevia voidaan hyödyntää verkkosisällön nopeaan ja turvalliseen toimitukseen. Maantieteellisesti hajautettujen PoP-asemien ansiosta palvelulla on korkea saatavuus. (Microsoft, 2024e)

Palvelu toimii julkisivuna muille palveluilla ja on käyttäjälle päin ensimmäinen kontakti, joka reitittää verkkoliikenteen muihin palveluihin ja tekee tarvittaessa kuormantasausta. Verkkosisällön toimitusta voidaan nopeuttaa erilaisilla ominaisuuksilla, kuten välivarastointi (caching). Palvelussa on sisäänrakennettu OSI-mallin tasoilla 3-4 toimiva suojaus hajautettuja palvelunestohyökkäyksiä (DDoS, Distributed Denial of Service) vastaan. Siihen voi myös kytkeä WAF (Web Application Firewall) -ominaisuuden, joka antaa suojaa verkkohyökkäyksiltä ja OSI-mallin tason 7 DDoS-suojauksen. Verkkoliikennettä Front Door -palveluun voidaan monitoroida reaaliajassa tai analysoida sisäänrakennettujen raporttien avulla selainkäyttöliittymästä. (Microsoft, 2024e)

Azure Front Door -palvelun hinnoittelumalli koostuu useasta osasta. Ohjelmavaihtoehdot ovat Standard ja Premium. Palvelun käytöstä veloitetaan jokaisesta tunnista, kun se on päällä. Premium-ohjelman juokseva kustannus on noin 10 kertaa kalliimpi kuin Standard-ohjelman. (Microsoft, 2025h)

Muita kuluja koituu verkkoliikenteestä käyttäjän, Azure Front Door -palvelun reunan (eng. Edge), sekä Azuren muiden palvelujen välillä. Jokaisesta 10 000 verkkopyynnöstä asiakkaalta Azure Front Door-palveluun reunaan veloitetaan, jossa hinta riippuu siitä, missä alueella pyyntö tapahtui. Standard-ohjelmassa hinta on pienempi kuin Premium-ohjelmassa, mutta Premiumissa WAF ja Private Link -ominaisuudet ovat ilmaisia. Kuviossa 4 esitetään hinnoitteluperusteet, jossa kohta 1 on verkkopyyntöjen lukumäärä. (Microsoft, 2025h)



Kuvio 4. Azure Front Door -palvelun hinnoittelumalli. (Microsoft, 2023c)

Verkkoliikenne Front Door -palvelun reunasta Azuren palveluihin (kohta 2) maksaa verkkoliikenteen tiedonsiirtomäärien mukaan, jokaisesta gigatavusta. Jos Front Door -palveluun on otettu käyttöön välivarastointi ja sen avulla toimitetaan verkkosisältö suoraan käyttäjälle reunalta, tätä kustannusta ei ole. Azuren pilvipalvelusta Front Door -palvelun reunaan palautuva vastaus (kohta 3) on ilmainen, kun taas siitä käyttäjälle tuleva liikenne (kohta 4) on maksullista siirrettyjen gigatavujen lukumäärän perusteella. Gigatavuihin perustuvan veloituksen määrä riippuu maantieteellisestä alueesta, jossa tiedonsiirto tapahtuu, niiden ollessa samat Standard- ja Premium-ohjelmissa. (Microsoft, 2023c)

4.2.4 Azure Cache for Redis

Azure Cache for Redis on Microsoftin hallinnoima pilvipalvelu, johon voidaan varastoida tietoa ja jakaa sitä sovelluksille suorituskykyisesti isoja määriä pienillä vasteajoilla. Se pohjautuu Redis-ohjelmistoon, jota käytetään välivarastona erityyppisen tiedon tallentamiseen ja lukemiseen palvelimen muistista. Se voidaan ottaa käyttöön itsenäisenä palveluna (eng. standalone) tai jonkin toisen Azuren tietokantapalvelun rinnalle. (Microsoft, 2024f)

Azure Cache for Redis -palvelua käyttämällä voidaan tehostaa sovelluksen suorituskykyä erilaisilla yleisesti käytetyillä sovellusarkkitehtuurin ratkaisuilla. Esimerkiksi verkkosovelluksen käyttäjän istuntotietoja voidaan pitää palvelun muistissa, mistä sitä voidaan käyttää nopeammin kuin esimerkiksi relaatiotietokannasta. Verkkosovelluksen harvoin muuttuva staattinen sisältö voidaan tarjota palvelusta, vähentäen varsinaisen verkkopalvelimen kuormitusta. Redis voi myös toimia tietokannan rinnalla niin, että sinne tallennetaan osa samasta tiedosta mitä tietokannassa käsitellään. Näin usein pyydettävää tietoa voidaan tarjota nopeammin välimuistista kaikille käyttäjille, kuin tietokannasta hakemalla. Tiedon tyyppi, jota välivarastoidaan, on ennalta määriteltyä ja joka soveltuu välimuistissa pidettäväksi. Välimuistia päivitetään ja tyhjätään erilaisten sääntöjen mukaan, esimerkiksi tietyn ajan kuluttua. (Microsoft, 2024f)

Azure Cache for Redis on saatavilla Basic, Standard, Premium, Enterprise ja Enterprise Flash -ohjelmina. Basic-ohjelmaa käytetään kehitysympäristöissä ja palvelun testaamisessa. Standard-ohjelma on tarkoitettu tuotantoympäristöihin ja se voi saavuttaa tiedon replikoinnin avulla 99.9% saatavuuden. Premium-ohjelma tarjoaa Standard-ohjelmaa paremman suorituskyvyn, saatavuuden, pienemmän verkkoviiveen ja lisäominaisuuksia. Aiemmin mainitut ohjelmat käyttävät avoimen lähdekoodin Redis-versiota. Näiden lisäksi ohjelmaksi voidaan valita Enterprise tai Enterprise Flash, jotka on toteutettu suljetun lähdekoodin Redis Enterprise teknologialla. Ne sisältävät erikoisominaisuuksia verrattuna avoimen lähdekoodin versioon, parhaimman saatavuuden (99.999%), isoimmat tallennuskapasiteettivaihtoehdot, sekä eniten yhtäaikaisia asiakkaiden verkkoyhteyksiä. (Microsoft, 2025i)

Kun sopiva ohjelma ominaisuuksiltaan on päätetty, sille on valittavissa myös eri palvelutaso. Palvelutasot eroavat toisistaan välimuistin koossa, verkon suorituskyvyssä/nopeudessa, sekä aktiivisten verkkoyhteyksien lukumäärässä asiakassovelluksiin. Parempi palvelutaso nostaa myös kustannuksia, joten tasoa säädetään tarpeen mukaisesti pitäen kulut mahdollisimman pieninä. (Microsoft, 2025i)

4.2.5 Azure SignalR

Azure SignalR on palvelu, jota käytetään reaaliaikaiseen viestintään palvelinlähtöisesti asiakassovellukseen, kuten verkko- tai mobiilisovellukseen. Se poistaa siis tarpeen tehdä jatkuvaa kyselyä (eng. polling) asiakassovelluksesta palvelinsovellukseen, onko esimerkiksi jokin tieto muuttunut palvelimen päässä. Viestinnässä käytetään ensisijaisesti WebSocket-protokollaa, mutta se voi myös vaihtoehtoisesti hyödyntää Server-Sent Events tai Long Polling -tekniikoita, jos palvelin tai päätelaite ei tue sitä. SignalR:n käyttökohteita ovat ohjelmat, jotka vaativat reaaliaikaista viestintää päätelaitteisiin tiedon muutoksista. Näitä ovat mm. chat-sovellukset, yhteistyösovellukset monella käyttäjällä, kohteiden GPS-seuranta reaaliajassa ja push-notifikaatiot. (Microsoft, 2023d)

Azure SignalR palvelua voidaan käyttää helposti palvelin-, sekä asiakassovelluksessa sille saatavien JavaScript, ASP.NET Core ja ASP.NET -ohjelmistokirjastojen avulla. Sitä voidaan myös ohjata REST-rajapinnan, Azure Function-palvelun tai Event Grid -integraation kautta. Azure SignalR -palvelun etu perinteiseen itse hallinnoitavaan SignalR-ohjelmistoon on automaattinen skaalautuvuus miljooniin yhteyksiin, tietoturva, sekä hyvä Azure-standardeihin ja palveluihin sopivuus. (Microsoft, 2023d)

Azure SignalR:n palvelutasot ovat Free, Standard ja Premium. Free-taso on ilmainen, mutta se soveltuu lähinnä vain palvelun testaamiseen, koska sen käyttö on rajoitettu maksimissaan 20 samanaikaiseen yhteyteen ja 20 tuhanteen viestiin päivässä. Standard-tasolle luvataan 99.9% ja Premium-tasolle 99.95% Service Level Agreement (SLA), eli palvelun saatavuus. Standard-taso skaalautuu 100:n ja Premium 1000:n SignalR-yksikköön, joista jokaiseen voi olla 1000 client-sovellusta yhteydessä samanaikaisesti. Jokaisesta käytössä olevasta yksiköstä koituu kustannuksia päivittäin. Ensimmäiset miljoona viestiä client-sovelluksiin päivittäin ovat ilmaisia, jonka jälkeen ne maksavat noin euron per miljoona viestiä. (Microsoft, 2025j)

4.2.6 Azure Monitor

Azure Monitor -palvelun avulla voidaan kerätä tietoa omasta palvelinympäristöstä tai pilvestä, analysoida sitä ja suorittaa toimenpiteitä saadun tiedon perusteella. Palvelun monitoroi sovelluksia, tietokantoja, käyttöjärjestelmiä, verkkoliikennettä, virtuaalitietokoneita tai mukautettuja lähteitä. (Microsoft, 2024g)

Palvelu kerää tietoa halutuista järjestelmistä. Tieto voi kertoa esimerkiksi sovelluksen suorituskyvystä ja tapahtumista, SQL-palvelimen kuormituksesta, konteista Azure Kubernetes -palvelussa, Azuren resurssien tapahtumista ja tilasta tai käyttöjärjestelmästä, jossa sovellus on käynnissä. Tietoa voidaan myös kerätä omasta mukautetusta lähteestä Azure Monitor -palveluun REST- tai Data Collection -rajapinnan kautta. Tieto tallennetaan tiedon tyypin perusteella omiin säilöihinsä, joita ovat jäljet (eng. traces), lokitiedot, metriikka ja muutokset. (Microsoft, 2024g)

Azure Monitor -palvelun Insights-ominaisuuksilla voidaan vähäisellä konfiguraatiolla ottaa vastaan tietoa resursseista, suodattaa ja visualisoida sitä. Application Insights -ominaisuus monitoroi verkkosovelluksen käyttöä, suorituskykyä ja saatavuutta. Lähteistä saatu tieto on mahdollista visualisoida Azure portal verkkosivustossa halutussa näkymässä erilaisien kuvaajien ja taulukoiden avulla. Tiedon analysointia varten on Azureen sisäänrakennettu erilaisia työkaluja: Log Analytics, Metrics explorer ja Change Analysis. Log Analytics käyttöliittymällä voidaan selata tai suodattaa tietoa halutuilla parametreilla kyselyjen avulla. Kyselyitä voidaan myös tallentaa ja käyttää visualisointiin tai hälytyksiin. Metric explorer auttaa kaavioiden tuottamisessa, metriikan arvojen piikkisten nousujen ja laskujen tutkimisessa, sekä trendien ja korrelaatioiden havainnoimisessa. Change Analysis -ominaisuus tarkkailee muutoksia Azuren resursseissa subscription-tasolla ja kertoo tapahtumista, kuten katkoksista, komponenttien vikatilanteista ja sivuston ongelmista. (Microsoft, 2024g)

Jos kriittisiä tapahtumia huomataan resursseissa, Azure Monitor Alerts-ominaisuudella voidaan asettaa sähköposti- tai tekstiviestihälytys. Hälytys voidaan asettaa ennalta määriteltyjen sääntöjen perusteella, kuten raja-arvojen ylittyessä, tai tietynlaisen lokiviestin vastaanottamisesta. Autoscale-ominaisuus säätää dynaamisesti resursseja niiden kuormituksesta saatavien tietojen perusteella, parantaen suorituskykyä ja tehden kustannussäästöjä. (Microsoft, 2024g)

4.3 LemonOnline Azuressa

4.3.1 Pääkomponenttien siirto pilvipalveluihin

Luvussa 3 käsiteltiin LemonOnlinen nykyarkkitehtuuria, jonka komponentit tulisi siirtää Azuren pilvipalveluihin järkevällä tavalla. Käyttäjähallinta-mikropalvelu on jo rakennettu Azuren pilvialustalle, eikä kuulu opinnäytetyön laajuuteen.

Monoliittisten ASP.NET Core -verkkosovellusten isännöintiä Microsoft Learn suosittelee Azure App Servicessä. Sen etuina on mm. PaaS-malli, skaalautuminen isoille liikennemäärille, sekä sisäänrakennettu kuormantasaus. Nykyinen LemonOnlinen ASP.NET MVC-mallinen verkkosovellus sopii teknologiansa puolesta palveluun, mutta joitain ominaisuuksia voidaan joutua refaktoroimaan käyttämään erillisiä pilvipalveluita. (Microsoft, 2022c)

Lemonsoftin eri versioita varten voidaan luoda oma App Service -resurssi, jota voi käyttää useampi asiakas. Azure App Service mahdollistaa laskentaresurssien, kuten virtuaalisten suorittimien, keskusmuistin ja levykoon lisäämisen tarvittaessa manuaalisesti, jota kutsutaan ylöspäin skaalaamiseksi (eng. scale up). Palvelua on mahdollista skaalata myös ulospäin (eng. scale out), missä kasvatetaan palvelun käyttämien virtuaalikoneiden instanssien lukumäärää joko manuaalisesti tai automaattisesti palvelun kuormituksen kasvaessa. (Microsoft, 2024h)

Nykyistä SQL-tietokantaa varten Azuressa järkevintä on käyttää Azure SQL Database -palvelua, jonka Elastic pools -ominaisuuden avulla voidaan luoda satoja tietokantoja samaan loogiseen SQL-serveriin. Elastic pool rajoittaa tietokantojen

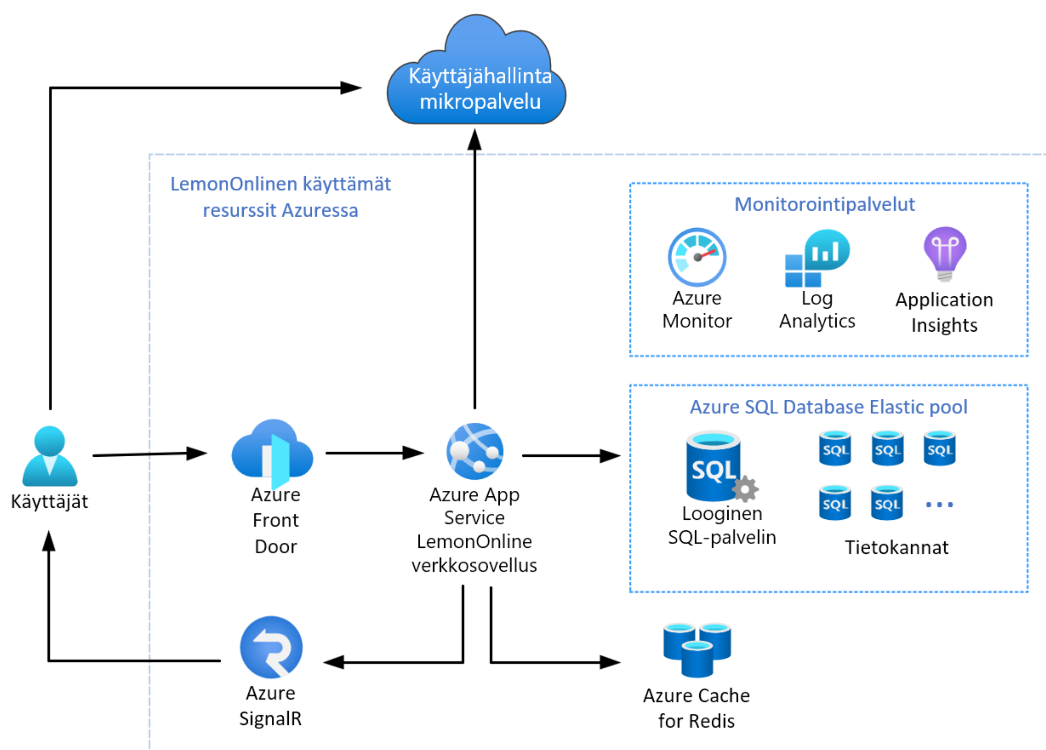
maksimimäärän, riippuen palvelutasosta, mutta tarvittaessa Azure SQL Database -instansseja voidaan luoda lisää asiakas- ja tietokantamäärien kasvaessa. Palvelun avulla saadaan kustannussäästöjä, kun tietokantojen kuormitus on vaihtelevaa ja palvelun laskentaresurssit voidaan jakaa niiden kesken. (Microsoft, 2024d)

Azure Cache for Redis -palvelua hyödynnetään tiedon välivarastointiin, josta usein kysytty ja harvoin muuttumaton tieto saadaan nopeammin ja kustannustehokkaammin kuin SQL-tietokannasta. Azure Cache for Redis -palvelun avulla voidaan myös saavuttaa kustannushyötyjä, kun App Serviceä skaalataan ja useampi instanssi voi käyttää saman Redis-palvelun tietoja, verrattuna jokaisen instanssin omaan palvelimen muistia käyttävään välivarastoon. (Microsoft, 2024f)

LemonOnline nykytilassa käyttää itse hallinnoitavaa ASP.NET-viitekehyksen SignalR-kirjastoa, joten pilvisiirtymässä olisi järkevää käyttää Azuren SignalR -palvelua, koska se ei vaadi ohjelmallisesti isoja muutoksia. Tästä on myös hyötyä, kun palvelinpään LemonOnline-sovellus isännöidään Azure App Service -palvelussa, joissa App Service -instanssien lukumäärää kasvatetaan, kun palvelua skaalataan isommalle käyttäjämäärälle. Näin kaikki instanssit voivat käyttää samaa Azure SignalR -palvelua, jolloin SignalR:ää ei tarvitse itse hallinnoida jokaisessa App Service -instanssissa erikseen. (Microsoft, 2023d)

4.3.2 Arkkitehtuuri

Pohjana LemonOnlinen pilviarkkitehtuurille voidaan käyttää Microsoft Learn-sivuston tarjoamien arkkitehtuuriesimerkkien perusmallista verkkosovellusta (Microsoft, 2025k). Tämän lisäksi arkkitehtuuriin implementoidaan Azure Front Door, Azure SignalR ja Azure Cache for Redis -palvelut. Kuvio 5 esittää LemonOnlinen pilviarkkitehtuurin, jossa mustat nuolet kuvastavat verkkoliikennettä LemonOnlinen asiakassovelluksen ja muiden komponenttien välillä.

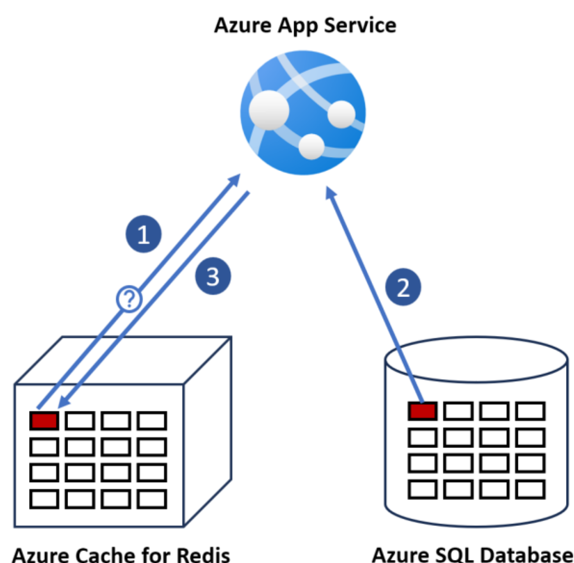


Kuvio 5. LemonOnlinen pilviarkkitehtuuri opinnäytetyössä.

Azure Front Door on ensimmäinen käyttäjää kohdin oleva palvelu, joka ohjaa käyttäjän oikeaan App Service resurssiin, joka sisältää asiakkaan käyttämän version LemonOnlinesta. Verkkoselaimeen ladataan App Servicestä LemonOnlinen Angular-käyttöliittymä, joka tarjotaan palvelinpään sovelluksen ASP.NET MVC-viitekehyksen avulla. Azure Front Doorin WAF-ominaisuus, DDoS-suoja tuovat turvaa mahdollisten verkkohyökkäysten varalta. Sen välivarastoinnilla voidaan myös nopeuttaa verkkosisällön tuottamista käyttäjälle.

Käyttäjän lataama LemonOnlinen web-käyttöliittymä kommunikoi Front Door -palvelun kautta App Servicen palvelinpään sovelluksen kanssa. App Service suorittaa mahdollisesti tietokantaoperaatioita Azure SQL Database -palveluun ja palauttaa tiedon takaisin käyttöliittymälle samaa reittiä kuin se vastaanotettiin. Palvelinpään sovellus on tarpeen mukaan myös yhteydessä käyttäjähallinta-mikropalveluun.

Tietoa voidaan myös välivarastoida Azure Cache for Redis -palveluun, josta se saadaan haettua nopeammin kuin relaatiotietokannasta. Azure Cache for Redis toimii Azure SQL Database palvelun rinnalla cache-aside -arkkitehtuurimallia mukaillen. Kuviossa 6 esitetään mallin toimintaperiaate, jonka ensimmäisessä vaiheessa App Servicen sovellus yrittää etsiä haluttua tietoa Azure Cache for Redis -palvelusta. Jos tieto löytyy, se hyödynnetään, eikä muita vaiheita tehdä.



Kuvio 6. Cache-aside -arkkitehtuurimallin toimintaperiaate. (Microsoft, 2025e)

Jos tietoa ei löydy, se noudetaan Azure SQL tietokannasta (vaihe 2) ja tallennetaan Azure Cache for Redis -palveluun (vaihe 3). Samaa tietoa myöhemmin luettaessa se on saatavilla välivarastosta nopeammin kuin relaatiotietokannasta tehtävällä kyselyllä. Jos tieto muuttuu, se täytyy poistaa välivarastosta, jolloin seuraavan kerran tietoa luettaessa se haetaan samat vaiheet läpi käyden ja tieto päivittyy prosessissa välivarastoon. Azure Cache for Redis -palveluun tallennettava tieto täytyy valita huolellisesti, jotta palvelusta saadaan paras hyöty. Tieto voi olla esimerkiksi harvoin muuttuvaa tai sen haku tietokannasta vaatii suuren määrän laskentaresursseja. (Microsoft, 2025e)

Azure SignalR -palvelua hyödynnetään reaaliaikaiseen tiedon päivitykseen LemonOnlinen käyttöliittymässä. Se viestii asiakassovellusta palvelinlähtöisesti muutoksista esimerkiksi tilanteissa, joissa useampi käyttäjä muokkaa samaa laskua tai tilausta. Azure App Service lähettää tietoa SignalR-palvelulle ennalta määritellyistä tapahtumista. Azure SignalR välittää viestin kaikille yrityksen käyttäjille, jotka käyttävät web-käyttöliittymää sillä hetkellä.

Azure App Serviceä, tietokantoja ja muiden palveluiden toimivuutta tarkkaillaan Azure Monitor -palvelun avulla. Palvelinpään sovellus instrumentoidaan Application Insights -palvelulla, joka lähettää loki- ja diagnostiikkatietoa Log Analytics -palvelulle. Azure SQL Database ja sen Elastic poolissa olevat tietokannat konfiguroidaan lähettämään myös diagnostiikkaa Log Analytics -palveluun.

4.3.3 Kustannusrakenne

LemonOnlinen kustannusrakenne muodostuu sen käyttämien Azuren palveluiden kustannuksista. Kustannuksissa ei oteta huomioon henkilötyöaikaa, eikä muita mahdollisia seikkoja, jotka aiheutuvat muista kuin suoraan Azuren palvelujen käytöstä. LemonOnlinen käyttämät palvelut käytiin läpi kappaleessa 4.3.2, sekä yksittäisten palveluiden kustannukset tunnistettiin kappaleessa 4.2. Palveluiden kokonaiskustannuksia ei pystytä ennalta laskemaan, tietämättä mitä eri palvelutasoja tarvitaan ja minkälaista käyttö tulee olemaan. Ennustetuilla tai arvioiduilla kustannuksilla ei ole kuitenkaan opinnäytetyön tulosten kannalta suurta merkitystä, vaan tärkein on tietää mistä asioista kustannukset aiheutuvat, jotta LemonOnlinea voidaan kehittää kustannustehokkaammaksi.

Opinnäytetyössä ei oteta huomioon Azure Monitor -palvelusta tulevia kustannuksia, sillä ne riippuvat käytön laajuudesta. Sen lukuisat ominaisuudet ovat hyödyllisiä infrastruktuurin tilan ja sovelluksen suorituskyvyn tarkkailuun, mutta eivät ole kuitenkaan pakollisia sovelluksen toimivuuden kannalta. Palvelusta aiheutuvia kustannuksia olisi myös todennäköisesti hankala pienentää, muuta kuin rajoittamalla ominaisuuksien käyttöä.

Kustannukset arkkitehtuurissa muodostuvat yleisellä tasolla seuraavista asioista:

- Front Door -palvelun käynnissäolo-aika
- Käyttäjän verkkoliikenteen kutsujen lukumäärä Azureen
- Käyttäjän ja Azuren välinen tiedonsiirtomäärä gigatavuina
- App Service -palvelun varaamat laskentaresurssit ja virtuaalikoneiden instanssien lukumäärä
- SQL Database -palvelun varaamat laskentaresurssit, tietokantojen käyttämä tallennustila, redundanttisuus ja varmuuskopiot
- Azure Cache for Redis -palvelun välivaraston koko, yksiköiden lukumäärä, valitun verkon nopeuden ja asiakasyhteyksien lukumäärä
- Azure SignalR -palvelun tarvittavien samanaikaisten asiakasyhteyksien, sekä viestin lukumäärä

Tästä voidaan päätellä, että realistisesti kustannuksia voidaan pienentää seuraavin tavoin:

- Vähentämällä käyttäjältä lähtevien kutsujen lukumäärää, eli web-käyttöliittymästä palvelinpäähän lähetettäviä HTTP-kutsuja
- Pienentämällä HTTP-kutsujen kokoa kaikessa verkkoliikenteessä web-käyttöliittymän ja Azuren välillä
- Vähentämällä palvelinpään sovelluksen laskentaresurssien käyttöä App Service -palvelussa
- Vähentämällä SQL serverin laskentaresurssien käyttöä
- Pienentämällä tietokantojen kokoa
- Vähentämällä SignalR:n lähettämien viestien lukumäärää
- Vähentämällä Azure Cache for Redis -palvelussa välivarastoitavan tiedon kokoa tai määrää

SignalR-palvelu tuo käyttäjälle hyödyllisiä lisäominaisuuksia LemonOnlineen, joten sen käyttöä yleisesti ei haluta vähentää. Sen käyttöä voidaan kuitenkin tarkastella

kriittisesti, tuoko se kaikissa tilanteissa tarpeeksi suurta hyötyä käyttäjälle kustannuksiin nähden. On myös riski, että ohjelmointivirheen seurauksena palvelua käytetään väärällä tavalla aiheuttaen ylimääräisiä kustannuksia.

Käyttämällä tiedon välivarastointia Azure Cache for Redis -palvelun avulla voidaan nopeuttaa tiedonhakua ja vähentää tietokantakyselyiden lukumäärää, joten sen käyttö teoriassa pitäisi vähentää kustannuksia. Palvelun käyttöä olisikin syytä lisätä, kunhan se tehdään järkevästi ja hyvien käytäntöjen mukaisesti.

5 OPINNÄYTETYÖPROSESSI

Opinnäytetyön tutkimusvaihetta varten tarvittiin runsaasti esitietoa LemonOnlinen pilviympäristöstä. Ennen tähän vaiheeseen pääsyä suunniteltiin ja rakennettiin Azureen LemonOnlinen pilviarkkitehtuuri, jota käsiteltiin luvussa 4. Tässä luvussa käydään läpi opinnäytetyön tutkimuskysymykset, aiheeseen liittyviä artikkeleita ja julkaisuja, sekä kuvataan tutkimusmenetelmiä.

5.1 Tutkimuskysymykset

LemonOnlinen pilviarkkitehtuuri ja sen kustannusrakenne selvitettiin kappaleessa 4.3. Selvityksestä saatujen tietojen perusteella opinnäytetyön tutkimuskysymykset rajattiin seuraaviin:

- Kuinka verkkopyyntöjen kokoa ja määrää voidaan vähentää web-käyttöliittymän ja Azuren välillä?
- Kuinka App Servicen ja SQL Serverin laskentaresurssien käyttöä voidaan vähentää?

Azure SignalR ja Azure Cache for Redis -palveluita ei tutkita kappaleessa 4.3.3 mainituista syistä, koska niiden käytön vähentäminen huonontaisi sovelluksen käyttökokemusta tai lisäisi kokonaiskustannuksia.

5.2 Julkaisut ja teoriat

Tutkimuskysymyksiin etsittiin vastauksia tutkimusartikkeleista ja verkkojulkaisuista. Opinnäytetyön tekijän omaa työkokemusta LemonOnlinen parissa hyödynnettiin teorioiden soveltuvuuden arvioinnissa, ottaen huomioon tuotteen yksilölliset ominaisuudet. Useimmat artikkelit käsittelevät suorituskykyä, eikä niissä välttämättä huomioida kustannusnäkökulmaa. Vaikkei opinnäytetyön tavoitteena ole sovelluksen nopeuttaminen tai sen avulla saatu käyttökokemuksen parantaminen, niin tekniikat, joilla voidaan vaikuttaa suorituskykyyn ovat kuitenkin useasti sidoksissa myös kustannuksiin. Esimerkiksi jos laskentakapasiteettia lisätään, voidaan

nopeutta parantaa, mutta pilvikustannukset kasvavat. Jos taas nopeutta saadaan vähentämällä laskentaresurssien tarvetta, se pienentää myös kustannuksia. Tavoitteena onkin löytää ne keinot, millä kustannuksia saadaan pienennettyä, ilman että sillä on negatiivisia vaikutuksia sovelluksen käyttökokemukseen.

Shailesh ja Suresh (2017) tekivät analyysin tekniikoista ja laadunarvioinnista web-suorituskyvyn optimointiin. Artikkelin luvussa 3 käydään läpi sääntöjä ja niiden vaikutuksia suorituskykyyn. Yhtenä sääntönä verkkopyyntöjen lukumäärää pienentämällä voidaan vähentää tiedonsiirtomäärää verkossa. Monta listattua menetelmää verkkopyyntöjen optimointiin käytetään LemonOnlinen käännösvaiheessa jo entuudestaan, kuten tiedostojen yhdistämistä, pienentämistä (eng. minification) ja pakkausta (eng. compression). Artikkelissa on myös kuvien optimointiin yleisesti käytettyjä keinoja. LemonOnline sisältää vain vähän staattisia kuvia, mutta dynaamisesti tuotetut kuvat, kuten ostolaskujen liitetiedostot, voisi olla hyödyllistä optimoida jo syöttövaiheessa ennen tallennusta.

Vepsäläinen ja muut (2024) kirjoittivat artikkelin web-sovellusten optimointitekniikoista, jonka kappaleessa 2.6 kerrotaan välimuistin käytöstä. Tuomalla välimuisti lähelle käyttäjää voidaan vähentää palvelimen pään prosessointia ja pienentää viivettä. Välimuisti voidaan myös toteuttaa eri tasoissa, kuten käyttäjä- tai palvelinpäässä, sekä erilaisin tekniikoin. Mertzin ja Nunesin (2017, s. 1–2) mukaan sovellustason välimuistin käyttöä hallitaan koodilla sovelluksen kerrosarkkitehtuurin käyttöliittymä-, palvelu-, liiketoimintalogiikka tai tiedon varastointi -kerroksissa. Sen avulla usein tarvittava tieto voidaan pitää helpommin saatavilla, vähentäen toistuvaa prosessointia ja laskentaa, lisäten web sovelluksen suorituskykyä ja skaalautuvuutta. Ohjelmistokehittäjät voivat päättää itse mitä tietoa välivarastoidaan ja miten välimuistin logiikka toimii, joten sen voi räätälöidä sovellusten yksilöllisiin ominaisuuksiin sopivaksi.

Mertz ja Nunes (2016) tekivät myös kvalitatiivisen tutkimuksen sovellustason välimuistista, jonka tarkoituksena on tarjota parhaita käytäntöjä menetelmän suun-

nitteluun, toteutukseen ja ylläpitoon. Tutkimus keskittyy palvelinpäässä tapahtuvaan tiedon välivarastointiin. Välimuistiin varastoidaan tietoa, jota pyydetään usein tai sen uudelleennoutamiseen tarvitaan paljon laskentaresursseja. Tieto on tyypillisesti varastoitu välimuistijärjestelmään avain-arvo-pareina. Menetelmällä vähennetään kuormitusta tietokantaan tai palveluihin, joita on hankala skaalata ylöspäin isommille käyttäjämäärille. Välimuistin suunnitteluun ja ylläpitoon kuuluu neljä haastetta: mitä tietoa varastoidaan, kuinka se tapahtuu, milloin se kuuluu varastoida tai tyhjentää muistista, sekä missä se kannattaa varastoida. Vaikka sovellustason välimuistia käytetään laajalti, niin se toteutetaan tyypillisesti tapauskohtaisesti sovellukseen sopivaksi tarpeiden mukaan. (Mertz & Nunes, 2016, s. 1)

Mertz ja Nunes (2016, s. 2–3) mukaan cache-aside -arkkitehtuurimallissa tietolähde ja välimuistijärjestelmä eivät ole tietoisia toisistaan. Välimuisti itsessään on passiivinen komponentti, joten sovelluskehittäjien täytyy itse ohjelmoida sen käyttö sovelluksen koodissa. Tähän kuuluu tiedon haku, varastoitavan tiedon nimeäminen ja ylläpito. Siksi välimuistin käytössä on myös huonoja puolia. Koodin toteutus, testaus, vianetsintä ja ylläpito vaativat lisäaikaa kehittäjältä. Toteutus voi vaatia vanhan lähdekoodin uudelleenkirjoitusta ja lisää sen kompleksisuutta, mikä voi olla aikaa vievää ja aiheuttaa ohjelmointivirheitä. Koodissa 1 esitetään Java-ohjelmointikielen esimerkillä sovellustason välimuistin toteutus, jossa ProductsRepository-luokka on vastuussa tuotteiden hausta tietokannasta. Luokan getProducts-metodissa yritetään ensin hakea tuotteet välimuistista. Jos tuotteita ei löydy, ne haetaan tietokannasta, asetetaan välimuistiin ja lopuksi metodi palauttaa tuotteet. Välimuistiin tallennettua tietoa päivitetään updateProduct-metodissa ja poistetaan deleteProduct-metodissa.

```
public class ProductsRepository {
  public List<Product> getProducts() {
    Cache cache = Cache.getInstance();
           List<Product>products = (List<Product>)
                   cache.get("products");
    if (products == null) {
      products = getProductsFromDB();
      cache.put("products", products);
```

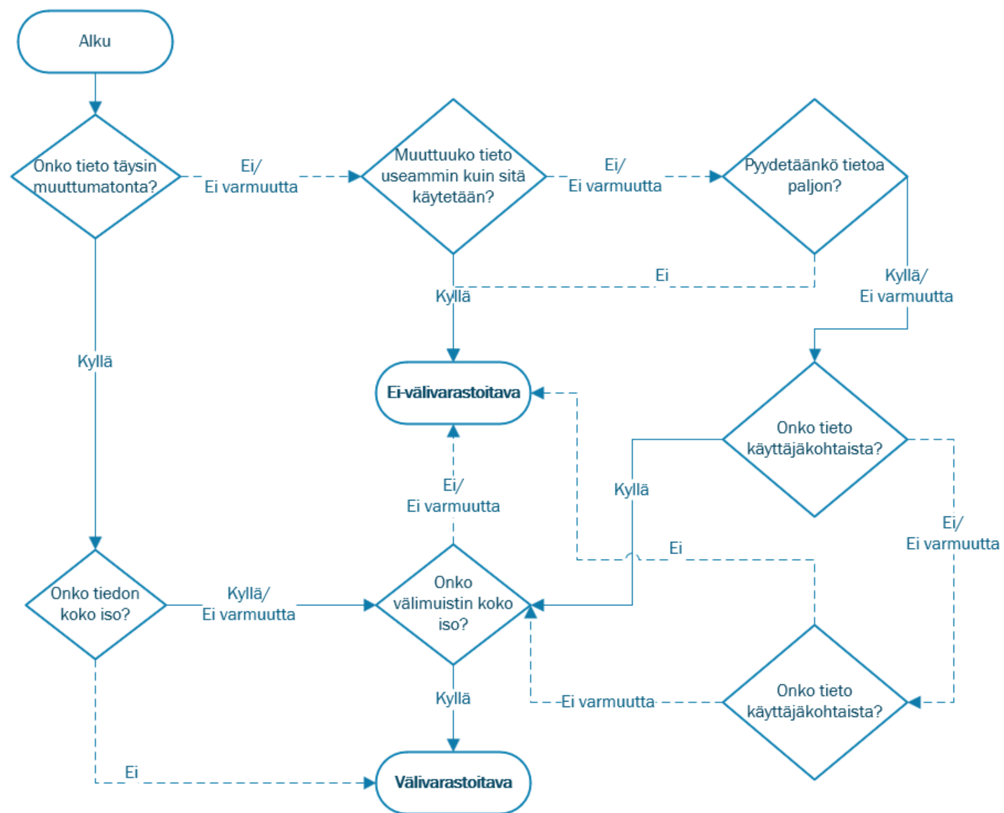
```
    }
    return products;
  }

  public void updateProduct(Product product) {
    Cache.getInstance().delete("products");
    updateProductIntoDB(product);
  }

  public void deleteProduct(Product product) {
    Cache.getInstance().delete("products");
    deleteProductFromDB(product);
  }
}
```

Koodi 1. Esimerkki sovellustason välimuistin toteutuksesta. (Mertz & Nunes, 2016, s. 3)

Tieto kannattaa pitää välimuistissa sovellusarkkitehtuurin siinä tasossa, missä se vähentää eniten prosessointia ja edestakaista tiedon välitystä. Esityskerros (asiakassovellus) on paras vaihtoehto, mikäli mahdollista. Jos tieto soveltuu palvelinpäässä varastoitavaksi, sen voi säilöä palvelu-, liiketoiminta- tai tietokantakerroksissa. Välivarastointia voidaan myös suorittaa useassa kerroksessa samanaikaisesti samalle tiedolle, koska voidaan olettaa, että todennäköisyys välimuistista haetulle tiedolle kasvaa. (Mertz & Nunes, 2016, s. 13.). Apuna päätöksentekoon, mikä tieto soveltuu välivarastoitavaksi, voidaan käyttää kuvion 7 vuokaaviota.



Kuvio 7. Tiedon välivarastoitavuus -vuokaavio. (Mertz & Nunes, 2016, s. 16)

Välimuistin käyttö vaikuttaa tehokkaalta tavalta vähentää ylimääräistä prosessointia LemonOnlinen pilviarkkitehtuurissa. Esimerkiksi teoriassa varastoimalla asiakassovelluksen muistissa harvoin muuttuvaa tietoa, olisi mahdollista vähentää toistuvia verkkopyyntöjä App Servicelle, joka joutuu prosessoimaan kutsun, hakemaan tietoa SQL palvelimelta, joka taas vuorostaan joutuu käyttämään laskentatehoa hakeakseen tiedon tietokannasta. Myös palvelinpäässä voidaan mahdollisesti vähentää ylimääräistä tietokantatason prosessointia, kun usein kysytty tieto voidaan varastoida Azure Cache for Redis -palvelun avulla, josta se voidaan noutaa teoriassa kustannustehokkaammin, kuin relaatiotietokannasta.

Azure Well-Architected Framework on viitekehys, joka auttaa suunnittelemaan laadukkaita pilviratkaisuita Azuressa. Se tähtää parhaisiin arkkitehtuurillisiin ratkaisuihin ja perustuu viiteen tukipylvääseen, joita ovat luotettavuus, turvallisuus, kustannusten optimointi, erinomainen toiminnallisuus ja suorituskyvyn tehokkuus

(Microsoft, 2025k). Kustannusten optimoinnin yksi osa-alue on koodin optimointi. Minimoimalla verkkoliikenne voidaan pienentää kustannuksia, joita tulee sisään- ja ulostulevasta tiedonsiirrosta, sekä nopeuttaa verkon suorituskykyä. Verkkoliikennettä voidaan optimoida seuraavilla tavoilla:

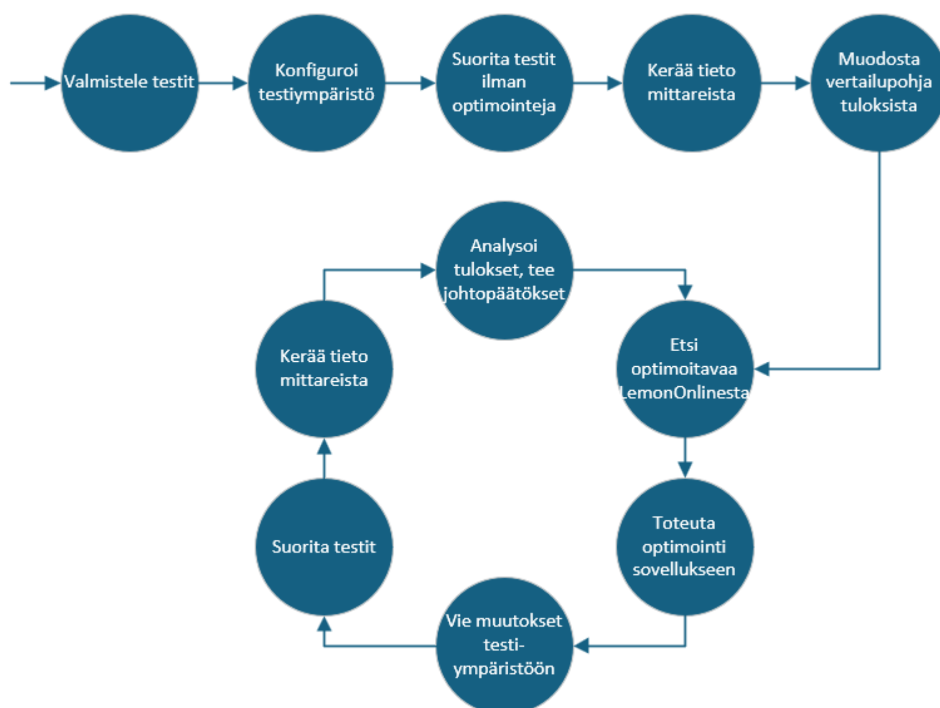
- Vähennä redundanttisia verkkopyyntöjä. Koodia analysoimalla voidaan löytää toisteisia kutsuja, joissa haetaan samaa tietoa useaan kertaan. Sovellus voi myös sisältää turhia kutsuja, joissa tietoa ei hyödynnetä. Toisteiset kutsut voidaan poistaa varastoimalla tieto kerran ja käyttää sitä uudelleen.
- Minimoi siirrettävän tiedon koko järjestelmän komponenttien välillä. Tieto voidaan pienentää ennen siirtoa erilaisilla pakkaustavoilla tai käyttämällä tehokkaampaa tietotyyppiä.
- Yhdistä verkkopyyntöjä. Pienempiä kutsuja yhdistämällä isommaksi ja hakemalla enemmän tietoa yhdellä kutsulla voidaan vähentää ylimääräistä prosessointitarvetta, jota tulee usean yhteyden muodostuksesta, sekä kokonaistiedonsiirtoa.
- Käytä tiedon sarjallistamista, jossa monimutkainen tietorakenne muunnetaan kevyempään ja standardisoituun muotoon, joka on helpompi lähettää verkon yli.

(Microsoft, 2023e)

5.3 Tutkimusmenetelmät

Tutkimuskysymyksien laatimisen ja niihin vastauksien etsimisen jälkeen suunniteltiin strategia, miten tutkimuksellista kehitystyötä tehtiin. Strategian suunnittelussa hyödynnettiin Azure Well-Architected Frameworkin ohjetta suorituskyvyn testaamiseen (Microsoft, 2023f), joka tarjoaa suosituksia ja hyviä käytäntöjä suorituskyvyn mittaukseen, tiedon keruuseen, testaustyökaluihin ja testausmenetelmiin. Testausta tarvitaan, että kehitystyön tuloksia voidaan mitata ja analysoida.

Ohjetta mukaillen valmisteltiin ensin testit, eli päätettiin, mitä suorituskykyyn liittyviä ominaisuuksia halutaan mitata, valittiin testaustyyppi, testaustyökalu ja luotiin testisuunnitelma. Testiympäristö konfiguroitiin vastaamaan todellista sovelluksen toimintaympäristöä. Testit suoritettiin ja tulokset analysoitiin. Ennen opinnäytetyön kehittämisvaihetta luotiin suorituskyvystä vertailuarvot ennen muutoksia, joita voi peilata eri kehityksen jälkeisiin mittaustuloksiin. Kehitys- ja tutkimistyössä analysoitiin resurssikäyttöä ja etsittiin optimoitavia asioita. Löydösten perusteella tehtiin yksittäisiä optimointeja teorioihin perustuen ja suoritettiin testiajot samaa LemonOnlinen Azure-ympäristöä vasten. Tulokset kirjattiin ja analysoitiin. Analyysin perusteella voitiin todeta, saavutettiinko optimoinnilla hyötyjä ja vastasiko teoria käytäntöä. Kun hyötyjä saavutettiin, tehtiin uusia optimointeja eri tavoin, samaa prosessia iteroiden. Kuvion 8 vuokaavio havainnollistaa tutkimusosuuden prosessin kulkua.



Kuvio 8. Opinnäytetyön tutkimuksellisen osan kehitysprosessi.

6 SUORITUSKYVYN TESTAUS JA MITTAUS

Tässä kappaleessa käydään läpi menetelmiä, joilla LemonOnlinen suorituskykyä testattiin ja mitattiin kehitystyön aikana. Testauksen suunnittelussa hyödynnettiin Azure Well-Architected Frameworkin suorituskyvyn testaus -oppaan suosittelemia strategioita ja käytäntöjä (Microsoft, 2023f).

6.1 Testien valmistelu

LemonOnlinea haluttiin kuormittaa testeillä mahdollisimman realistisesti, oikean käyttäjän toimintoja mukaillen. Testien tuli olla myös toistettavissa aina täsmälleen samanlaisesti jokaisella kerralla. Tästä syystä sovelluksen manuaalinen testaaminen olisi hankala toteuttaa, joten testausta varten valittiin Playwright-työkalu, jonka avulla voidaan tehdä päästä-päähän (eng. end-to-end) -testausta automatiikan avulla. Playwright on eräänlainen robotti, joka tekee ennalta määrittelyt toiminnot valitussa web-selaimessa TypeScript/JavaScript-koodilla kirjoitettujen skriptien mukaisesti. Skripteillä suoritetaan esimerkiksi käyttäjää simuloivia klikkauksia painikkeisiin, linkkeihin ja muihin web-elementteihin, sekä tehdään syötteitä web-lomakekenttiin. Testiajosta tulee lopuksi raportti, josta näkee statistiikkaa sen kulusta. (Playwright, n.d.)

LemonOnline on laaja verkkosovellus, joka sisältää useita kymmeniä sivuja ja satoja toimintoja. Koko tuotteen testaaminen olisi erittäin aikaa vievää, joten testauksen laajuus määriteltiin koskemaan vain sen käytetyimpiä näyttöjä ja toimintoja. Lemonsoft kerää tietoa käyttäjistä Google Analytics ja Hotjar -palveluiden avulla. LemonOnlinen asiakassovellus on instrumentoitu lähettämään statistiikkaa käyttäjien sivuvierailuista Google Analyticsiin. Käyttäjien istuntoja nauhoitetaan satunnaisesti Hotjar-palveluun obfuskoituna, eli niistä on poistettu käyttäjän tai yrityksen yksilöivät tiedot. Nauhoitteiden avulla halutaan parantaa käyttökokemusta ja niistä voidaan tarkkailla esimerkiksi hiiren liikkeitä ja käyttäjän suorittamia toimintoja. Google Analytics -palvelusta saadun statistiikan mukaan viimeisen vuoden ajalta kymmenen eniten vierailtua LemonOnlinen sivua olivat seuraavat:

- Ostolaskujen hyväksyntä
- Sovellukset
- Etusivu
- Tuotannon työjono
- Asiakaskeskus
- Työaikaleimaus
- Laskutuskeskus
- Myyntitilaus
- Lasku
- Ostolasku

Hotjar-nauhoitteista tutkittiin käyttäjiä kyseisillä näytöillä ja tehtiin johtopäätökset, mitkä olivat eniten käytetyimmät toiminnot. Toiminnot kerättiin yhteen ja muodostettiin niistä käyttötapauksia, joissa on yhtenäinen polku mitä testeissä voidaan toistaa. Lopulta muodostettiin viisi erillistä testiskenaariota, jotka koskevat yhtä tai useaa näyttöä. Skenaariot nimettiin testattavan asian mukaan ja siinä tehtävät toiminnot numeroitiin toteutusjärjestyksessä:

- Etusivu
  1. Avaa etusivu
  2. Odota, että kaikki tiedot ovat lataantuneet
- Ostolasku
  1. Avaa Ostolaskukeskus
  2. Luo uusi ostolasku
  3. Lisää ostolaskulle rivi
  4. Lisää kierrätysjonoon kirjautunut käyttäjä ja tallenna
  5. Siirry Ostolaskujen hyväksyntä -näytölle uuden ostolaskun numerolla
  6. Hyväksy ostolasku
  7. Siirry Ostolasku-näytölle ostolaskun numerolla
  8. Poista ostolasku
- Tuotantotyö

1. Avaa Nimikerekisteri-näyttö
2. Luo uusi nimike
3. Laita nimikkeelle ominaisuudeksi ”Valmistus, puolivalmiste” ja tallenna
4. Siirry Tuoterakenne-näytölle
5. Lisää uusi rivi työvaiheisiin ja tallenna
6. Siirry Uusi tuotantotyö-näytölle
7. Tallenna ja vie tuotantoon
8. Navigoi Tuotannon työjono -näytölle
9. Suodata työlista työnumerolla
10. Valitse työ ja kirjaa se valmiiksi Valmistuskirjaus-näytöllä
11. Avaa Nimikerekisteri-näyttö uusi nimike valittuna
12. Poista nimike

- Myyntitilaus
  1. Avaa Asiakaskeskus-näyttö
  2. Luo uusi Myyntitilaus
  3. Lisää myyntitilaukselle rivi ja tallenna
  4. Poista myyntitilaus
- Työaika
  1. Avaa Työaikaleimaus-näyttö
  2. Leimaa sisään
  3. Leimaa ulos

Testiskenaarioiden perusteella kirjoitettiin vastaavat toiminnot TypeScript-ohjelmointikielellä. Etusivun testin lähdekoodissa (koodi 2) navigoidutaan etusivulle ja odotetaan kunnes kaikki tiedot ovat lataantuneet, eikä verkkoliikennettä enää tapahdu. Lopuksi tarkistetaan, että sivun otsikko täsmää.

```
test("expect that the main page opens up", async ({
page, context }) => {

await page.goto(`${Utils.readBaseUrl()}/lemon/home`);
await page.waitForURL('**/lemon/home');
await page.waitForLoadState('networkidle');
```

```
expect(await page.title()).toContain('Etusivu');
});
```

Koodi 2. Etusivu-testin lähdekoodi.

Muiden testien lähdekoodit ovat liitteinä: Tuotantotyö (liite 1), Ostolasku (liite 2), Myyntitilaus (liite 3) ja Työaika (liite 4).

6.2 Testiympäristön konfigurointi

Azure-testiympäristö rakennettiin kappaleen 4.3.2 kuvion 5 arkkitehtuurin mukaisesti. Pilvipalveluille asetettiin sopivat ohjelmat ja palvelutasot, joiden valintaperusteena oli pienet kustannukset ja sopivat laskentaresurssit suorituskyvyn mittauksen helpottamiseksi, kuitenkin niin että sovellus toimii riittävän nopeasti automaattitestausta varten.

Elastic pooliin vietiin kaksi tietokantaa, nimeltään Database #1 ja Database #2, joita tarvitaan sovelluksen toiminnallisuuteen. Tietokannat sisälsivät ennalta luotua dataa testausta varten.

App Serviceen asennettiin LemonOnlinen versio 2024.9. Sovellus instrumentoitiin Application Insights-palvelulla, jonka avulla voidaan kerätä tietoa sen tapahtumista ja suorituskyvystä. Testien perimmäinen tavoite oli kerätä tietoa, mitä kustannuksiin vaikuttavia resursseja käytetään ja kuinka paljon. Suorituskykymittauksissa haluttiin valita sellaisia ominaisuuksia, mitä on mahdollista mitata automaattisesti ilman manuaalista työtä.

Azure portaalin Dashboard-sivulle voidaan luoda näkymä erilaisten widgettien avulla, jotka näyttävät metriikkaa pilvipalveluista. Dashboardiin lisättiin kaikista eri LemonOnlinen pilviarkkitehtuuriin kuuluvista pilvipalveluista tai niiden osista mittarit. Mittareiksi valittiin palveluiden ja resurssien ominaisuuksia, joilla oli merkitystä kustannustehokkuuden kannalta. Dashboardin mittarit ovat lueteltu taulukossa 1.

Taulukko 1. Azure portaalin Dashboardin suorituskykymittarit.

Azure-palvelu	Mittarin selite	Mittaustapa	Yksikkö
Front Door	Verkkopyyntöjen lukumäärä	summa	tavu
Front Door	Pyynnöissä tullut data	summa	tavu
Front Door	Vastauksissa lähetetty data	summa	tavu
App Service Plan	Prosessorin laskenta-aika	summa	sekunti
App Service	Prosessorin käyttöaste	maksimi	prosentti
App Service	Prosessorin käyttöaste	keskiarvo	prosentti
App Service	Muistin käyttöaste	maksimi	prosentti
App Service	Muistin käyttöaste	keskiarvo	prosentti
Elastic pool ja tietokannat	eDTU-kapasiteetin kulutus	maksimi	eDTU
Elastic pool ja tietokannat	eDTU-kapasiteetin kulutus	keskiarvo	eDTU
Elastic pool ja tietokannat	Prosessorin käyttöaste	maksimi	prosentti
Elastic pool ja tietokannat	Prosessorin käyttöaste	keskiarvo	prosentti
Redis for Cache	Muistin käyttö	maksimi	tavu
Redis for Cache	Muistin käyttö	keskiarvo	tavu
SignalR	Lähetettyjen viestien lukumäärä	summa	kpl
SignalR	Ulospäin lähetetty data	summa	tavu

6.3 Testien ajo

Playwright-testit ajettiin Chrome-selaimella, käyttäen normaalia LemonOnlinen 2024.9-versiota vertailupohjan luomiseksi. Vertailupohjaa hyödynnettiin, kun samat testit suoritettiin muokatuille versioille, joissa oli tehty optimointeja. Ennen mittaamisen aloittamista, testiagentilla kirjauduttiin sisään ja tallennettiin kirjautumistiedot muistiin, ettei kirjautumista tarvinnut suorittaa enää yksittäisten testiskriptien suoritusten välillä. Käyttäjätunnuksilla oli aktivoituna kaikki lisenssit ja roolit, sekä asetettu etusivulle oletuswidgetit. Testiskriptit ajettiin yhden kerran juuri ennen varsinaisten testien aloittamista, ettei Azure-palveluiden ”kylmäkäynnistys” aiheuta viiveitä tai heittoja ensimmäisiin mittaustuloksiin. Palvelimen pään välimuisti haluttiin myös saada tällä tavoin täytettyä etukäteen, että sen vaikutukset näkyisivät heti mittaustuloksissa.

Testikokoelma, joka piti sisällään viisi eri testiskenaariota, ajettiin kymmenen kertaa kolmena eri päivänä, eli yhteensä kolmekymmentä kertaa. Yhden testikokoelman ajo kesti keskimäärin kaksi ja puoli minuuttia. Testin valmistumisen jälkeen

pidettiin tauko, kun testiajon statistiikka kirjattiin ylös. Testien uudelleenajo käynnistettiin manuaalisesti n. viiden minuutin välein. Testausta haluttiin suorittaa eri päivinä, jotta voitiin varmistaa, että tulokset pysyvät samankaltaisina testausajankohdasta riippumatta.

Testien ajon jälkeen Azure portaalin Dashboardin aikavälisuodatusta säädettiin vastaamaan testin alkua ja loppua. Mittarit näyttävät arvoja minuutin tarkkuudella. Mittarien arvoja ei voitu kerätä suoraan käyttöliittymästä, sillä arvojen tarkkuus kärsi pyöristysten takia, esim. tavut muunnettiin kilotavuiksi tai megatavuiksi arvon suuruuden mukaan. Tarkat arvot saatiin kerättyä verkkoselaimen network-välilehdeltä, verkkopyynnöissä palautunutta tietoa tutkimalla. Testiajojen aikatiedot ja mittarien arvot kerättiin Excel-taulukkoon. Taulukon alle laskettiin kaikkien testiajojen mittarien keskiarvo, mediaani, minimi, maksimi ja varianssi.

6.4 Tulokset

Vertailupohjana käytetyn suorituskykymittauksen tulokset ovat liitteessä 5. Mittarien näyttämät luvut pysyivät melko tasaisina eri testiajojen välillä, lukuun ottamatta App Servicen prosessorin laskenta-aikaa, käytön keskiarvoa ja maksimia, joissa esiintyi piikkejä yksittäisissä testiajoissa. Kyseisissä mittareissa onkin mielekkäämpää tarkastella mittarien mediaania keskiarvon sijaan. Testiajojen kestossa oli yllättävän paljon vaihtelua, vaikka testit suoritettiin samoin täysin automatisoidusti ja toiminnot suoritettiin ilman viiveitä, heti käyttöliittymän sen salliessa. Yksittäinen testiajo kesti nopeimmillaan 2 minuuttia 15 sekuntia ja hitaimmillaan 2 minuuttia 55 sekuntia. Ero aiheutui todennäköisesti palvelimen vastausaikojen vaihtelusta, kutsujen valmistuessa välillä hitaammin.

Application Insights -palvelun lokitiedoista tuotettiin listaus yksittäisen testin aikana käytetyimmistä rajapintametodeista, jotta saataisiin ymmärrys mitä tietoja asiakassovellus hakee eniten palvelimelta. Tietoja hyödynnettiin optimointeja suunnitellessa.

7 KUSTANNUSTEHOKKUUDEN KEHITTÄMINEN

LemonOnlinen optimointivaiheessa hyödynnettiin kappaleessa 5.2 esitettyjä tekniikoita ja menetelmiä. Optimoinneissa keskityttiin pääasiassa välivarastointiin ja verkkoliikenteen vähentämiseen asiakassovelluksen ja pilven välillä. Optimointien kriteereinä oli, että niistä saatava hyöty voidaan todistaa mittareilla, eikä optimointi saa huonontaa käyttökokemusta. Optimointi tulee myös olla toistettavissa, jotta samaa menetelmää voidaan soveltaa LemonOnlinessa jatkossa, muuallakin kuin yhteen esimerkkinä käytettyyn tilanteeseen.

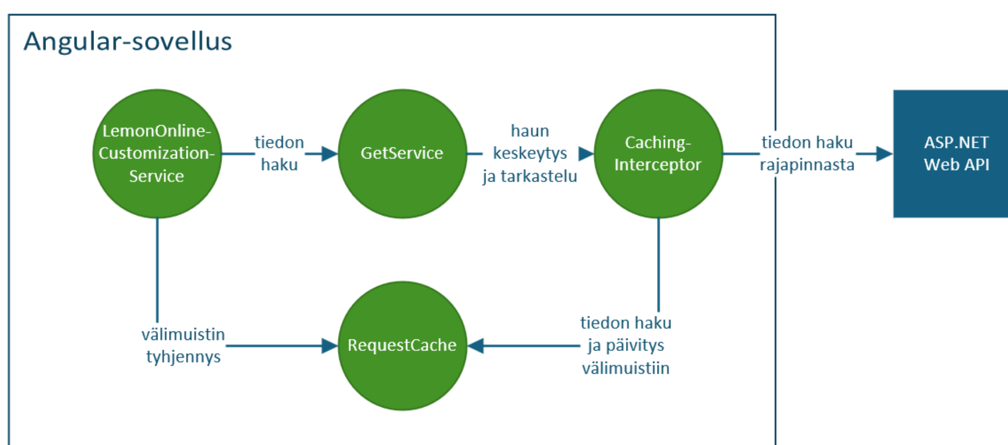
Optimointiesimerkit tehtiin LemonOnlinen 2024.9 -versiota pohjana käyttäen, joita varten luotiin omat branchit Lemonsoftin Git-repositorioon. Jokaisesta branchista käännettiin LemonOnlinen uusi versio, joka asennettiin App Serviceen. Testit suoritettiin jokaiselle esimerkille luvun 6 menetelmin, kerättiin mittarien tiedot Excel-taulukoihin ja verrattiin uusia arvoja vertailupohjan arvoihin.

Optimoinneista syntyi lopulta viisi esimerkkiä, joiden toteutus ja tulokset käydään tässä kappaleessa läpi. Tuloksissa mainitaan vain merkityksellisimmät mittarit, joissa ei esiintynyt paljon varianssia testiajojen välillä. Kaikki mittaukset voidaan tarvittaessa lukea liitteinä olevista taulukoista.

7.1 Välimuistin käyttö asiakassovelluksessa

LemonOnlinessa käyttäjä voi tehdä hänelle yksilöllisiä mukautuksia käyttöliittymään, jotka tallennetaan tietokantaan, niin että mukautukset ovat voimassa myös seuraavilla kerroilla, kun sovellusta käytetään. Käyttäjä voi esimerkiksi mukauttaa etusivulla näkyviä widgettejä, eri sivuilla näkyvien taulukkojen sarakeasetuksia, suodatinpalkissa näkyviä suodatinkenttiä tai pikasuodattimia. Yhden mitatun testiajon aikana päätepistettä kutsuttiin kymmeniä kertoja, jossa samaa tietoa haettiin osittain useaan kertaan. Ensimmäisen kerran haettu tieto voitaisiin välivarastoida sovelluksen muistiin. Toisteinen tiedonhaku minimoimalla voitaisiin teoriassa vähentää verkkoliikennettä ja kutsuista aiheutuvaa prosessointia pilvessä.

Esimerkissä hyödynnettiin LemonOnlinen asiakassovelluksesta jo nykyäänkin löytyvää välimuistijärjestelmää, jossa ohjelmoija voi päättää välivarastoidaanko tiettyyn API-päätepisteeseen tapahtuvan kutsun paluuarvo. Ratkaisuun lisättiin puuttuva mekanismi, jonka avulla yksittäiseen kutsuun sidottu paluuarvo voidaan tyhjentää välimuistista, muuten kuin verkkoselain virkistämällä ja lataamalla koko verkkosivu uudelleen. Välimuistin toimintaperiaate esitetään kuviossa 9.



Kuvio 9. LemonOnlinen asiakassovelluksen välimuistin toimintaperiaate.

Kuvion ympyröiden palvelut ovat JavaScript-luokkia, jotka käyttävät Angularin Dependency Injection -mekanismia, jonka avulla Angular luo niistä esiintymät (Angular.dev, 2025b). Palveluita voidaan siis ottaa käyttöön injektoimalla ne yksittäisiin Angular-komponentteihin (esim. älykkäät komponentit) tai muihin palveluihin. Tässä esimerkissä käytetyt luokat ovat singleton-tyyppisiä, eli niitä on vain yksi esiintymä koko sovelluksessa. LemonOnlineCustomizationServicen avulla voidaan tehdä käyttäjän mukautuksiin liittyviä CRUD-pyyntöjä ja sitä käytetään LemonOnlinen eri komponenteissa, joissa on mukatettavia toimintoja.

Kuviossa tiedon haku alkaa, kun LemonOnlineCustomizationServicen pyytää halutulla URI:lla ja parametreilla tietoa GetServicestä, joka käynnistää verkkopyynnön ja asettaa välimuistin käytöstä kertovan tunnisteen sen header-tietoihin. CachingInterceptor keskeyttää pyynnön ennen sen lähtöä rajapintaan ja tarkastaa

kutsun header-tietojen perusteella, onko tieto välivarastoitavissa. Jos tieto on välivarastoitava, CachingInterceptor pyytää tietoa RequestCache-luokasta, joka toimii välimuistina. Välimuisti koostuu avain-arvo -pareista, joissa avaimena on API-päätepisteen osoite parametreineen ja arvo on verkkopyynnöstä palautuva arvo. Jos tieto löytyy, se palautetaan heti GetServicelle ja verkkopyyntö ei lähde rajapintaan, muussa tapauksessa tuore tieto haetaan rajapinnasta, asetetaan välimuistiin ja palautetaan se LemonOnlineCustomizationServicelle.

Esimerkissä muutettiin LemonOnlineCustomizationServicen tekemät yksittäisten UI-kontrollien mukautusten tiedonhaut välivarastoitaviksi. Välimuistin tyhjennys toteutettiin saman tiedon uudelleen tallennuksessa ja poistossa. Tieto soveltuu hyvin välimuistissa pidettäväksi, sillä se ei muutu usein, sitä käytetään paljon ja se muuttuu vain käyttäjän toimesta, joten välimuistia on helppo ylläpitää.

Menetelmän avulla yhden testiajon aikana verkkopyyntöjen lukumäärä asiakassovelluksesta Azure Front Door -palveluun väheni keskimäärin kymmenellä, kuten liitteen 6 mittaustuloksista voidaan todeta. Saavutettu hyöty oli siis suhteellisen pieni kokonaisuudessaan, vähennyksen ollessa yhteensä alle prosentti kaikista verkkopyynnöistä. Tämä johtunee vain yhden tyyppisen tiedon välivarastoimisesta. Myös testiskenaarioiden välillä tapahtuva verkkosivun uudelleenlataus heikentää vaikutusta välimuistin tyhjentyessä. Välimuistijärjestelmää on kuitenkin kohtalaisen helppo toistaa erityyppistä tietoa haettaessa, joten siinä on potentiaalia isompaankin hyötyyn. Sovelluskohteita tulevaisuudessa voisi olla esimerkiksi combobox-tyyppisten hakukenttien tietolähteet.

7.2 Verkkosivun uudelleenlataukset

Angular-viitekehyksellä tehdyt sovellukset ovat SPA-tyyppisiä (Single-Page Application). Sovellus käyttää yhtä HTML-dokumenttia, johon luodaan komponenttipohjaisen lähdekoodin avulla koko verkkosivusto. Käyttäjän liikkuessa eri sivujen välillä, siinä näkyvät elementit luodaan dynaamisesti JavaScriptin avulla. Selaimen

osoiterivi päivitetään vastaamaan sivua, jolle käyttäjä navigoituu. Angular-viitekehys sisältää Router-ominaisuuden, joka tekee reitityksen käyttäjän navigoituessa eri sivujen välillä, jolloin kehittäjän tehtäväksi jää eri verkkosivuja vastaavien moduulien tai komponenttien verkko-osoitteiden määrittely. (Angular.dev, 2025c)

SPA-sovellus on yleensä isompi kooltaan ladata verkkoselaimeen ensimmäisen kerran verkkosivustoa käytettäessä, kuin traditionaalinen verkkosivusto, jossa eri sivuilla on omat HTML-dokumenttinsa ja sivun käyttämät JavaScript- CSS- ja muut tiedostot. SPA-sovelluksessa voidaan pitää sen muistissa usealla eri sivulla tarvittavaa tietoa, joten alkulatauksen jälkeen se ei vaadi enää yhtä paljon eri tiedostojen lataamista käytön aikana. Tästä syystä koko sovelluksen uudelleenlataaminen istunnon aikana aiheuttaa samojen tiedostojen uudelleen lataamisen palvelimelta, sovelluksen alustuksessa tehtävien verkkopyyntöjen toistamisen, sekä välimuistissa olevien tietojen menettämisen.

Suorituskykytesteissä huomattiin, että työaikaleimauksien (sisään, ulos) jälkeen selainikkuna virkistetään ohjelmallisesti ja koko verkkosovellus ladataan uudelleen. Tämä tehtiin sivulla näkyvien tietojen päivittämiseksi, sillä sivun luova näyttökomponentti hakee alustuksessaan mm. kirjautuneen käyttäjän edellisen leimauksen tiedot. Näyttökomponenttia ei oletuksena luoda uudelleen, vaikka reititys nykyiseen osoitteeseen tehtäisiin ohjelmallisesti Router-luokan navigate-metodin avulla. Tämän kappaleen optimointiesimerkissä kyseistä oletusta muutettiin toimimaan päinvastoin, asettamalla sovelluksen juuritasolla ominaisuus onSameUrlNavigation: ”reload”. Koska sovelluksen kaikkien näyttökomponenttien ei haluta toimivan samoin, sovelluksen juuritason reititykseen tehtiin uusi luokka nimeltään CustomRouteReuseStrategy, joka implementoi Angular-kirjaston abstraktin RouteReuseStrategy-luokan. TypeScript-kielessä on mahdollista toteuttaa (implements) abstraktit luokat periytymättä (extends) siitä. CustomRouteReuseStrategy-luokassa abstrakti shouldReuseRoute-metodi määrittelemällä ja käyttöönottamalla luokka sovelluksen juurimoduulin providers-konfiguroinnissa, voidaan kustomoida saman reitin (route) uudelleenkäytössä tehtävät tapahtumat. Metodi

määritettiin toimivan niin, että työaikaleimaus-näytön reittiin navigoituessa uudestaan, näyttökomponentti luodaan ja alustetaan uudelleen. Näiden toimenpiteiden jälkeen selaimen virkistävä koodi työaikaleimaus-näytössä voitiin korvata Router-luokan navigate-metodin käytöllä, jolloin koko sovellusta ei ladata jokaisen leimaustapahtuman jälkeen uudestaan.

Työaikaleimaus-sivun uudelleenlatauksen olisi voinut estää myös refaktoroimalla koko näyttökomponentti toimimaan eri tavalla leimausten jälkeen, niin ettei selainikkunan virkistystä olisi tarvinnut tehdä. Optimointiesimerkki olikin enemmänkin laastariratkaisu piilevään ongelmaan, että voitiin todistaa uudelleenlatauksesta tapahtuva turha laskenta- ja verkkoresurssien kulutus. Esimerkkitoteutusta ei siis tule käyttää sellaisenaan, vaan sovellus tulee ohjelmoida niin, ettei virkistystä tarvita.

Merkittävimmät optimoinnista saadut hyödyt näkyvät taulukossa 2, jossa vertaillaan suorituskykymittareiden prosentuaalista muutosta ennen ja jälkeen optimoinnin. Vertailtavat luvut ovat testiajojen mediaaneja.

Taulukko 2. Mittarit verkkosivun uudelleenlatausten poiston jälkeen. (liite 7)

Azure-palvelu	**Mittarin selite ja mittaustapa**	**Muutos %**
Front Door	Verkkopyyntöjen lukumäärä, summa	-8,3%
Front Door	Vastauksissa lähetetty data, summa	-11,7%
Front Door	Pyynnöissä tullut data, summa	-6,1%
App Service Plan	Prosessorin laskenta-aika, summa	-15,9%
App Service	Prosessorin käyttöaste (%), keskiarvo	-20,3%
Elastic pool	eDTU-kapasiteetin kulutus, keskiarvo	-6,2%
Elastic pool	Prosessorin käyttöaste, keskiarvo	-5,4%

Tulokset vastasivat odotuksia, eli pilvipalveluiden resurssikäyttö pieneni ja verkkoliikenne väheni optimointien jälkeen. Lisähuomiona testiajojen suoritusaika (mediaani) nopeutui n. 9 sekuntia. Ajansäästö saavutettiin todennäköisesti sillä, että testiajossa ei jouduttu enää lataamaan ja alustamaan verkkosivua kahta ylimääräistä kertaa.

7.3 Tilanhallintajärjestelmän hyödyntäminen

LemonOnlinen toimintakeskusasetusten avulla voidaan mukauttaa sovelluksen toimintaa asiakasyritysten tarpeiden mukaisesti. Asetukset voivat esimerkiksi määrittää, miten jokin sovelluksen ominaisuus tai bisneslogiikka käyttäytyy tietyssä tilanteessa. Asetuksia muuttamalla voidaan vaikuttaa sovellukseen yrityslaajuisesti kaikkiin tai vain yksittäisiin käyttäjiin.

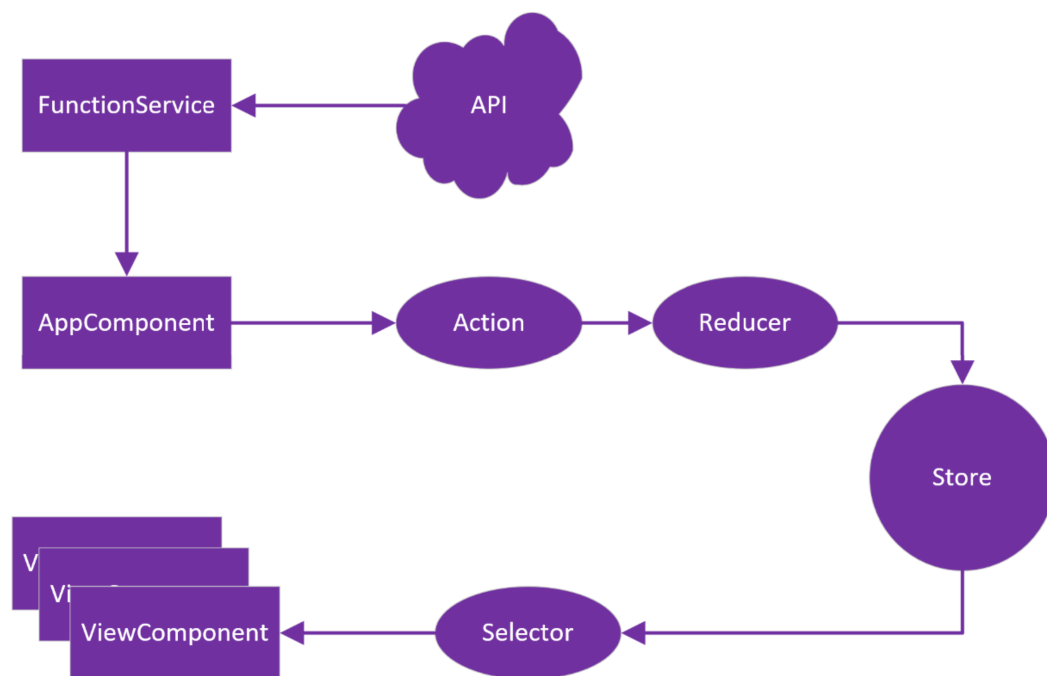
Mittausten mukaan yksittäisiä toimintokeskusasetuksia haettiin testiajon aikana kymmeniä kertoja. Asetuksia haetaan yksi kerrallaan eri näyttökomponenteissa tarpeen mukaan. Vaikka yksittäiset asetukset varastoidaan asiakassovelluksen välimuistiin ensimmäisen kutsun jälkeen, niitä on määrällisesti niin paljon, että kaikkien tarvittavien asetusten haku kuormittaa verkkoa ja taustapalveluita, koska tietoa haetaan pieniä määriä kerrallaan ja monessa verkkopyynnössä.

LemonOnline käyttää NgRx-kirjastoa, jonka avulla voidaan rakentaa Angular-sovelluksiin keskitetty tilanhallintajärjestelmä. Optimointiesimerkissä tilanhallintajärjestelmää hyödynnettiin tallentamalla LemonOnlinen toimintokeskusasetukset järjestelmän muistiin. Kaikki yrityksen laajuiset ja kirjautuneen käyttäjän toimintokeskusasetukset haetaan kerralla varastoon, josta tarvittavat asetukset voidaan hakea yksittäisiin näyttökomponentteihin.

NgRx-tilanhallintajärjestelmä koostuu neljästä pääelementistä: action, reducer, selector ja Store. Action on tapahtuma, jonka käynnistää Angular-komponentti tai palvelu. Reducer laskee tapahtuman tietojen ja vanhan tilan perusteella uuden tilan. Tila viedään Storeen (NgRx-luokka), joka toimii järjestelmän tietovarastona. Selector-funktiot tarkkailevat eri tilojen muutoksia ja välittävät niiden perusteella tietoa komponenteille tai palveluille. (NgRx.io, 2025b)

Optimointiesimerkissä sovelluksen LemonOnlinen juurikomponentti, AppComponent, hakee toimintokeskusasetukset FunctionServicen kautta rajapinnasta. Se käynnistää uniikin asetuksia varten luodun tapahtuman, eli actionin. Esimerkissä

hyödynnettiin olemassa olevaa reduceria, johon lisättiin kuuntelu asetusten tapahtumalle. Reducer ei esimerkissä muokkaa tietoa, vaan päivittää asetuksien tilan sellaisenaan Storeen. Asetusten noutoa varten luotiin oma selector, jota kaikki toimintokeskuasetuksia hyödyntävät näyttökomponentit käyttävät. Kuvio 10 havainnollistaa kyseistä tiedon kulkua tilanhallintajärjestelmässä.



Kuvio 10. Toimintokeskusasetukset LemonOnlinen tilanhallintajärjestelmässä.

Optimoinnin merkittävimmät seuraukset näkyvät taulukossa 3, jossa vertaillaan suorituskykymittareiden prosentuaalista muutosta ennen ja jälkeen optimoinnin. Vertailtavat luvut ovat testiajojen mediaaneja.

Taulukko 3. Mittarit tilanhallintajärjestelmään käytön jälkeen. (liite 8)

Azure-palvelu	**Mittarin selite ja mittaustapa**	**Muutos %**
Front Door	Verkkopyyntöjen lukumäärä, summa	-5,6%
Front Door	Vastauksissa lähetetty data, summa	+2,1%
Front Door	Pyynnöissä tullut data, summa	-6,4%
App Service Plan	Prosessorin laskenta-aika, summa	-4,7%
App Service	Prosessorin käyttöaste (%), keskiarvo	-8,8%
Elastic pool	eDTU-kapasiteetin kulutus, keskiarvo	+4,5%
Elastic pool	Prosessorin käyttöaste, keskiarvo	+4,5%

Tulosten perusteella verkkopyyntöjen lukumäärä pieneni huomattavasti, mutta asiakassovellukseen lähetetty datamäärä kasvoi aavistuksen verran. Tämä selittyy kaikkien asetuksien haulla, myös sellaisten, joita ei testiajon aikana tarvittu. Kaikkien asetusten haku tietokannasta myös kulutti enemmän Elastic poolin eDTU-kapasiteettia. Testikokoelmaa ei ajeta yhdessä istunnossa, vaan kokoelman jokaisen testin välillä testiagentti avaa uuden selaimen. Tästä syystä kaikki toimintokeskusasetuksen haettiinkin useamman kerran yhden testiajon aikana. Tällä oli todennäköisesti iso vaikutus myös tuloksissa näkyviin haittapuoliin, eli resurssikulutuksen kasvuun tietyissä palveluissa. Tulokset ovat luonnollisesti sitä parempia, mitä kauemmin käyttäjän yksittäinen istunto kestää, ennen selaimen sulkemista tai virkistämistä. Mitä enemmän sovelluksen alustuksessa haettuja toimintokeskusasetuksia voidaan hyödyntää, verrattuna vain tarvittaessa yksittäin hakuun, sitä parempi hyöty optimointiesimerkillä voidaan saavuttaa.

7.4 Verkkopyyntöjen optimointi

LemonOnlinessa on MyLemon-niminen valikko, josta kirjautunut käyttäjä voi nähdä hänelle yksilöityjä korttiwidgettejä, jotka sisältävät ilmoituksia ja tarjoavat mahdollisuuden navigoitua eri näytöille. Korteissa voi näkyä esimerkiksi käyttäjälle tarkastettaviksi osoitettujen ostolaskujen tai matkalaskujen lukumäärä. MyLemon-valikon korteissa näkyvä tieto haetaan aina sovelluksen alustuksessa jokaista korttia varten erikseen omalla verkkopyynnöllään. Korttien lukumäärä riippuu käyttäjän rooleista ja lisensseistä.

Optimointiesimerkissä kaikkien korttiwidgettien sisältämä tieto haettiin yhdellä kerralla rajapinnasta, tekemällä WidgetController-luokkaan uusi API-päätepiste, jonka kutsuun liitettiin metodi tietojen hakua ja palautusta varten. Angular-sovelluksen WidgetService-luokkaan tehtiin uusi metodi, joka ottaa parametreinaan korttiwidgettien yksilölliset numerot, joiden tieto haetaan http-kutsulla rajapinnasta. Sovelluksen juurikomponentti, AppComponent, refaktoroitiin käyttämään uutta datan hakumetodia Mylemon-valikon korttilistauksen tuottamiseen.

Merkittävimmät optimoinnista saadut hyödyt näkyvät taulukossa 4, jossa vertaillaan suorituskykymittareiden prosentuaalista muutosta ennen ja jälkeen optimoinnin. Vertailtavat luvut ovat testiajojen mediaaneja.

Taulukko 4. Mittarit verkkopyyntöjen optimoinnin jälkeen. (liite 9)

Azure-palvelu	Mittarin selite ja mittaustapa	Muutos %
Front Door	Verkkopyyntöjen lukumäärä, summa	-6,8%
Front Door	Pyynnöissä tullut data, summa	-8,1%
App Service Plan	Prosessorin laskenta-aika, summa	-11,4%
App Service	Prosessorin käyttöaste (%), maksimi	-35,3%
App Service	Prosessorin käyttöaste (%), keskiarvo	-29,1%

Kuten tuloksista voidaan todeta, App Servicen prosessoinnin määrä pieneni merkittävästi optimoinnin seurauksena. Tämä oli hieman yllättävää, sillä tieto mitä haettiin, oli täsmälleen sama mitä ennen optimointiakin. Ilmiö voi selittyä sillä, että samalla verkkopyyntöjen lukumäärän pienetessä, App Servicen ei tarvinnut käsitellä http-kutsuja yhtä montaa, vähentäen prosessorin kuormitusta.

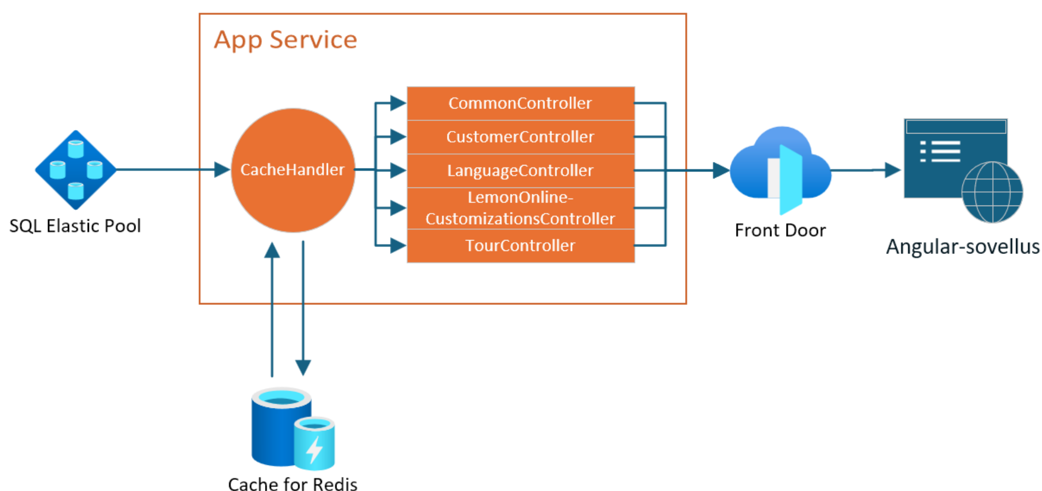
7.5 Välimuistin käyttö palvelinpäässä

Palvelinpään välimuistia haluttiin käyttää yhtenä optimointiesimerkkinä. Suorituskykytesteissä huomattiin, että monet useasti kutsuttavat rajapintakutsut ovat sellaisia, mitkä palauttavat harvoin muuttuvaa tietoa, mutta ne haetaan kuitenkin aina uudestaan tietokannasta. Tiedot olivat seuraavia:

- Käyttäjän tekemät käyttöliittymän mukautukset
- Tieto, onko käyttäjä nähnyt erilaiset sovelluksen toiminnallisuutta esittelevät opastukset (eng. tour)
- Tieto, onko asiakasyrityksen tietokannassa sovelluksen esittelytarkoitukseen käytettyä (demo) dataa
- Käyttäjän kielivalintaan perustuvat asiakassovelluksen termien kielikäännökset
- Yrityksen asiakkaiden huomio-, tai varoitustekstit

Optimoinnissa haluttiin asettaa kaikki yllä mainitut tiedot Azure Cache for Redis -välimuistiin, jotta tietokannan kuormitusta voitaisiin vähentää ja tiedot saataisiin nopeammin välimuistista. Välimuistijärjestelmään kuuluu itse Redis-palvelun lisäksi sen hallintaa varten CacheHandler-luokka, joka asettaa ja hakee tietoja välimuistiin avain-arvo -pareina, joissa avaimena toimii tiedolle yksilöllinen tunnistemerkkijono, sekä arvona varastoitava tieto. Olemassa olevaan luokkaan lisättiin metodit objektimuodossa olevan tiedon varastoimiseen, hyödyntäen JsonConvert-luokan SerializeObject ja DeserializeObject -metodeita. Lisäksi tehtiin apufunktiot avaimen luontiin yrityksen laajuista ja vain yhdelle yrityksen käyttäjälle välivarastoitavaa tietoa varten.

LemonOnlinen ASP.NET Web API -rajapintaan tulevat verkkopyynnöt käsitellään eri Controller-luokissa, jotka on nimetty tiedon kategorian mukaan. Esimerkkinä kielikäännöksiin liittyvä tieto kulkee LanguageController-luokan kautta. Aiemmin mainittujen välivarastoitavien tietojen vastaavat kontrollerit refaktoroitiin käyttämään tiedon hakuun CacheHandler-luokkaa, joka tarkastaa ensin löytyykö pyydetty tieto välimuistista Azure Cache for Redis -palvelusta. Jos tieto löytyy, se palautetaan asiakassovellukseen. Jos tietoa ei löydy välimuistista, se noudetaan bisneslogiikkakerroksesta, joka puolestaan hakee sen tietokantakyselyllä. Tietokannasta haettu tieto asetetaan välimuistiin myöhempää käyttöä varten ja palautetaan asiakassovellukseen. Välimuistijärjestelmän toimintaa optimointiesimerkissä esitellään kuviossa 11, jossa nuolet osoittavat suuntaa, mistä haettu tieto kulkeutuu asiakassovellukseen.



Kuvio 11. Tietovirta palvelinpään välimuistista asiakassovellukseen.

Optimoinnin merkittävimmät vaikutukset näkyvät taulukossa 5, jossa vertaillaan suorituskykymittareiden prosentuaalista muutosta ennen ja jälkeen optimoinnin. Vertailtavat luvut ovat testiajojen mediaaneja.

Taulukko 5. Mittarit palvelinpään välimuistin käytön jälkeen. (liite 10)

Azure-palvelu	Mittarin selite ja mittaustapa	Muutos %
App Service Plan	Prosessorin laskenta-aika, summa	-3,3%
App Service	Prosessorin käyttöaste (%), keskiarvo	-4,8%
Database #1	eDTU-kapasiteetin kulutus, keskiarvo	+3,3%
Database #2	eDTU-kapasiteetin kulutus, keskiarvo	-100%
Redis for Cache	Muistin käyttö, keskiarvo	+180,4%

Tuloksista voidaan nähdä, että välimuistiin tallennetun tiedon koko luonnollisesti kasvaa sen käytön lisääntyessä. Database #1 -tietokannasta haettiin vähemmän tietoa, mutta odotusten vastaisesti se ei näkynyt mittauksissa positiivisesti, tietokannan eDTU-käytön pysyessä lähes ennallaan. Toista tietokantaa ei kuormitettu optimoinnin jälkeen enää ollenkaan, koska tiedot haettiin testiajon aikana aina välimuistista. Käyttö oli yhdessä testiajossa muutenkin vähäistä, joten mittauksesta voidaan päätellä lähinnä vain, että välimuistijärjestelmä toimii oikein. Optimointiesimerkkiin valitut tiedot eivät vaatineet paljoa laskentatehoa, mikä voi selittää vähäisiä muutoksia tietokantojen kuormitusta tarkkailevissa mittareissa. Tärkeimpänä huomiona voidaan todeta, että App Servicen prosessointiaika väheni, mistä

voidaan päätellä, että sama tieto välimuistista haettaessa on kevyempi tehdä kuin tietokannasta.

8 TULOKSET

Luvun 7 optimointimenetelmien avulla saatiin näkyviä parannuksia LemonOnlinen suorituskykyyn. Opinnäytetyössä haluttiin kuitenkin parantaa kustannustehokkuutta, joten lopuksi haluttiin osoittaa, miten suorituskykyä parantamalla voidaan vaikuttaa kustannuksiin. Tätä varten kaikki yksittäiset optimointiesimerkit otettiin käyttöön samaan aikaan LemonOnlineen, suoritettiin testiajot ja kerättiin suorituskykymittarien arvot. Merkittävimmät muutokset, kun kaikki optimoinnit olivat käytössä samaan aikaan, näkyvät taulukossa 6. Vertailtavat luvut ovat testiajojen mediaaneja ja muutos niissä esitetään prosentuaalisesti.

Taulukko 6. Mittarit, kun kaikki optimointiesimerkit olivat käytössä. (liite 11)

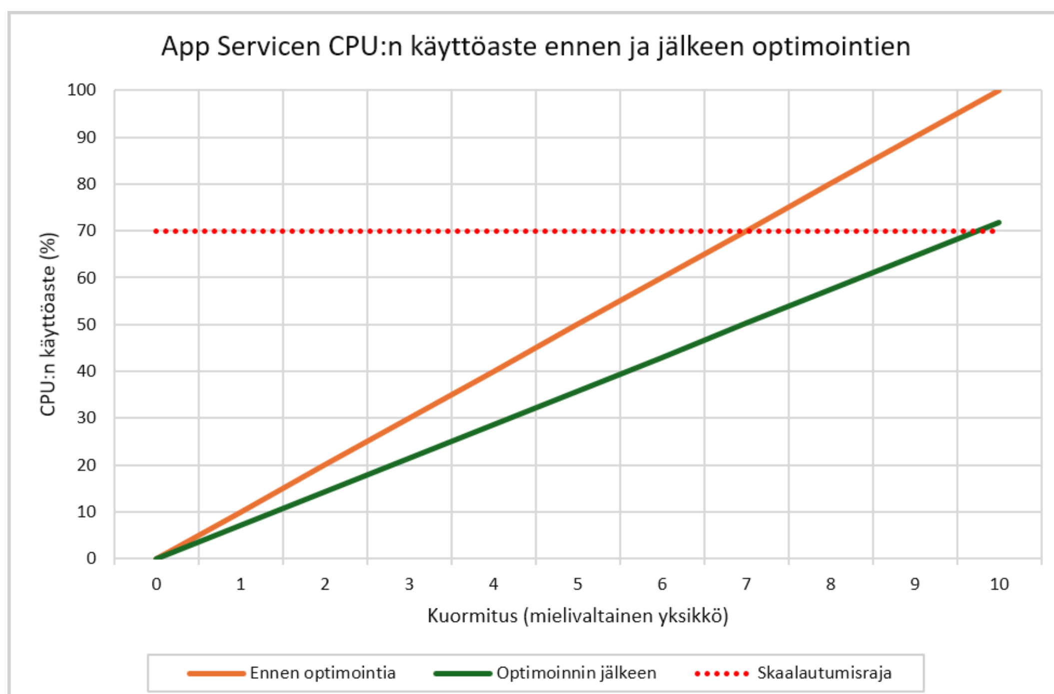
Azure-palvelu	Mittarin selite ja mittaustapa	Muutos %
Front Door	Verkkopyyntöjen lukumäärä, summa	-19,2%
Front Door	Pyynnöissä tullut data, summa	-18,4%
Front Door	Vastauksissa lähetetty data, summa	-11,5%
App Service Plan	Prosessorin laskenta-aika, summa	-16,5%
App Service	Prosessorin käyttöaste (%), maksimi	-41,2%
App Service	Prosessorin käyttöaste (%), keskiarvo	-28,3%
Database #2	eDTU-kapasiteetin kulutus, keskiarvo	-100%
Redis for Cache	Muistin käyttöaste (%), keskiarvo	+180,4%

Tuloksista voidaan päätellä, että optimoinnit kumosivat osin toistensa vaikutusta, mutta niiden yhteisvaikutus oli silti merkittävä. Optimoinnit olivat kuitenkin vain yksittäisiä esimerkkejä, mutta samoja menetelmiä toistamalla voidaan saavuttaa vieläkin suurempi hyöty. Esimerkeissä ei tehty sovelluksen laajuista optimointia, koska se olisi vaatinut huomattavan määrän aikaa ja substanssiosaamista. Sovellus kehittyy jatkuvasti, vanhojen ominaisuuksien poistuessa ja uusien lisääntyessä. Esimerkkien olikin tarkoitus toimia malleina, miten kehitystyötä voidaan jatkossa tehdä suorituskyky huomioiden ja optimointimenetelmiä hyödyntäen.

Koska Azure App Servicen laskentaresursseja käytetään optimointien avulla vähemmän, voidaan teoriassa palvelun resurssikapasiteettia laskea. App Servicen

kustannuksiin vaikuttaa sen instanssien lukumäärä, sekä App Service Planin ohjelma ja palvelutaso. Instanssien lukumäärää voidaan skaalata automaattisesti kuormituksen mukaan niin, että laskentakapasiteettia voidaan pitää mahdollisimman pienenä, mutta kuitenkin tarpeeksi suorituskykyisenä, ettei sovelluksen käyttö hidastu tai esty. Esimerkkinä jos prosessorin käyttö pysyy keskimäärin tietyllä ajanjaksolla yli 70%, voidaan lisätä yksi instanssi ja jakaa kuormitusta eri instansseille, jolloin yksittäisen instanssin prosessorin käyttöaste laskee. Vastaavasti ruuhkahuippujen ulkopuolella, kun instanssien prosessorien käyttöaste laskee alle 30%, voidaan ylimääräinen instanssi pudottaa käytöstä, jolloin kustannukset laskevat varatun resurssikapasiteetin pienentyessä.

Jos oletetaan, että prosessorin käyttöaste nousee lineaarisesti LemonOnlinen kuormituksen kasvaessa ja optimoinnit pienentävät App Servicen prosessorin käyttöastetta n. 28%, kuten optimointiesimerkit yhdistämällä tapahtui (taulukko 6), App Servicen tulisi teoriassa pysyä suorituskykyisempänä ja sietää enemmän kuormitusta, ennen kuin instanssien lukumäärää tarvitsee lisätä ja kuormaa jakaa. Kuvio 12 esittää prosessorin käyttöasteen ennen ja jälkeen optimoinnin, jossa kuormituksen kasvaessa (x-akseli) prosessorin käyttö kasvaa hitaammin optimointien avulla, jolloin tarve App Servicen instanssien lukumäärän kasvattamiselle tulee myös hitaammin.



Kuvio 12. App Servicen CPU:n käyttöaste optimointien jälkeen.

Saavutettu hyöty riippuu LemonOnlinen kuormituksesta, mikä vaihtelee sovelluksen käyttäjämäärien ja käyttötavan mukaan. Kustannushyötyjä on siis erittäin vaikea laskea etukäteen tietämättä tarkkaa kuormituksen määrää, käytettyä App Servicen ohjelmaa, palvelutasoa, sekä instanssien lukumäärää. Voidaan kuitenkin luoda teoreettinen skenaario, jossa LemonOnline isännöidään Sweden Central -alueella sijatsevassa App Servicessä. Automaattiskaalaus on asetettu nostamaan instanssien lukumäärää, kun 70% käyttöaste ylittyy, sekä laskemaan sitä käyttöasteen mennessä alle 30% ja instanssien lukumäärän ollessa kaksi tai enemmän. Skenaariossa optimoinnit vähentävät suorittimen käyttöastetta 28%. Vähennys saadaan kertomalla alkuperäinen käyttöaste luvulla 0,72. Koska käyttöaste mitataan prosentteina, siitä ei voida suoraan vähentää 28 prosenttiyksikköä. App Service Plan käyttää Premium v3 -ohjelman eri palvelutasoja ja ”Pay as you go” -hinnoittelumallia, eli hinnat ovat ilman alennuksia, koska palvelun käyttöön ei sitouduta tietylle ajanjaksolle (Microsoft, 2025f). Taulukko 7 esittää miten optimointi vaikuttaa kustannuksiin, kuormituksen vaihdellessa eri palvelutasoilla ja App Servicen instanssien lukumäärällä.

Taulukko 7. Optimointien vaikutus kustannuksiin.

Palvelu-taso	Instanssit	CPU % ennen	CPU % jälkeen	Vaikutus kustannuksiin
P1v3	3	50	36	Ei vaikutusta
P1v3	3	75	54	Instansseja ei tarvitse lisätä, koska käyttö laski alle 70%. Säästö 234€/kk
P2v3	3	60	43,2	Ei vaikutusta
P2v3	2	40	28,8	Voidaan vähentää yksi instanssi, koska käyttö laski alle 30%. Säästö 470€/kk
P2v3	2	65	46,8	Ei vaikutusta
P3v3	1	90	64,8	Instansseja ei tarvitse lisätä, koska käyttö laski alle 70%. Säästö 939€/kk
P3v3	2	45	32,4	Ei vaikutusta
P4mv3	2	35	25,2	Voidaan vähentää yksi instanssi, koska käyttö laski alle 30%. Säästö 2077€/kk
P4mv3	2	65	46,8	Ei vaikutusta
P5mv3	1	80	57,6	Instansseja ei tarvitse lisätä, koska käyttö laski alle 70%. Säästö 4153€/kk
P5mv3	2	55	39,6	Ei vaikutusta

Jos skaalausta ei tehdä lainkaan, optimointien jälkeen voidaan mahdollisesti valita alempi palvelutaso, ellei sillä ole isoa vaikutusta sovelluksen suorituskykyyn käyttäjille, minkä ansiosta kustannussäästö on palvelutasojen hinnan erotus.

Front Doorin -palvelun kustannussäästöjä voidaan arvioida teoreettisesti aiempien testiajojen tulosten perusteella. Koska yhden testiajon aikana mitattu Front Dooriin tulevien verkkopyyntöjen lukumäärä, sekä sisään- ja ulosmenevä datan määrä tiedetään, voidaan testiajoa pitää yksikkönä laskelmissa. Palvelun kuormituksen kasvua simuloidaan kasvattamalla testiajojen lukumäärää. Kun optimointien vaikutus Front Door -palvelun eri mittareihin tiedetään ja paljonko palvelun käyttö maksaa, voimme laskea paljonko optimointien avulla voidaan säästää kuormituksen ollessa tietyn suuruinen.

Front Door -palveluun saapuneet verkkopyynnöt maksavat Euroopan alueella 0,0143€ per 10 000 kappaletta (Microsoft, 2023c). Kun kaikki optimoinnit ovat käytössä, yhden testiajon aikana verkkopyyntöjä lähtee asiakassovelluksesta Front

Door -palveluun 269 kpl vähemmän. Näiden tietojen avulla voidaan laskea, paljonko säästöä saadaan verkkopyyntöjä vähentämällä (taulukko 8).

Taulukko 8. Säästö verkkopyyntöjä vähentämällä.

Testiajoja	Vähentynyt verkkopyyntöjen lukumäärä	Säästö
100	26 900	0,02 €
1 000	269 000	0,23 €
10 000	2 690 000	2,31 €
100 000	26 900 000	23,13 €
1 000 000	269 000 000	231,34 €
10 000 000	2 690 000 000	2 313,40 €

Front Door -palvelun kustannuksia käytiin tarkemmin läpi kappaleessa 4.2.3. Käytännössä kuluja aiheutuu verkkoliikenteestä pyyntöjen lukumäärän lisäksi myös palveluun sisään tulevan ja lähtevän datamäärän perusteella. Euroopan alueella Front Door -palvelusta App Servicelle kulkeva data, eli käytännössä sama kuin asiakassovelluksesta palveluun tuleva data, maksaa 0,02€/Gt. Front Doorista asiakassovellukseen (suunta ulospäin) kulkeva data maksaa 0,079€/Gt ensimmäisen 10 Teratavun osalta kuukauden aikana, jonka jälkeen se on hieman halvempi (Microsoft, 2023c). On kuitenkin hankala arvioida, miten paljon dataa kulkee kuukaudessa, joten laskentakaavaa yksinkertaistetaan oletuksella, että datan määrä on alle 10 Teratavua kuukaudessa, jolloin hinta pysyy vakiona. Kustannussäästö riippuen testiajojen lukumäärästä näkyy taulukossa 9, jossa ennen ja jälkeen optimointia mitattu siirretyn datan määrän erotus kerrotaan siirron suuntaa vastaavalla maksulla.

Taulukko 9. Säästö verkkoliikenteen datamäärää vähentämällä.

Testiajoja	Vähentynyt datan määrä, sisään (Gt)	Vähentynyt datan määrä, ulos (Gt)	Säästö
100	0	1	0,05 €
1 000	0	7	0,52 €
10 000	1	65	5,20 €
100 000	11	655	51,97 €
1 000 000	113	6 550	519,70 €
10 000 000	1 131	65 499	5 197,04 €

Taulukon 8 ja 9 tulosten perusteella näyttää siltä, että verkkoliikennettä täytyy kulkea Front Door -palvelun kautta erittäin paljon, ennen kuin sitä optimoimalla saadaan merkittäviä kustannushyötyjä. Verkkopyyntöjen prosessointi App Servicessä kuluttaa kuitenkin sen laskentaresursseja, mitkä aiheuttavat lisäkustannuksia. Tästä syystä verkkopyyntöjen optimointi hyödyttää isossa kuvassa kustannustehokkuutta.

Laskelmat ovat puhtaasti teoreettisia, eikä niiden perusteella voida tehdä tarkkoja arvioita kustannusten potentiaalisesta laskusta. Pilviresurssien käyttöön vaikuttaa monta muuttujaa, sillä LemonOnlinen käyttö vaihtelee suuresti reaalimaailmassa. Opinnäytetyön testit yrittivät simuloida oikeata käyttöä, mutta kuormitus tapahtui vain yhdellä käyttäjällä. Optimoinneista saatu hyöty ei siis ole suoraan verrattavissa oikeaan tuotantoympäristöön, jossa käyttäjiä on tuhansia ja sovelluksen käyttötavan vaihdellessa jokaisella käyttäjällä. Lemonsoftin käyttämät eri pilviresurssien palvelutasot ja instanssien lukumäärä tuotantokäytössä ovat paljon suurempia kuin käytetyssä testiympäristössä. Opinnäytetyölle oli rajallinen budjetti käytössä ja pilvipalveluille valittiin mahdollisimman halvat palvelutasot, niin että testaus pystyttiin suorittaa sujuvasti. Näin ollen laskelmat ovat suuntaa antavia, mitä luokkaa säästöt voisivat olla optimaalisessa tilanteessa. Todellinen hyöty voidaan nähdä vasta oikeassa tuotantoympäristössä oikeassa käytössä.

Opinnäytetyön tavoitteena oli löytää parhaat keinot, miten LemonOnlinea voidaan kehittää ja optimoida niin, että pilven käytön kustannuksia voidaan pienentää. Tutkimuskysymykset tarkentuivat opinnäytetyön edetessä seuraaviin:

- Kuinka verkkopyyntöjen kokoa ja määrää voidaan vähentää web-käyttöliittymän ja Azuren välillä?
- Kuinka App Servicen ja SQL Serverin laskentaresurssien käyttöä voidaan vähentää?

Luvun 7 optimointiesimerkit käyttöönottamalla, sekä toistamalla menetelmiä vanhaa koodia refaktoroidessa ja uutta kehittäessä, voidaan pienentää pilven käytön kustannuksia, kuten tämän luvun laskelmissa esitettiin. Tutkimuskysymyksiin saatiin siis vastaus kaiken muun osalta, paitsi SQL-serverin laskentaresursseja ei saatu pienennettyä. Aihe olisi vaatinut lisätutkimusta ja todennäköisesti suoraan Azure SQL Database ja Elastic pool -pilvipalveluiden käytön optimointia.

Opinnäytetyön päätavoitteen, eli kustannustehokkuuden parantamisen lisäksi eri pilvipalveluiden toimintaperiaate ja kustannusrakenne selvitettiin. LemonOnlinea varten luotiin Azuressa toimiva testiympäristö, kirjoitettiin Playwright-testit automaattitestaamista varten ja tehtiin mittauksia testiajoista. Prosessista saatua tietoa voidaan hyödyntää myös jatkossa, jos suorituskykyä halutaan testata tulevaisuudessa. Opinnäytetyön kustannustehokkuutta parantavia menetelmiä voidaan periaatteessa soveltaa muissakin verkkosovelluksissa, kuin LemonOnlinessa.

9 JOHTOPÄÄTÖKSET

9.1 Tutkimusaihe

Opinnäytetyön aihevalinta oli työelämän tarpeisiin räätälöity ja erityisesti Lemonsoftia hyödyttävä tuloksineen. Vastaavanlaisia tutkimuksia ei siis juuri löytynyt julkisista lähteisistä. Tutkimuksia muiden verkkosovelluksien optimoinnista oli tehty, jos pilviympäristöä ja kustannustehokkuutta ei huomioitu. Tästä syystä teoriapohjan muodostamiseen käytettiin tutkimusartikkeleita ja muita lähteitä, jotka keskittyivät verkkosovellusten yksittäisiin optimointimenetelmiin. Menetelmien soveltamisessa hyödynnettiin tekijän omaa työkokemusta LemonOnlinen parissa. Optimointi vaati syvää asiantuntemusta sovelluksen teknisestä puolesta ja liiketoimintalogiikasta. Tutkimuksessa apuna käytettyä Azure Well Architected Frameworkia kannattaa yleisesti hyödyntää sovelluksen ja pilvipalveluiden optimointiin, sekä suorituskyvyn testaamiseen.

9.2 Testaaminen ja mittarit

Opinnäytetyön testauksessa käytetyn Azure-pilviympäristön rakentaminen, LemonOnlinen isännöinti ja konfigurointi pilvessä toimivaksi oli työn hankalin ja aikaa vievin osuus. Toteutus oli kuitenkin onnistunut, jokseenkin LemonOnlinen raportointityökalut eivät toimineet, vaan niiden toimintakuntoon saamiseksi olisi vaadittu lisätyötä. Tämän takia raportointitoiminnallisuus jätettiin pois, eikä sen käyttöä tai suorituskykyä voitu testata.

Suorituskykytestaus Playwright frameworkilla oli erittäin kätevää, vaikka se onkin alun perin tarkoitettu verkkosovellusten toiminnallisuuden varmistamiseen. Manuaalisesti samojen testien tekeminen olisi ollut huomattavasti hitaampaa, eikä testiajoja olisi ollut mielekästä tehdä useita. Myös Azure Load Testing -palvelua harkittiin testaustyökaluksi, mutta sen huonoimpana puolena testejä olisi joutunut muokata mahdollisesti optimointien jälkeen lähdekoodin muuttuessa. Palve-

lun avulla simuloidaan vain verkkoliikennettä, mitä käyttöliittymästä kulkisi sovelluksen käytön aikana. Playwright-testit olisi jälkikäteen ajateltuna kannattanut kirjoittaa niin, että koko testikokoelma olisi ajettu samalla sessiolla selainta sulkematta välillä. Näin välimuistin käytön hyöty olisi näkynyt paremmin testituloksissa. Toisaalta sovelluksen oikea käyttö voi olla myös useammassa selainikkunassa tapahtuvaa, kun eri sivuja avataan välilehdille, kun eri tietoja halutaan tarkastella ilman jatkuvaa navigointia.

Osa suorituskykymittareista oli tarkempia kuin toiset. Verkkopyyntöihin liittyvät mittarit olivat suhteessa luotettavampia kuin esimerkiksi prosessorin käyttöön liittyvät mittarit. Prosessorin laskenta-ajan ja käyttöasteen mittarit heittelivät testiajojen välillä, joten niistä oli vaikea tehdä johtopäätöksiä ajamatta testejä monta kertaa ja vertaamalla mediaania. App Service suorittaa mahdollisesti muutakin kuin pelkkää LemonOnlinen vaatimaa laskentaa testiajoissa, joka saattaa näkyä tuloksissa. Jälkikäteen ajateltuna testiajot olisi voinut suorittaa useammalla Playwright-testiagentilla kerrallaan ja valitsemalla sellainen App Service Planin palvelutaso, että kuormitus olisi vienyt n. puolet App Servicen prosessorin käyttöasteesta ennen optimointeja. Näin menetellen olisi mahdollisesti saatu helpommin vertailtavia tuloksia optimointien jälkeen.

Testitulokset kertovat lopulta vain sen, mitä kuormitus oli itse valitsemallani kuormituksella, eli tietyllä käyttäjämäärällä ja ennalta määritellyillä käyttäjää simuloivilla testitapauksilla. Sovelluksen oikea käyttö ei vastaa testiskenaarioita, joissa pyrittiinkin vain simuloimaan käyttäjien yleisesti tekemiä toimintoja. Tuotantoympäristössä on tuhansia käyttäjiä, jotka tekevät eri toimintoja yritysten tarpeiden mukaisesti. Testauksessa käytetyt palvelutasot ovat myös erilaiset, joten laskentakapasiteetti ja palveluiden yleinen suorituskyky on tuotannossa eri tasolla kuin opinnäytetyössä. Tämän takia optimointien hyöty voidaan nähdä vasta tuotantoympäristössä suorituskykyä mittaamalla. Opinnäytetyön mittaukset ovat lähinnä suuntaa antavia kokoluokasta, mitä niistä saatava hyöty voisi parhaimmillaan olla.

9.3 Optimoinnit

Painotus optimoinneissa oli sovellustason välimuistiratkaisuiden hyödyntämisessä, joiden avulla saatiin aikaan suhteellisen pienellä vaivalla isoja suorituskyky- ja kustannushyötyjä. Välimuistijärjestelmät olivat jo sovelluksessa entuudestaan, mutta niissä oli puutteita, joiden takia niitä ei voitu käyttää ilman pientä lisäkehitystä. Välimuisteja käyttäessä on erittäin tärkeää tietää varastoitavan datan ominaisuudet: koko, muutostiheys, muutoksen aiheuttajat, kuinka usein tietoa pyydetään ja soveltuvuus eri käyttäjille. Välimuistijärjestelmiä kannattaa kehittää tulevaisuudessa vieläkin paremmiksi, mahdollisimman helposti käyttöön otettaviksi ja tuomalla ne kehittäjän yleiseksi työkaluksi muiden optimointimenetelmien ohella.

Azure Cache for Rediksen hyödyt voidaan nähdä parhaimmillaan vasta tuotantoympäristössä, kun sinne saadaan välivarastoitua enemmän dataa ja sellaista tietoa, mitä usea käyttäjä voi hyödyntää. Testien aikana oli vain yksi käyttäjä, joka käytti välivarastoa, joten välimuistin maksimaalista hyötyä ei voitu saavuttaa.

Sovelluksen käyttämän pilviympäristön kustannusrakenne ja sen kustannustehokkuutta heikentävät asiat olisi hyvä tuoda kehittäjien tietoisuuteen, jotta niiltä vältytään jatkossa ja osataan refaktoroida vanha koodi suorituskykyisemmäksi. Olisikin ensisijaisen tärkeää jo uusia ominaisuuksia suunnitellessa ja kehittäessä tehdä se suorituskykyä silmällä pitäen, koska jälkikäteen on aina hankalempi suorittaa optimointia ja kustannukset ovat silloin moninkertaiset.

9.4 Tulokset

Tutkimuksen päätavoitteet saavutettiin, eli löydettiin keinoja, miten LemonOnlinea voitiin kehittää ja optimoida kustannustehokkaammaksi pilviympäristössä. Työn tuloksena syntyi konkreettisia menetelmiä, joiden hyöty todistettiin mittauksilla ja laskelmilla. Lisäksi Azuren tarjoamat palvelut ja niiden kulurakenne esiteltiin. Lemonsoft voi realistisesti säästää heti pilvikustannuksissa ottamalla optimoinnit käyttöön katsomalla toteutukseen mallia opinnäytetyössä käytetyistä Git-

brancheista. Osa tuotetusta koodista vaatii vielä mahdollisesti perusteellisempaa testausta, eikä sovellu sellaisenaan tuotantokäyttöön ennen siistimistä. Optimointimenetelmiä voidaan lisäksi soveltaa myös muissa tilanteissa, kuin opinnäytetyön esimerkeissä. Lisäsäästöihin on selvästi potentiaalia, mutta optimoinneissa kannattaa myös huomioida paljonko optimointi vie kehittäjältä työaikaa, onko sillä vaikutusta sovelluksen käytettävyyteen tai joutuuko karsimaan muita ominaisuuksia. Myös mahdollinen optimoinnin ylläpito voi vaatia aikaa ja panostusta. Ideaalina olisikin löytää sovelluksesta paikkoja, joita helposti optimoimalla saisi aikaan isot kustannushyödyt.

Opinnäytetyö keskittyi sovelluksen optimointiin, mutta kustannushyötyjä voidaan saavuttaa myös valitsemalla parhaiten soveltuvat palvelut Azuressa ja konfiguroimalla ne tehokkaasti. Azure App Servicen dynaamisella skaalaamisella voidaan saavuttaa säästöjä, automaattisesti kapasiteettia nostaen ruuhkahuippujen aikaan, sekä laskemalla sitä, kun käyttö on vähäistä.

SQL Serverin ja tietokantojen optimointi jäi vähäiselle huomiolle opinnäytetyössä. Jatkotutkimuksena voisi selvittää mahdollisia optimointikeinoja, miten Elastic poolin resurssikäyttöä ja tietokantojen kokoa voisi pienentää. Opinnäytetyössä käytetyt välimuistit eivät näyttäneet pienentävän kuormitusta suuresti, toki niitä hyödynsikin vain yksi käyttäjä testien aikana. Azure SQL Database ja Elastic pool -palvelut ovat kallein osa pilviarkkitehtuurissa, joten niiden käyttöä optimoimalla on mahdollista saavuttaa isoja kustannushyötyjä.

LÄHTEET

Angular.dev. (2025a). What is Angular? Noudettu 1.1.2025 osoitteesta https://angular.dev/overview

Angular.dev. (2025b). Using Angular routes in a single-page application. Noudettu 22.2.2025 osoitteesta https://angular.dev/guide/di/dependency-injection

Angular.dev. (2025c). Understanding dependency injection. Noudettu 22.2.2025 osoitteesta https://angular.dev/guide/routing/router-tutorial

Lemonsoft. (2021). ERP hankintaopas. Noudettu 1.1.2025 osoitteesta https://news.lemonsoft.fi/oppaat/erp-hankinta

Lemonsoft. (2025a). Tuotteet ja palvelut. Noudettu 1.1.2025 osoitteesta https://www.lemonsoft.fi/tuotteet/

Lemonsoft. (2025b). Lemonsoft Oyj. Noudettu 1.1.2025 osoitteesta https://www.lemonsoft.fi/lemonsoft-oy/

Lemonsoft. (2025c). LemonOnline – Selainkäyttöinen toiminnanohjausjärjestelmä (ERP). Noudettu 5.5.2025 osoitteesta https://lemonsoft.fi/ohjelmakohtaiset-kuvaukset/lemononline-selainkayttoinen-erp/

Mertz, J. & Nunes, I. (2016). A qualitative study of application-level caching. IEEE Transactions on Software Engineering, 42(4), 1–25. https://doi.org/10.1109/TSE.2016.2633992

Mertz, J. & Nunes, I. (2017). Understanding application-level caching in web applications: A comprehensive introduction and survey of state-of-the-art approaches. ACM Computing Surveys, 9(4), artikkeli 39. https://doi.org/10.48550/arXiv.2011.00477

Microsoft. (2020). Introduction to SignalR. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Microsoft. (2021). Introduction to Visual Basic. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/dotnet/visual-basic/reference/language-specification/introduction

Microsoft. (2022a). ASP.NET Overview. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/aspnet/overview

Microsoft. (2022b). Database Engine Instances (SQL Server). Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/database-engine-instances-sql-server?view=sql-server-ver16

Microsoft. (2022c). Azure hosting recommendations for ASP.NET Core web apps. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/azure-hosting-recommendations-for-asp-net-web-apps

Microsoft. (2023a). Overview of .NET Framework. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview

Microsoft. (2023b). ASP.NET MVC Overview. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

Microsoft. (2023c). Understand Azure Front Door billing. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/frontdoor/billing

Microsoft. (2023d). What is Azure SignalR Service? Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/azure-signalr/signalr-overview

Microsoft. (2023e). Recommendations for Optimizing Code Costs. Noudettu 11.2.2025 osoitteesta https://learn.microsoft.com/en-us/azure/well-architected/cost-optimization/optimize-code-costs

Microsoft. (2023f). Recommendations for Performance Testing. Noudettu 11.2.2025 osoitteesta https://learn.microsoft.com/en-us/azure/well-architected/performance-efficiency/performance-test

Microsoft. (2024a). What is SQL Server? Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16

Microsoft. (2024b). App Service Overview. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/app-service/overview

Microsoft. (2024c). What is Azure SQL Database? Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview

Microsoft. (2024d). Elastic pools help you manage and scale multiple databases in Azure SQL Database. Microsoft Learn. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview

Microsoft. (2024e). What is Azure Front Door? Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/frontdoor/front-door-overview

Microsoft. (2024f). What is Azure Cache for Redis? Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/azure-cache-for-redis/cache-overview

Microsoft. (2024g). Azure Monitor overview. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/azure-monitor/overview

Microsoft. (2024h). Scale up an app in Azure App Service. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/app-service/manage-scale-up

Microsoft. (2025a). What is Cloud Computing? Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing

Microsoft. (2025b). What is IaaS? Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-iaas/

Microsoft. (2025c). What is PaaS? Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-paas/

Microsoft. (2025d). Serverless Computing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-serverless-computing/

Microsoft. (2025e). Cache-Aside pattern. Noudettu 22.1.2025. osoitteesta https://learn.microsoft.com/en-us/azure/architecture/patterns/cache-aside

Microsoft. (2025f). App Service on Windows pricing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/pricing/details/app-service/windows/

Microsoft. (2025g). Azure SQL Database pricing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/elastic/

Microsoft. (2025h). Azure Front Door pricing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/pricing/details/frontdoor/

Microsoft. (2025i). Azure Cache for Redis pricing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/pricing/details/cache/

Microsoft. (2025j). Azure SignalR Service pricing. Noudettu 1.1.2025 osoitteesta https://azure.microsoft.com/en-us/pricing/details/signalr-service/

Microsoft. (2025k). What is the Azure Well-Architected Framework? Noudettu 11.2.2025 osoitteesta https://learn.microsoft.com/en-us/azure/well-architected/what-is-well-architected-framework

Microsoft. (2025l). Basic web application. Noudettu 1.1.2025 osoitteesta https://learn.microsoft.com/en-us/azure/architecture/web-apps/app-service/architectures/basic-web-app

Mozilla. (2024a). HTML: Creating the Content. MDN Web Docs. Noudettu 1.1.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Your_first_website/Creating_the_content

Mozilla. (2024b). CSS: Styling the Content. MDN Web Docs. Noudettu 1.1.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Your_first_website/Styling_the_content

Mozilla. (2024c). JavaScript. MDN Web Docs. Noudettu 1.1.2025 osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript

NgRx.io. (2025a). What is NgRx? Noudettu 1.1.2025 osoitteesta https://ngrx.io/docs

NgRx.io. (2025b). @ngrx/store. Noudettu 26.2.2025 osoitteesta https://ngrx.io/guide/store

Playwright. (n.d.). Installation. Noudettu 12.2.2025 osoitteesta https://playwright.dev/docs/intro

Shailesh, K. S. & Suresh, P. V. (2017). An analysis of techniques and quality assessment for web performance optimization. Indian Journal of Computer Science and Engineering, 8(2), 61–69. ISSN: 0976-5166. Noudettu 5.5.2025 osoitteesta https://www.ijcse.com/docs/INDJCSE17-08-02-100.pdf

T-SQL Tutorial. (n.d.). Introduction to Transact-SQL. Noudettu 1.1.2025 osoitteesta https://www.tsql.info/

TypeScript. (2024). TypeScript for the New Programmer. Noudettu 1.1.2025 osoitteesta https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

Vepsäläinen, J., Hellas, A. & Vuorimaa, P. (2024). Overview of web application performance optimization techniques. arXiv. https://doi.org/10.48550/arXiv.2412.07892

W3Schools. (n.d.). Sass Introduction. Noudettu 1.1.2025 osoitteesta https://www.w3schools.com/sass/sass_intro.php

LIITTEET

LIITE 1
TUOTANTOTYÖ-TESTIN LÄHDEKOODI

```
1  test("expect that new product can be created, set as production job, completed and deleted", async ({ page, context }) => {
2  await page.goto(`${Utils.readBaseUrl()}/logistic/product`);
3  await page.waitForURL('**/logistic/product');
4  await page.waitForLoadState('networkidle');
5  // Click create new product toolbar button
6  await page.getByTestId('ls-product-0fbfd500-3218-40f6-ae39-6abd7267b86a').getByRole('button').first().click();
7  await page.waitForSelector('c-dialog', { state: 'visible' });
8  const productCode = Utils.randomProductCode();
9  // Fill product code
10 await page.locator('c-dialog #productCode').click();
11 await page.locator('c-dialog #productCode').fill(productCode);
12 // Create the new product
13 await page.getByTestId('ls-new-product-dialog-3ad0d113-5be8-45b3-8acc-74e8a251f75f-button').click();
14 await page.waitForURL('**/logistic/product/*');
15 await expect(page.locator(`.toast-text >> text=Nimike ${productCode} luotu`)).toHaveText(`Nimike ${productCode} luotu`);
16 // Capture the product number from the URL for use in the later tests
17 const pageUrl = page.url();
18 const productNumber = pageUrl.split('/').pop(); // the product number is the last segment of the url
19 // wait for test id ls-product-info-1e3cf5bc-779f-4b94-baf9-cf7784b59c0c-combobox-input input value to be the product code
20 await page.waitForSelector('div.toolbar-title-wrapper:has-text(" Nimikerekisteri - ' + productCode + ' ")');
21 // Open attributes tab
22 await page.getByTestId('ls-product-8b48ed4e-7d37-4db3-a77c-a5105f1757c8-attributes').click();
23 // Click "reset filters" button to show all attributes
24 await page.locator('#ls-attributes-grid-2 [data-ref="resetFiltersButton"]').click();
25 // click div that has inner text of "2. Valmistus, puolivalmiste"
26 await page.locator('.b-grid-cell:has-text("2. Valmistus, puolivalmiste")').click();
27 // Save all the changes
28 await page.getByTestId('ls-product-09d740d9-28f1-4982-b7be-ea5a70577168').getByRole('button').click();
29 await expect(page.locator(`.toast-text >> text=Nimike ${productCode} on tallennettu`)).toHaveText(`Nimike ${productCode} on tallennettu`);
30 // Navigate to product structure
31 await page.getByTestId('ls-product-17a324b1-e3d1-4551-af83-3a21a8da2cc1').click();
32 await page.locator('.cdk-overlay-container span:has-text("Tuoterakenne")').first().click();
33 await page.waitForURL('**/logistic/productstructure/**');
34 // Open work phases tab
35 await page.locator('#subworkstages-button').click();
36 // Click add new row button
37 await page.locator('#structure-subworkstages-grid [data-ref="createNewButton"]').click();
38 await expect(page.locator(`.toast-text >> text=Työvaihe lisätty ja tallennettu rakenteeseen`)).toHaveText(`Työvaihe lisätty ja tallennettu rakenteeseen`);
39 // Navigate to new production job with the current product
40 await page.getByTestId('toolbar-dropdown-button5-in-desktop-toolbar').click();
41 await page.locator('ls-dropdown-menu span:has-text(" Uusi tuotantotyö ")').click();
42 await page.waitForURL('**/production/productionnewjob');
43 // Get the work number of the production job
44 const workNumber = await page.locator('.b-grid-cell[data-column="Worknumber"]').first().innerText();
45 // Wait for the save button not to be disabled
46 await page.waitForSelector('c-toolbar-menu-item-crud-save:not(.--disabled)');
47 // Save and add to production
48 await page.locator('c-toolbar-menu-item-crud-save').getByRole("button").first().click();
49 // Confirm the first dialog that appears
50 await page.waitForSelector('c-common-dialog c-dialog', { state: 'visible' });
51 await page.locator('c-dialog c-primary-button').getByRole("button").click();
52 // Confirm the second dialog that appears
53 await page.waitForSelector('ls-done c-dialog', { state: 'visible' });
54 await page.locator('c-dialog div.content:has-text("Aloituslupa")').click();
55 await page.locator('c-dialog c-primary-button').getByRole("button").click();
56 await expect(page.locator('.toast-text >> text=Tallennus valmis')).toHaveText('Tallennus valmis');
57 // Close the navigation dialog that appears
58 await page.waitForSelector('ls-navigation c-dialog', { state: 'visible' });
59 // Navigate to production job list
60 await page.goto(`${Utils.readBaseUrl()}/production/productionjoblist`);
61 await page.waitForURL('**/production/productionjoblist');
62 // Click last page button to show the latest production job
63 await page.locator('#joblist button[data-ref="lastPageButton"]').click();
64 // Click the row with work number
65 page.locator(`div[role="gridcell"]:has-text("${workNumber}")`).first().click();
66 // Wait for the page to load the production job from server, before pressing finish.
67 await page.waitForLoadState('networkidle');
68 // Click finish production job button
69 await page.locator('button[data-ref="jobDone"]').click();
70 // Wait for the automatic navigation to the production done page after the barcode has been entered
71 await page.waitForURL('**/production/productionjobactiondone/**');
72 // Leave comment before finishing the production job
73 await page.getByTestId('ls-production-job-amount-done-d3c438be-53b6-4b6c-a697-1650ca5e0306').click();
74 await page.getByTestId('ls-production-job-amount-done-d3c438be-53b6-4b6c-a697-1650ca5e0306').fill('Työ valmis');
75 // Click the "Done" button in the production done page
76 await page.getByTestId('ls-production-job-action-done-417664af-dfcb-4192-845a-e47b8c5e93b3').getByRole('button').click();
77 // The page will navigate automatically back to production job list
78 await page.waitForURL('**/production/productionjoblist/*');
79 // Navigate back to the product page using the product number
80 await page.goto(`${Utils.readBaseUrl()}/logistic/product/${productNumber}`);
81 await page.waitForURL('**/logistic/product/*');
82 // Delete the product that was created in the start of the test
83 await page.waitForSelector('[data-testid="ls-product-46b2518d-65b6-46bb-8d0a-99cbabcc70d1"]:not(.--disabled)');
84 await page.getByTestId('ls-product-46b2518d-65b6-46bb-8d0a-99cbabcc70d1').getByRole('button').click();
85 // Confim the delete
86 await page.waitForSelector('c-common-dialog c-dialog', { state: 'visible' });
87 await page.locator('c-dialog c-primary-button').getByRole("button").click();
88 await expect(page.locator(`.toast-text >> text=Nimike ${productCode} poistettu`)).toHaveText(`Nimike ${productCode} poistettu`);
89 });
```

LIITE 2

OSTOLASKU-TESTIN LÄHDEKOODI

```
1  test("expect that new purchase invoice can be created, accepted in circulation and deleted", async ({ page, context }) => {
2  await page.goto(`${Utils.readBaseUrl()}/fm/purchaseinvoicecentrum`);
3  await page.waitForSelector('ls-purchase-invoice-centrum-summaries c-loader-bar');
4  await page.waitForSelector('ls-purchase-invoice-centrum-summaries c-loader-bar', { state: 'detached' });
5  expect(await page.title()).toContain('Ostolaskukeskus');
6  // Click create new purchase invoice toolbar button
7  await page.getByTestId('ls-purchase-invoice-centrum-5a936756-9a48-4352-91a6-8088903418ab').getByRole('button').click();
8  // Click create in the dialog
9  await page.waitForSelector('c-dialog', { state: 'visible' });
10 await page.getByTestId('ls-purchase-invoice-new-dialog-content-08bc84d7-8ecd-46fe-9380-4d543f2fbd88-button').click();
11 await expect(page.locator('.toast-text >> text=Ostolasku luotiin onnistuneesti. Avataan ostolaskua')).toHaveText('Ostolasku luotiin onnistuneesti. Avataan ostolaskua');
12 // Wait for the new invoice to open
13 await page.waitForURL('**/purchaseinvoice/*');
14 expect(await page.title()).toContain('Ostolasku ');
15 // Capture the invoice number from the URL for use in the later tests
16 const url = page.url();
17 const invoiceNumber = url.split('/').pop(); // the invoice number is the last segment of the url
18 // Input sum of invoice
19 await page.getByTestId('ls-purchase-invoice-side-bar-018c0fa1-d144-4cc0-b70b-8ec55699d294').click();
20 await page.getByTestId('ls-purchase-invoice-side-bar-018c0fa1-d144-4cc0-b70b-8ec55699d294').fill("123");
21 // Create new invoice row
22 await page.locator('ls-purchase-invoice-row-list c-data-table-controls').getByRole('button').filter({ hasText: 'Lisää rivi' }).click();
23 await page.waitForLoadState('networkidle');
24 // Input row amount
25 await page.locator('[id^="dataCell-Purchase_invoicerow_total-Summa-"]').fill("123");
26 await page.waitForLoadState('networkidle');
27 // Open circulation tab
28 await page.getByTestId('ls-purchase-invoice-content-area-22577243-c08a-4d89-8d33-2490772ac502-circulation').click();
29 // Create new circulation row
30 await page.locator('ls-purchase-invoice-circulation-list c-data-table-controls').getByRole('button').filter({ hasText: 'Lisää rivi' }).click();
31 await page.waitForSelector('ls-purchase-invoice-circulation-list c-data-table c-loader-bar');
32 await page.waitForSelector('ls-purchase-invoice-circulation-list c-data-table c-loader-bar', { state: 'detached' });
33 // Get the logged in user name from the top right, assuming that he can be selected for the invoice circulation
34 const spanContent = await page.locator('[data-testid="app-704d03d7-1492-4d14-aa2f-f5f6f564efde"] span').first().textContent();
35 // Trim the content to remove any starting or ending spaces
36 const loggedInUserName = spanContent.trim();
37 // Fill the circulation row combo search input with the name of the logged in user
38 await page.locator('[id^="dataCell-Person_number-"]').click({delay: 1000})
39 await page.locator('[id^="dataCell-Person_number-"]').pressSequentially(loggedInUserName);
40 // Select the logged in user from the dropdown
41 await page.locator(`c-dropdown-menu span:text-is(" ${loggedInUserName} ")`).click();
42 await page.waitForSelector('c-data-table c-loader-bar');
43 await page.waitForSelector('c-data-table c-loader-bar', { state: 'detached' });
44 // Select the user as the approver
45 await page.locator('c-data-table c-select').getByRole('button').click();
46 await page.locator(`c-dropdown-menu span:text-is(" Hyväksyjä ")`).click();
47 await page.waitForSelector('c-data-table c-loader-bar');
48 await page.waitForSelector('c-data-table c-loader-bar', { state: 'detached' });
49 // Save invoice
50 await page.getByTestId('ls-purchase-invoice-e21bba67-c82e-40b4-9620-9a3a8dcf1672').getByRole('button').click();
51 // Confirm that saving was succesful from Toast text
52 await expect(page.locator('.toast-text >> text=Ostolasku tallennettu')).toHaveText('Ostolasku tallennettu');
53 await page.goto(`${Utils.readBaseUrl()}/fm/newpurchaseinvoicecirculation/${invoiceNumber}`);
54 await page.waitForURL('**/newpurchaseinvoicecirculation/*');
55 // Wait for approve button to be enabled and then click it
56 await page.waitForSelector('[data-testid="toolbar-button-10-in-desktop-toolbar"]:not(.--disabled)');
57 await page.getByTestId('toolbar-button-10-in-desktop-toolbar').last().click();
58 await expect(page.locator('.toast-text >> text=Ostolasku hyväksytty')).toHaveText('Ostolasku hyväksytty');
59 // Delete the purchase invoice
60 await page.goto(`${Utils.readBaseUrl()}/fm/purchaseinvoice/${invoiceNumber}`);
61 await page.waitForURL('**/purchaseinvoice/*');
62 await page.waitForSelector('[data-testid="ls-purchase-invoice-c82abcdd-fc0e-4ba8-a1c7-7f41ef5c3313"]:not(.--disabled)');
63 await page.getByTestId('ls-purchase-invoice-c82abcdd-fc0e-4ba8-a1c7-7f41ef5c3313').click();
64 await page.waitForSelector('c-dialog', { state: 'visible' });
65 await page.locator('c-dialog c-primary-button').click();
66 await expect(page.locator('.toast-text >> text=Ostolasku poistettu')).toHaveText('Ostolasku poistettu');
67 });
```

LIITE 3

MYYNTITILAUS-TESTIN LÄHDEKOODI

```
1  test("expect that new sales order can be created, saved and deleted", async ({ page, context }) => {
2  await page.goto(`${Utils.readBaseUrl()}/crm/customercentrum`);
3  await page.waitForURL('**/crm/customercentrum');
4  // Pick the first customer from the customer combo box
5  await page.getByTestId('ls-customer-search-3676b2b1-faaf-47eb-9abb-6cad926cf871-combobox-dropdown-toggle').click();
6  await page.locator('c-dropdown-menu').first().click();
7  // Create new sales order
8  await page.getByTestId('ls-customer-centrum-8fff647b-19ff-4bc2-81de-dd0f8f4174da').click();
9  await page.getByTestId('ls-customer-centrum-67ca611b-effc-438c-b52a-7a11fbe18b3b').click();
10 await page.waitForSelector('c-dialog', { state: 'visible' });
11 await page.locator('c-dialog c-primary-button:not(.--disabled)').click();
12 await page.waitForURL('**/logistic/salesorder/*');
13 // Add new row to the sales order
14 await page.locator('ls-row-management-grid [data-ref="createNewButton"]:not(.b-disabled)').click();
15 await page.waitForSelector('div.b-grid-cell.b-editing.b-focused', { state: 'visible' });
16 // Select first product from the product combo box
17 await page.locator('.b-grid-cell[data-column="Row_productcode"] input').press(' '); // fill the combo with space to open the dropdown
18 await page.locator('.b-float-root .b-menuitem').first().click();
19 // Save the sales order
20 await page.locator('[data-testid="ls-sales-order-aaee42f2-ccda-4587-ac3e-5b9c52dc14c7"]:not(.--disabled)').click();
21 // Delete the sales order
22 await page.locator('[data-testid="ls-sales-order-01b4a5cb-d2c7-40aa-90e2-4bab05d1f002"]:not(.--disabled)').click();
23 await page.waitForSelector('c-dialog', { state: 'visible' });
24 await page.locator('c-dialog c-primary-button').click();
25 await expect(page.locator(`.toast-text >> text=Myyntitilaus poistettu`)).toHaveText(`Myyntitilaus poistettu`);
26 // Wait for the automatic navigation to the sales order list view
27 await page.waitForURL('**/logistic/sales-orders');
28 await page.waitForLoadState('networkidle');
29 expect(await page.title()).toContain('Myyntitilaukset');
30 });
```

LIITE 4

TYÖAIKA-TESTIN LÄHDEKOODI

```
1  test("expect that the person can sign in and out of work ", async ({ page, context }) => {
2  // Set geolocation permissions to 'denied'
3  await context.grantPermissions([], { origin: `${Utils.readBaseUrl()}` });
4  // Go to work time stamping terminal
5  await page.goto(`${Utils.readBaseUrl()}/hr/worktimestampterminal`);
6  // Log in
7  await page.getByTestId('ls-worktime-stamping-5b27261a-66bd-424f-997c-5d2265f39006-button').click();
8  // Close the info dialog
9  await page.locator('c-dialog c-primary-button').click();
10 // Wait for the page to refresh automatically
11 await page.waitForURL('**/hr/worktimestampterminal');
12 // Log out
13 await page.getByTestId('ls-worktime-stamping-5b27261a-66bd-424f-997c-5d2265f39006-button').click();
14 // Close the info dialog
15 await page.locator('c-dialog c-primary-button').click();
16 // Wait for the page to refresh automatically
17 await page.waitForURL('**/hr/worktimestampterminal');
18 await page.waitForLoadState('networkidle');
19 expect(await page.title()).toContain('Työaikaleimaus');
20 });
```