Mikko Huotari

OSSI-SOVELLUKSEN MODERNI-SOINTI

Opinnäytetyö

Tekniikan ammattikorkeakoulututkinto

Ohjelmistotekniikan koulutus

2025

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä/Tekijät	Mikko Huotari
Työn nimi	OSSI-sovelluksen modernisointi
Toimeksiantaja	Etelä-Savon Koulutus Oy
Vuosi	2025
Sivut	45 sivua
Työn ohjaaja(t)	Tuomas Reijonen

TIIVISTELMÄ

Tämä opinnäytetyö toteutettiin Etelä-Savon Koulutus Oy:n toimeksiantona, ja se perustuu syventävällä harjoittelujaksollani aloitettuun opiskelijoiden osaamisen- ja projektienhallintajärjestelmä OSSIn modernisointiprojektiin. Työn aihe on luoda analysoiva prosessikuvaus OSSIn modernisoinnista nykyaikaisilla teknologioilla. Työssä käsitellään vanhan ja uuden sovelluksen teknologiat, sovelluksen arkkitehtuuri, syyt modernisoinnin tarpeelle sekä työn toteutusvaihe. Lisäksi lopuksi käsitellään sovellukselle mahdollisia jatkokehityskohteita.

Modernisoinnin taustalla oli halu tarjota opiskelijoille mahdollisuutta osallistua uuden sovelluskehityksen prosessiin sekä valmistella heitä paremmin tämänhetkisen työelämän vaatimuksiin teknologioiden osalta. Lisäksi vanhan OSSIn arkkitehtuuri ja käytetyt teknologiat olivat vanhentumassa, mikä olisi johtanut suureen määrään koodin uudelleen kirjoittamista.

Modernisoinnissa hyödynnettiin muun muassa Docker-konttiteknologiaa palveluiden hallinnan ja käyttöönoton helpottamiseksi sekä siirtymistä nykyaikaisiin JavaScript-kehyksiin käyttöliittymän rakentamisessa. Opinnäytetyössä kuvataan valittujen teknologioiden etuja, projektin toteutusvaiheita sekä niiden ratkaisuja.

Työn tavoitteena on tarjota hyödynnettävä prosessikuvaus OSSIn kaltaisten sovellusten modernisointiin sekä tuottaa Etelä-Savon Koulutus Oy:lle arvokasta dokumentaatiota jatkokehityksen tueksi. OSSI-sovelluksen modernisoinnin tuloksena sovelluksen ylläpidettävyys, skaalautuvuus sekä suorituskyky paranevat merkittävästi, mikä tukee Etelä-Savon Koulutus Oy:n OSSI-projektin tavoitteita. Lisäksi opiskelijoiden osaaminen nykyaikaisten teknologioiden parissa kasvaa, mikä edistää heidän työllistymismahdollisuuksiaan ja vastaa entistä paremmin työelämän jatkuvasti muuttuviin tarpeisiin. Sovelluksen kehittäminen on tätä opinnäytetyötä kirjoittaessa kesken, minkä johdosta opinnäytetyön sisältö vastaa sovelluksen sen hetkistä tilaa.

Asiasanat: sovelluskehitys, ohjelmistoarkkitehtuuri, prosessikuvaus, jatkokehitys

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author (authors)	Mikko Huotari
Thesis title	OSSI Application Modernization
Commissioned by	Etelä-Savon Koulutus Oy
Time	2025
Pages	45 pages
Supervisor	Tuomas Reijonen

ABSTRACT

This thesis was commissioned by Etelä-Savon Koulutus Oy, and it is based on the modernization project of the student competence and project management system OSSI, which started during my internship. The topic of the thesis is to create an analytical process description of the modernization of OSSI with modern technologies. The thesis discusses the technologies of the old and new application, the architecture of the application, the reasons for the need for modernization, and the implementation phase of the work. Finally, possible areas for further development of the application are discussed.

The modernization was based on the desire to offer students the opportunity to participate in the process of new application development, as well as to better prepare them for the requirements of today's working life in terms of technologies. In addition, the architecture of the old OSSI, and the technologies used, were becoming obsolete, which would have led to a large number of code rewritings.

The modernization utilized, among other things, Docker container technology to facilitate the management and deployment of services, as well as the transition to modern JavaScript frameworks in the construction of the user interface. The thesis describes the benefits of the selected technologies, the implementation phases of the project and their solutions.

The aim of the thesis is to provide a usable process description for the modernization of applications such as OSSI and to produce valuable documentation for Etelä-Savon Koulutus Oy to support further development. As a result of the modernization of the OSSI application, the maintainability, scalability and performance of the application will improve significantly, which supports the goals of the OSSI project of Etelä-Savon Koulutus Oy. In addition, students' competence in modern technologies increases, which promotes their employment opportunities and better meets the ever-changing needs of working life. The development of the application is in progress at the time of writing this thesis, which is why the content of the thesis corresponds to the current state of the application.

Keywords: application development, software architecture, process description, further development

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä keskitytään opiskelijoiden osaamisen- ja projektienhallintajärjestelmän modernisoinnin prosessiin. Työn tarkoituksena luodaan analysoiva prosessikuvaus, jossa käydään läpi vanhan sovelluksen käytössä olevat teknologiat, syyt modernisoinnille, uuteen sovellukseen valitut teknologiat, lyhyt kuvaus tähänastisesta toteutuksesta sekä jatkokehitysmahdollisuudet. Työssä perehdytään uuden, kehitteillä olevan version suunnitteluun ja sovellusarkkitehtuuriin. Lisäksi tehdään katsaus käytössä olleisiin kehitystyökaluihin, versionhallintaan sekä pilvipalveluihin.

Projektia toteutetaan Etelä-Savon Koulutus Oy:lle, ja sen kehityksestä vastaavat pääsääntöisesti oppilaitoksen opiskelijat yhteistyössä opettajien kanssa. Oman syventävän harjoitteluni vietin projektin alkuvaiheessa asiakaspuolen kehitystiimin jäsenenä.

2 SOVELLUKSEN JA TYÖN TARKOITUS

OSSI-sovellus toimii verkkoalustana Etelä-Savon Koulutus Oy:n tieto- ja viestintätekniikan opiskelijoille sekä henkilökunnalle. Sovelluksen ensisijainen tarkoitus on seurata opiskelijan osaamisen karttumista. Lisäksi se toimii muun muassa työajanseurannassa, projektien suorittamisessa sekä viestintäkanavana opiskelijan, oppilaan sekä työelämän edustajan välillä. Sovelluksella toisin sanoen seurataan opiskelijan taivalta tutkinnon suorittamisen edetessä.

Työn tarkoituksena on kuvata OSSI-sovelluksen modernisointiprosessi analysoimalla käytettyjä teknologiota ja menetelmiä. Työn painopiste vastaa sovelluskehitystä syventävässä harjoittelussani Etelä-Savon Koulutus Oy:llä työtehtävien painottuessa asiakaspuolen kehittämiseen. Opinnäytetyön lopputuloksesta Etelä-Savon Koulutus Oy saa analyysin tekniikoiden taustalta ja näkökulmia erityyppisten valintojen varalta tulevaisuutta ajatellen. OSSI-sovelluksen kehitys on tätä opinnäytetyötä kirjoittaessa kesken.

3 VANHAN SOVELLUKSEN ANALYSOINTI

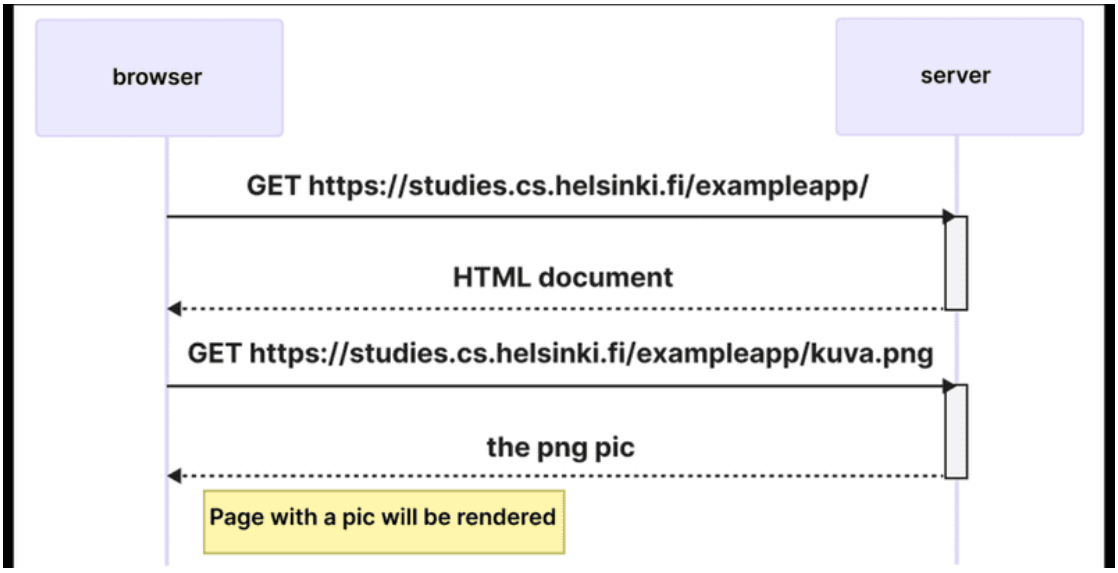
Tässä kappaleessa tutustutaan vanhassa OSSI-sovelluksessa käytössä olleisiin teknologioihin. Lisäksi kerrotaan yleistietoa käytetyistä teknologioista sekä tutkitaan mahdollisia haasteita kyseisen teknologian osalta, kun vastassa on nopeasti kehittyvä ympäristö. Kappaleen lopuksi käydään läpi vanhan sovelluksen rajoitteita ja kehitystarpeita.

3.1 Vanhan sovelluksen asiakaspuoli

Sovelluksen asiakaspuolella tarkoitetaan verkkosivun tai sovelluksen näkyvää osaa, jonka kanssa käyttäjä on vuorovaikutuksessa. Yleisimmät asiakaspuolen ohjelmointiin käytettävät ohjelmointikielet ovat HTML, CSS ja JavaScript. Asiakaspuolen tehtävänä on määrittää, millaisia elementtejä verkkosivu tai -sovellus pitää sisällään ja mitä niistä tapahtuu esimerkiksi käyttäjän klikatessa elementtiä. Asiakaspuolen kehitys liittyy vahvasti muun muassa UX/UI-suunnitteluun, palvelumuotoiluun ja saavutettavuuteen. (Itewiki s.a.)

UI (User Interface) eli käyttöliittymä tarkoittaa määritelmänsä mukaan kohtaa, jonka kautta käyttäjä on palvelun, sovelluksen tai verkkosivun kanssa vuorovaikutuksessa (Virtanen 2016). Virtanen kertoo myös, että brändäys, värit, fontit ja efektit ovat osa UI:ta, kun taas UX (User Experience) vastaa siitä, mitä tunteita UI:n käyttäminen herättää.

Käyttöliittymän toimintaa vanhassa web-sovelluksessa voidaan kuvata, kuten on tehty seuraavassa Full stack openin (s.a.) kuvassa (kuva 1), joka esittää selaimen ja palvelimen välistä kommunikaatiota.



Kuva 1. Selaimen ja palvelimen välistä kommunikaatiota (Full stack open s.a.)

Kuvan 1 esimerkissä käyttäjä lähettää selaimellaan GET-pyynnön palvelimelle, joka palauttaa kyseistä pyyntöä vastaavan HTML-dokumentin. Kun dokumentti on ladattu selaimeen, voidaan tässä kohtaa puhua kyseisen sovelluksen käyttöliittymästä. Käyttäjä klikkaa haluamaansa tekstiä tai painiketta, jonka jälkeen palvelin palauttaa kyseiseen valintaan perustuen sille määritellyn toiminnon, tässä tapauksessa kuvan PNG-formaatissa, joka renderöidään käyttöliittymään. Uudemmilla web-tekniikoilla jokaista klikkausta ja pyyntöä ei tarvitse lähettää palvelimelle, vaan ne pystyvät hyödyntämään elementtien eri tiloja.

Shah (2023) kertoo seuraavien ohjelmointikielten olevan yleisiä sovelluksen asiakaspuolella:

HTML
HyperText Markup Language: HTML on verkkosivujen perusta, joka määrittelee sivun rakenteen ja sisällön. Se sisältää perusrakennuspalikat, kuten otsikot, kappaleet, kuvat ja linkit.

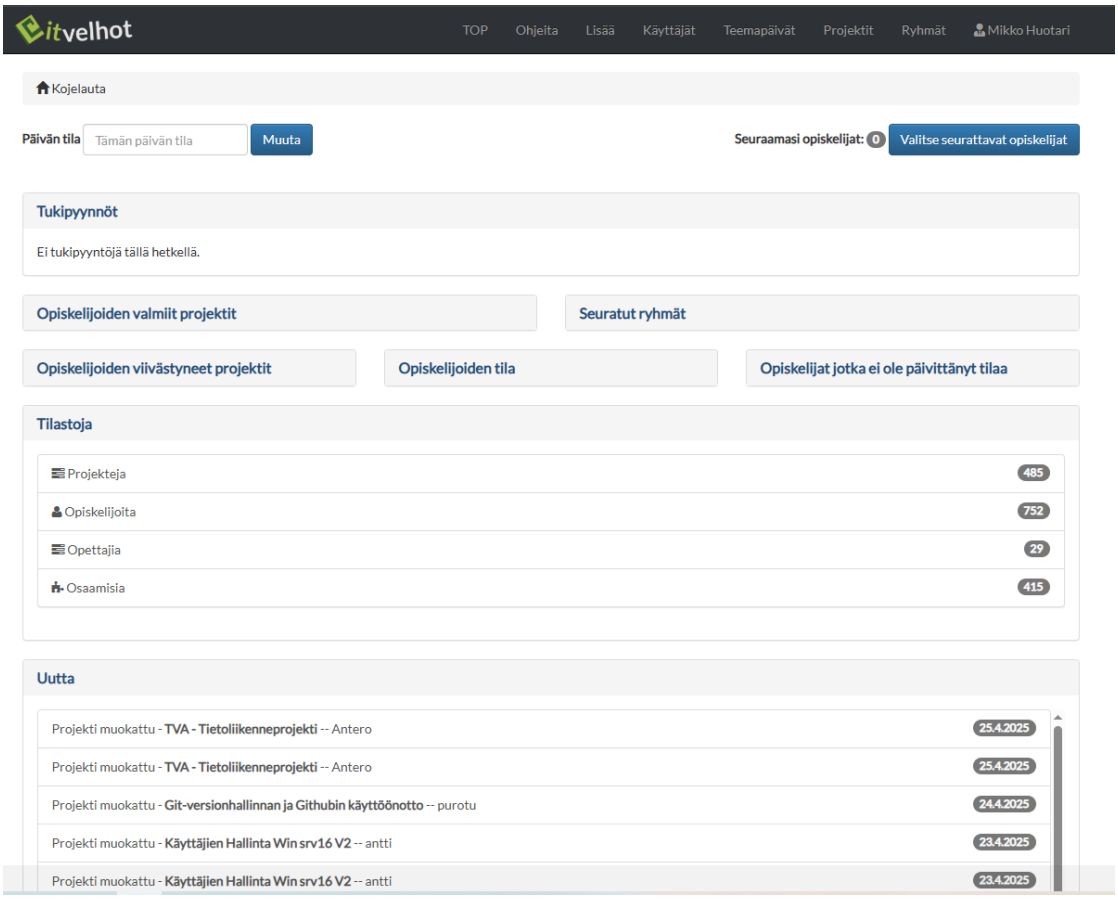
CSS
Cascading Style Sheets: CSS vastaa verkkosivujen muotoilusta ja esitystavasta. Se ohjaa HTML-elementtien ulkoasua, mukaan lukien fontit, värit, koot ja sijainti.

JavaScript:

JavaScript lisää verkkosivuille vuorovaikutteisuutta ja dynaamista käyttäytymistä. Sen avulla käyttäjät voivat olla vuorovaikutuksessa verkkosivuston kanssa, kuten napsauttaa painikkeita, lähettää lomakkeita ja käsitellä sisältöä.

Asiakaspuolen kehykset:
Suosittuja käyttöliittymäkehyksiä ovat React, Vue.js ja Angular. Nämä kehykset tarjoavat jäsennellyn lähestymistavan monimutkaisten ja uudelleenkäytettävien käyttöliittymäkomponenttien rakentamiseen.

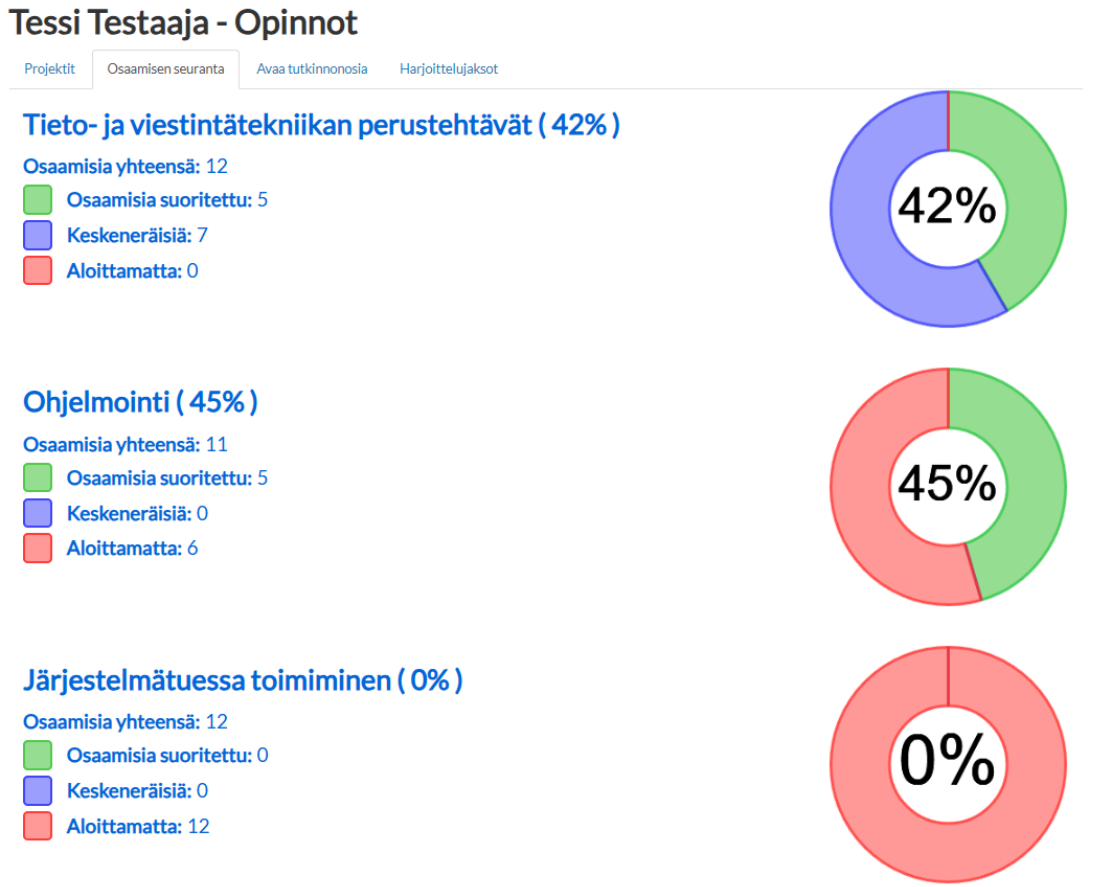
Otetaan seuraavaksi katsaus vanhan OSSIn käyttöliittymään. Kuvassa 2 on näkymä vanhan OSSI-sovelluksen etusivusta. Käyttöliittymä on pelkistetty.



Kuva 2. Vanhan OSSI-sovelluksen etusivu

Kuvasta 2 todetaan sivuston toiminnallisuuden olevan käyttötarpeeseen riittävä, mutta ulkoasuun ja käyttäjän käyttökokemukseen on nykytekniikoin mahdollista tuoda merkittävää parannusta.

Kuvassa 3 esitellään opiskelijoiden osaamisen seuranta -välilehteä. Kuvien 2 ja 3 on tarkoitus auttaa hahmottamaan vanhan sovelluksen ulkoasua ja lisäämään tietoa vanhan sovelluksen käyttöliittymästä.



Kuva 3. Opiskelijan osaamisen seuranta -välilehti vanhassa OSSI-sovelluksessa

Kuvassa 3 on CodeIgniterilla luotu dataan perustuvaa graafista näkymää käyttöliitymälle.

3.2 Javascript

JavaScript otettiin käyttöön vuonna 1995 keinona lisätä ohjelmia verkkosivuille Netscape Navigator -selaimessa. Kieli on sittemmin otettu käyttöön kaikissa muissa tärkeimmissä graafisissa verkkoselaimissa. Se on mahdollistanut nykyaikaiset verkkosovellukset – eli sovellukset, joiden kanssa voi olla suoraan vuorovaikutuksessa ilman, että sivua ladataan uudelleen jokaista toimintoa varten. JavaScriptiä käytetään myös perinteisemmillä verkkosivustoilla tarjoamaan erilaisia interaktiivisuuden ja älykkyyden muotoja. (Haverbeke 2024.)

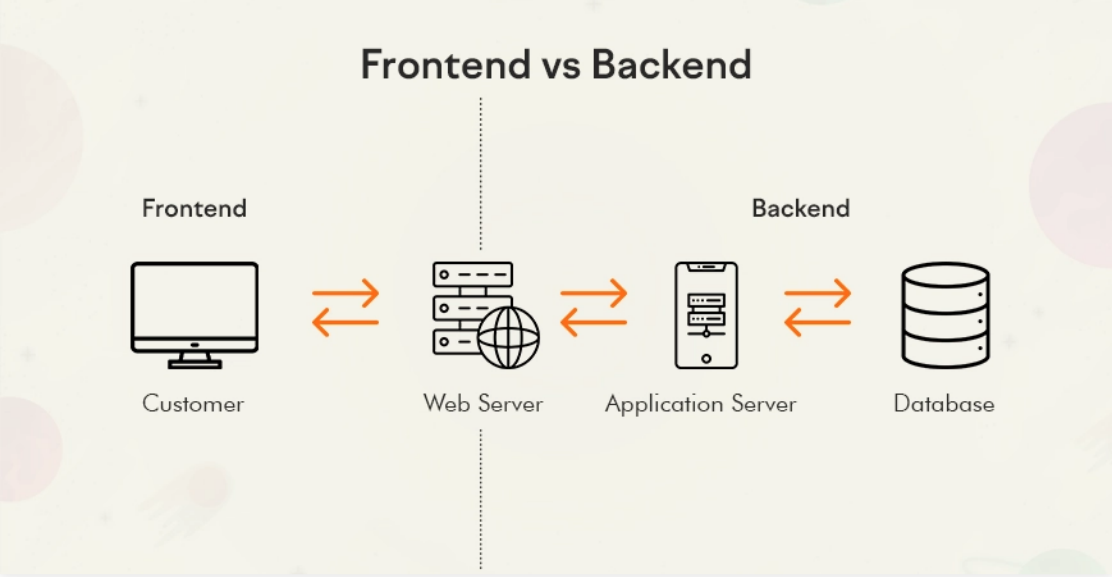
Haverbeke (2024) kertoo Javascriptin olevan erittäin joustava ja aloittelijaystävällinen ohjelmointikieli, mutta kääntöpuolena ongelmien paikantamisesta tulee merkittävästi haastavampaa juuri sen joustavuuden takia. Lisäksi virheet,

jotka toisen, jäykemmän kielen kääntäjä huomaisi jo ennen ohjelman suorittamista, eivät Javascriptillä nouse yhtä helposti esiin.

3.3 Vanhan sovelluksen palvelinpuoli

Yksinkertaistettuna verkkosivuston palvelinpuoli on kaikki se, mitä käyttäjä ei näe. Se vastaa käyttäjän lähettämiin pyyntöihin lähettämällä tietoja palvelimelta käyttöliittymään näytettäväksi. Verkkosivuston palvelinpuoli koostuu esimerkiksi palvelimista, tietokannoista, käyttöjärjestelmistä, sovellusliittymistä ja muusta, jotka kaikki yhdessä varmistavat, että käyttäjälle tarjotaan oikeat tiedot mahdollisimman nopeasti. (Airfocus. s.a.)

Kuvassa 4 sovelluksen eri osien toimintaa voidaan tiivistetysti kuvata näin:
Asiakas on käyttäjän näkemä osa sovelluksesta (esim. selainkäyttöliittymä);
Palvelinpuoli on näkymätön osa, joka käsittelee sovelluslogiikan ja datan;
Web-palvelin toimii välikätenä, joka ohjaa liikennettä käyttäjän ja palvelinpuolen välillä;
Tietokanta on tietojen pysyvä säilytyspaikka, johon palvelinpuoli kytkeytyy.



Kuva 4. Yleisnäkymä, kuinka web-sovelluksen eri osat toimivat yhdessä. (Shah 2023)

Shah (2023) kertoo palvelinpuolen ydintoiminnoiksi seuraavaa:

Käyttäjäpyyntöjen käsittely:

Taustakehittäjät luovat palvelinpuolen komentosarjoja, jotka käsittelevät käyttäjien pyyntöjä, kuten uusien käyttäjien rekisteröintiä, kirjautumisten käsittelyä ja tietojen hakemista tietokannoista.

Tietojen tallentaminen ja hallinta:
Taustakehittäjät suunnittelevat ja toteuttavat tietokantoja verkkosivuston tietojen, kuten tuotetietojen, käyttäjäprofiilien ja tilaustietojen tallentamiseen ja hallintaan.

Turvallisuuden varmistaminen:
Taustakehittäjät toteuttavat turvatoimia käyttäjätietojen suojaamiseksi ja luvattoman käytön tai tietomurtojen estämiseksi.

Viestintä ulkoisten palveluiden kanssa:
Taustakehittäjät integroituvat ulkoisiin palveluihin, kuten maksuyhdyskäytäviin, sähköpostipalveluntarjoajiin ja sosiaalisen median alustoihin, laajentaakseen verkkosivuston toimintoja.

3.3.1 Linux

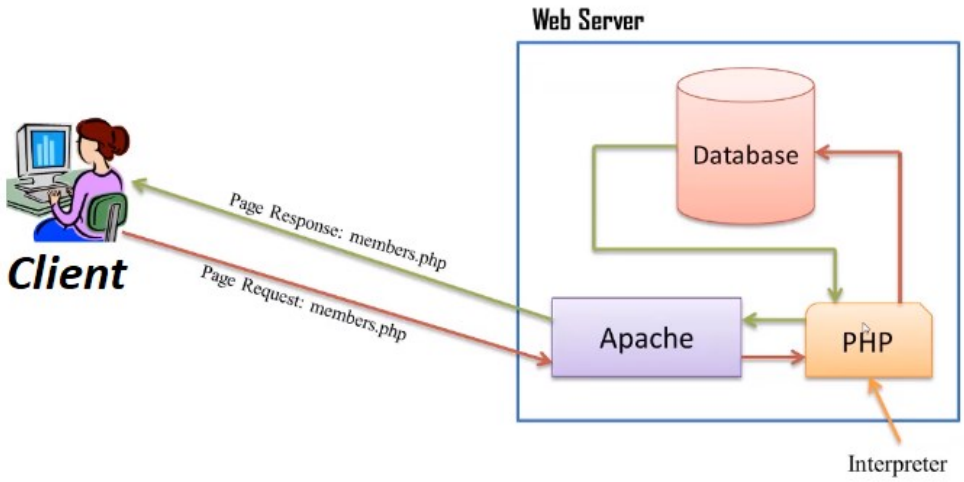
Linux on Unixiin perustuva avoimen lähdekoodin käyttöjärjestelmä, joka tunnetaan vakaudestaan, turvallisuudestaan ja joustavuudestaan. Se toimii käyttöjärjestelmänä useille laitteille tietokoneista ja palvelimista älypuhelimiin. Kernel-ydinkomponentti hallitsee laitteistoresursseja ja helpottaa ohjelmistoviestintää. Käyttäjät voivat mukauttaa ja jakaa omia versioitaan, joita kutsutaan jakeluiksi, kuten Ubuntua ja Fedoraa. Linuxia suositaan sen tehokkaan komentorivikäyttöliittymän ja laajan ohjelmointituen vuoksi. (Ha 2024.)

3.3.2 PHP ja CodeIgniter

Nimi PHP tulee englannin kielen sanoista PHP: Hypertext Preprocessor. PHP:n vahvuuksiin kuuluu sen laaja soveltuvuus eri käyttöjärjestelmille, kuten Windowsille, Linuxille, Unixille ja Mac OS X:lle. Se on yhteensopiva lähes kaikkien nykyään käytettyjen palvelimien kanssa. Koodi suoritetaan palvelimen puolella. (W3Schools s.a.)

Kuvassa 5 Asiakas pyytää verkkosivua palvelimelta, palvelin lähettää pyynnön PHP-tulkille, joka muuntaa pyynnön konekielelle ja etsii sitten kyseisen sivun

tietokannasta. Mikäli sivu löytyi, se palautuu PHP-tulkkiin, ja tulkki lähettää tiedot verkkopalvelimelle, jonka jälkeen verkkopalvelin palauttaa vastauksen asiakkaalle. (Kumar 2020.)



Kuva 5. Tiedonkulku asiakkaan, palvelimen, PHP-tulkin sekä tietokannan välillä (Kumar 2020)

W3Schools (s.a.) kertoo PHP:n vahvuuksista seuraavaa:

- Sen avulla voi luoda dynaamista sivustonsisältöä.
- PHP voi luoda, avata, lukea, kirjoittaa, poistaa ja sulkea tiedostoja palvelimella.
- Sen avulla voidaan kerätä lomaketietoja.
- Sillä voi lähettää ja vastaanottaa evästeitä.
- PHP voi lisätä, poistaa ja muokata tietokannan tietoja.
- PHP:tä voidaan käyttää käyttäjien käyttöoikeuksien hallintaan.
- PHP:n avulla voidaan salata tiedot.

Näistä tiedoista voidaan päätellä PHP:n soveltuvan sovelluskehitykseen vielä nykyäänkin.

CodeIgniter

CodeIgniter on PHP:llä kirjoitettu web-sovellusten kehys. CodeIgniterin suorituskykyyn suuntautunut suunnittelu näkyy PHP-kehyksen kevyessä rakenteessa. Tämä perustuu ohjelmistoarkkitehtuurimalliin Model View Controller

(MVC). MVC:n perusperiaate on ohjelmakoodin ja esitystavan tiukka erottaminen. Tämä saavutetaan modulaarisen ohjelmistorakenteen ja PHP-koodin ulkoistamisen avulla. Siinä on kolme keskeistä osaa: tietomalli (Model), esitystapa (View) ja ohjain (Controller). (IONOS 2023.)

Alla MVC:n toimintaperiaatteet on kuvattu IONOSin (2023) mukaan seuraavasti:

- **Tietomalli** (Model) edustaa CodeIgniterin pohjalta kehitetyn verkkosovelluksen tietorakennetta. Tätä tarkoitusta varten malliluokat määritellään lähdekoodissa. Näitä ovat erikoistoiminnot, joiden avulla tietokannan tietoja voidaan käyttää, tallentaa tai päivittää.

- **Esitys** (View) on sovelluksen osa, joka esitetään käyttäjille. Yleensä tämä on HTML-asiakirja, johon sisältö integroidaan dynaamisesti PHP:n kautta. Näkymä on pohjimmiltaan eräänlainen malli. CodeIgniter tarjoaa mahdollisuuden määritellä näkymässä verkkosivun elementtejä, kuten ylä- ja alatunnisteen tai RSS-sivustot. Yleensä verkkosovellukset käyttävät useita näkymiä viittaamaan sisältöön samaa tietomallia käyttäen. Tämä mahdollistaa ohjelman eri ominaisuuksien esittämisen eri näkymissä.

- **Ohjain** (Controller) toimii välittäjänä mallin, näkymän ja minkä tahansa muun resurssin välillä, jota tarvitaan HTTP-pyynnön käsittelyyn tai verkkosivuston dynaamiseen luomiseen. Tämä komponentti ottaa vastaan saapuvia pyyntöjä, vahvistaa syötteen, valitsee halutun näkymän ja välittää sisällön, jonka tietomalli on ladannut tietokannasta.

Vanhan OSSI-sovelluksen suurin yksittäinen modernisoinnin syy syntyi juuri CodeIgniter kehyksen päivitystarpeesta.

3.3.3 Apache

Apache on yksi maailman yleisimmin käytössä olevista web-palvelinohjelmistoista, jota käytetään verkkosivustojen sisällön välittämiseen. Sen tehtävänä on käsitellä selainten tekemiä pyyntöjä, toimittaa verkkosivut käyttäjälle ja tallentaa sivustojen sisältöä. Web-palvelin koostuu ohjelmistosta sekä sen taustalla toimivasta laitteistosta, joka vastaanottaa HTTP- tai HTTPS-protokollien kautta tulevia pyyntöjä. (Hess 2023.)

3.3.4 MySQL

MySQL on avoimen lähdekoodin relaationaalinen tietokannan hallintajärjestelmä, joka käyttää SQL:ää tietokantojen luomiseen ja hallintaan. MySQL tallentaa tiedot rivi- ja saraketaulukoihin, jotka on järjestetty skeemoihin. Rakenne määrittää, miten tiedot järjestetään, tallennetaan ja kuvataan eri taulukoiden välisissä suhteissa. Tämän formaatin avulla kehittäjät voivat helposti tallentaa, noutaa ja analysoida monia tietotyyppejä, kuten yksinkertaista tekstiä, numeroita, päivämääriä, kellonaikoja tai JSONia ja vektoreita. (Erickson 2024a.)

Erickson (2024a) kertoo, että tällainen looginen rakenne mahdollistaa nopean haun ja antaa kehittäjille joustavan tavan määritellä suhteet (yksi–yksi, yksi–moni jne.). Järjestelmä valvoo sääntöjen toteutumista, joten hyvin suunniteltuna se estää ristiriitaisen, päällekkäisen tai orvon datan syntymisen.

3.3.5 MariaDB Server

MariaDB Server on avoimen lähdekoodin ohjelmisto, ja relaatiotietokantana se tarjoaa SQL-rajapinnan tietojen käyttämiseen. MariaDB Server muuttaa tiedot jäsennellyiksi tiedoiksi monenlaisissa sovelluksissa. MariaDB Serveriä käytetään, koska se on nopea, skaalautuva ja kestävä. (MariaDB s.a.)

3.4 Sovelluksen rajoitteet ja kehitystarpeet

Sovellus oli rakennettu CodeIgniterilla, joka oli tuolloin kevyt ja laajasti käytetty PHP-kehys. Se tarjosi selkeän rakenteen MVC-mallin mukaisesti ja nopeutti kehitystä. Tämän lisäksi sovelluksen laajentuessa ja teknologian kehittyessä CodeIgniterin rajoitukset kuitenkin nousivat esiin, ja modernisointi nähtiin tarpeelliseksi. Tavoitteena oli tuoda sovellus lähemmäs nykyaikaisia vaatimuksia muun muassa paremman ylläpidettävyyden, laajennettavuuden ja tietoturvan näkökulmasta.

4 MODERNISOINTISUUNNITELMA

4.1 Sovelluksen vaatimukset

Ohjelmistoprojektin vaatimuksiin kuuluvat toiminnot, ominaisuudet ja rajoitukset, jotka lopputuotteen on täytettävä. Toisin sanoen vaatimukset määrittelevät, mitä ohjelmiston tulee tehdä, miltä sen tulee näyttää ja mitkä ehdot on täytettävä, jotta sitä voidaan pitää onnistuneena. (Visure Solutions s.a.)

Järjestelmän dokumentoinnissa vaatimukset on usein jaettu toiminnallisiin vaatimuksiin ja ei-toiminnallisiin vaatimuksiin. Toiminnallinen vaatimus kuvaa, mitä järjestelmän tulee voida tehdä, ja ei-toiminnallinen vaatimus järjestelmän ominaisuuksia tai laatuvaatimuksia.

Toiminnallinen vaatimus
Käyttäjä voi lähettää viestin toiselle käyttäjälle.

Ei-toiminnallinen vaatimus
Järjestelmän tulee kestää 600 yhtäaikaista käyttäjää.

OSSI-sovellukselle toteutettiin ”Käyttäjäryhmät ja toiminnot” -dokumentaatiota, mutta virallista vaatimusmäärittelydokumentaatiota ei luotu. Sovelluksen vaatimuksia voi kuvata esimerkiksi seuraavilla vaatimuksilla:

Toiminnalliset vaatimukset

1. Käyttäjän tulee voida kirjautua palveluun O365-tunnuksilla.
2. Opiskelija voi liittyä hänelle avattuihin projekteihin.
3. Opettaja voi hyväksyä osaamisia opiskelijalle.
4. Opiskelijan tulee voida kommentoida palautettua projektia.
5. Opiskelija voi lähettää opettajalle tukipyynnön.

Ei-toiminnalliset vaatimukset

1. Järjestelmän tulee olla tietoturvallinen.
2. Käyttöliittymän on oltava värimaailmaltaan saavutettava.
3. Järjestelmän tulee toimia kohtuullisella vasteajalla.
4. Järjestelmän tulee kestää 200 yhtäaikaista käyttäjää.
5. Käyttöliittymän teemat vastaavat organisaation ohjeistusta.

Käyttäjäryhmät ja toiminnot

Opettajat

- Opiskelijoiden hallinta
- Oppimistehtävien ja tutkinnonosien avaaminen
- Osaamisen kehittymisen seuranta
- Oppimiskokonaisuuksien (teemapäivät) luominen
- Tehtävien tarkastus ja palaute
- Työpaikkajaksojen hallinta ja viestintä työpaikkaohjaajien kanssa

Opiskelijat

- Oppimistehtävien tekeminen ja palauttaminen
- Tehtäviin käytetyn ajan kirjaaminen
- Osaamisen kehittymisen seuranta
- Työpäiväkirjamerkinnät työpaikalla
- Viestintä opettajien ja työpaikkaohjaajien kanssa
- Tukipyyntöjen tekeminen opettajille

Työpaikkaohjaajat

- Opiskelijan osaamisen kehittymisen seuranta
- Viestintä opiskelijan ja opettajien kanssa
- Palautteen antaminen työskentelystä

OSSI-sovellukselle luotiin suunnitteluvaiheessa alustava listaus vaatimuksista, joita mahdollisuuksien mukaan päivitettiin vastaamaan käyttökohteita ja sovelluksen tarpeita.

4.2 Ohjelmistoarkkitehtuuri

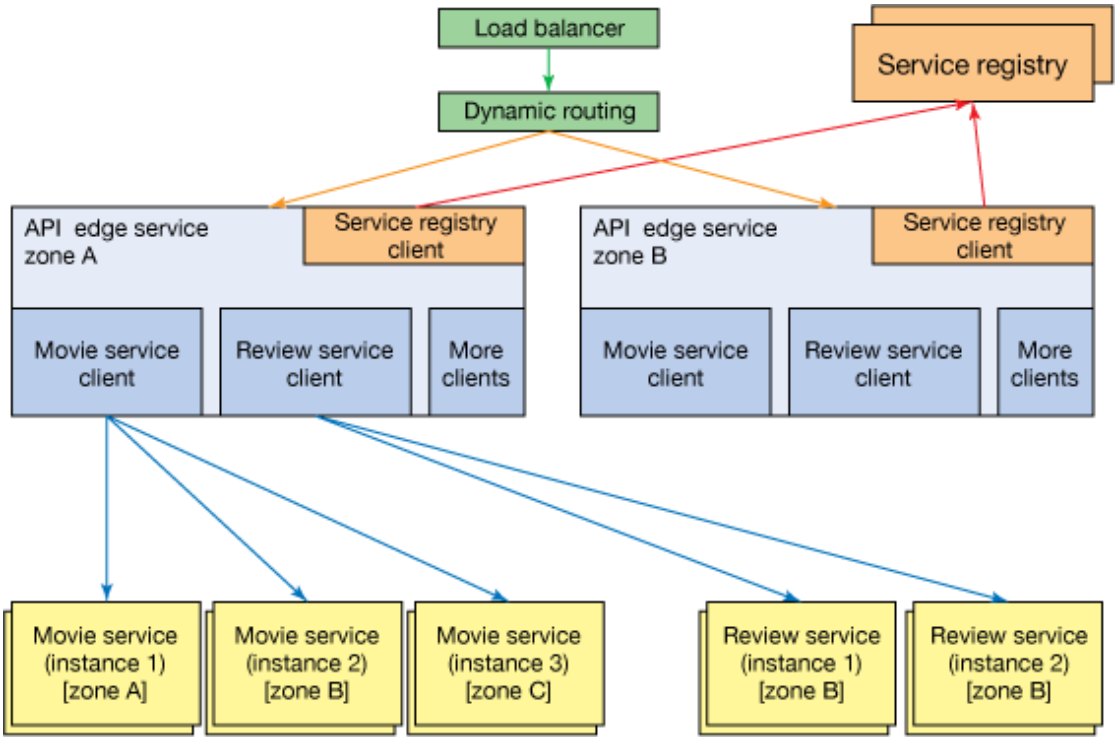
Järjestelmän ohjelmistoarkkitehtuuri edustaa järjestelmän yleiseen rakenteeseen ja käyttäytymiseen liittyviä suunnittelupäätöksiä. Arkkitehtuuri auttaa sidosryhmiä ymmärtämään ja analysoimaan, miten järjestelmä saavuttaa olennaiset ominaisuudet, kuten muokattavuuden, käytettävyyden ja turvallisuuden. Se käsittää esimerkiksi sovelluksen modulaarisen rakenteen, kerrokset, komponentit ja niiden väliset rajapinnat. (Carnegie Mellon University s.a.)

Ohjelmistoarkkitehtuuri edustaa järjestelmän varhaisimpia suunnittelupäätöksiä. Nämä varhaiset päätökset ovat vaikeimpia saada oikein ja hankalimpia muuttaa myöhemmin kehitysprosessissa, ja niillä on kauaskantoisimmat vaikutukset. Toisin sanoen, jos arkkitehtuurissa tehdään virheitä tai laiminlyöntejä, niiden korjaaminen myöhemmin on kallista ja hankalaa. (Bass, Clements & Kazman 2003.)

Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuurissa sovellus koostuu useista pienistä, itsenäisistä palveluista, joista kukin keskittyy yhteen tiettyyn liiketoimintoon tai alueeseen (Microsoft Learn 2023). Mikropalvelumallissa komponentit otetaan käyttöön itsenäisesti, ja ne kommunikoivat jonkinlaisen REST-, tapahtumavirtaus- ja viestivälittäjät -yhdistelmän kautta, joten on mahdollista, että jokainen yksittäinen sovelluksen osa voidaan optimoida kyseiselle palvelulle (Osowski 2024).

Kuvassa 6 on Osowskin (2024) luoma esimerkki mikropalveluarkkitehtuurista videontoistosovelluksessa. Jokainen kuvan laatikko hallinnoi tietoa, skaalautuu, tietää sijaintinsa sekä tunnistaa, mistä saa haettua tarvittavan tiedon itsenäisesti.



Kuva 6. Mikropalveluimplementaatio videontoistosovelluksesta (Osowski 2024)

Tällä tavalla mikropalveluarkkitehtuuri mallintaa sovelluksen joukoksi moduuleja, jotka voidaan tarvittaessa päivittää tai korvata itsenäisesti vaikuttamatta koko järjestelmään. Ozkaya (2023) kertoo mikropalveluarkkitehtuurin soveltuvan erityisesti, kun sovellukselta halutaan joustavuutta uusien toimintojen käyttöönotossa ilman käyttökatkoksia tai yksittäisten palveluiden itsenäistä skaalautuvuutta ja ketteryyttä.

Mikropalveluarkkitehtuurin heikkoudet

Mikropalveluarkkitehtuurin joustavuudella on kuitenkin hintansa. Se lisää merkittävästi järjestelmän monimutkaisuutta. Reselman ja McKenzie (2023) listaavat mikropalveluarkkitehtuurin heikkouksiksi muun muassa:

- Monimutkaiset integraatiot ja riippuvaisuudet.
- Järjestelmää voi olla vaikeaa testata.
- Vikojen paikannus on haastavampaa.
- Tiedonkääntöongelmat REST-rajapintojen välillä.

Tästä voidaan vetää johtopäätös, että mikropalveluarkkitehtuuri ei ole paras vaihtoehto, mikäli kyseessä on yksinkertainen sovellus tai mikäli kehitystiimiin

jäsenet ovat kokemattomia. Sen sijaan monimutkaisemmissa sovelluksissa ja kokeneen tiimin käsissä mikropalveluarkkitehtuuri on hyvä valinta.

4.3 Asiakaspuolen suunnitelma

Tässä kappaleessa kerrotaan sovelluksen asiakaspuolen käyttöön valikoiduista teknologioista. Teknologiat on selitetty tarkemmin omissa luvuissaan. OSSI-sovelluksen asiakaspuoli päätettiin uudistaa moderneilla teknologioilla käyttäen React-kirjastoa ja TypeScript -ohjelmointikieltä. Tämä valinta tehtiin tukemaan kehittäjäjäseninä toimineiden opiskelijoiden valmiuksia vastata työelämän vaatimuksia. Sovelluksen sisällä on käytössä lukuisia tekstikenttiä, joihin haluttiin tietyt vaatimukset täyttävä tekstieditori. Tekstieditorin tuli tukea perinteisiä tekstinkorostustoimintoja, upotettuja videoita sekä merkkauskieltä.

4.3.1 React ja Typescript

React on JavaScript-kirjasto deklaratiivisten, tehokkaiden ja konfiguroitavien käyttöliittymien rakentamiseen. Sen avulla voidaan rakentaa monimutkaisia käyttöliittymiä "komponenteista", jotka ovat pieniä, itsenäisiä koodinpätkiä. TypeScript puolestaan on oliopohjainen ja tiukasti tyypitetty ohjelmointikieli, joka on JavaScriptin yläjoukko. TypeScriptillä kirjoitettu koodi muunnetaan JavaScriptiksi, jota voidaan käyttää missä tahansa JavaScriptiä tukevassa ympäristössä, kuten selaimissa ja omissa sovelluksissa. (Simplilearn 2025.)

Sovellustenkehitys Reactilla ja TypeScriptillä on nykypäivänä suosittu yhdistelmä. Barić (2024) kertoo TypeScriptin käyttämisen vahvuuksista Reactin kanssa muun muassa seuraavaa:

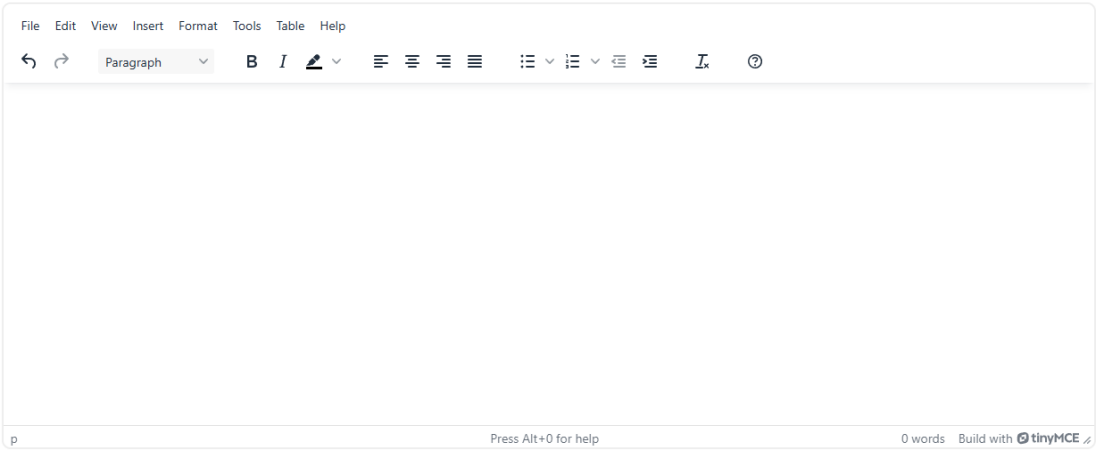
- **Tyyppiturvallisuus**: Yksi TypeScriptin suurimmista eduista on tyyppitarkistaminen. Määrittelemällä tyypit selkeästi voidaan virheet havaita jo kehitysprosessin varhaisessa vaiheessa, mikä tekee koodista vähemmän altista virheille.
- **Parannettu refaktorointi**: Vahva tyypitys tekee koodin refaktoroinnista turvallisempaa. Kääntäjä voi varoittaa, jos muutokset rikkovat olemassa olevia toimintoja, mikä vähentää ajonaikaisten virheiden todennäköisyyttä.

- **Skaalautuvuus**: TypeScript on hyödyllinen suuremmissa sovelluksissa, joissa on tärkeää ylläpitää yhdenmukaisia tietotyyppejä useissa komponenteissa ja palveluissa.

Yhteenvetona TypeScriptillä työskentelemistä voidaan todeta sen olevan koodin kirjoitusvaiheessa hitaampaa, mutta jälkikäteen selvitettäviä virheitä tulee vastaan huomattavasti vähemmän.

4.3.2 TinyMCE

TinyMCE on tekstieditori, jonka avulla käyttäjät voivat luoda muotoiltua sisältöä käyttäjäystävällisessä käyttöliittymässä (Tiny s.a.). Sen avulla sovelluksen käyttöliittymän tekstikenttiin tuodaan esimerkiksi fontin vaihto, tekstin lihavointi, kursivointi ja erilaiset luettelointityylit ilman erillistä asiakaspuolen koodin kirjoittamista. Lisäksi TinyMCE:n laajennuksilla on mahdollista lisätä esimerkiksi mediaelementtejä, kuten kuvia, linkkejä tai videoita.

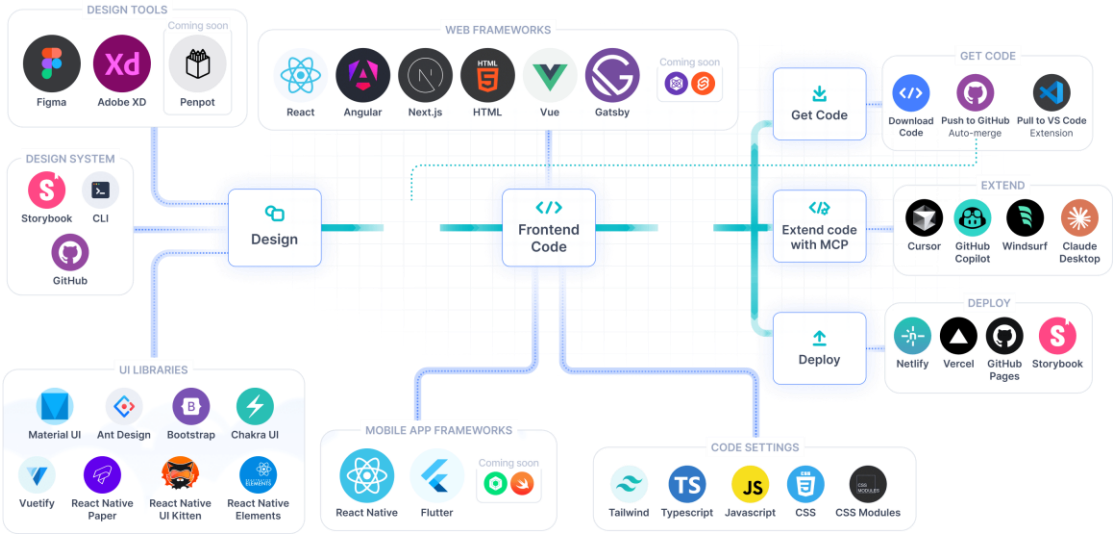


Kuva 7. Käyttöliittymän tekstikenttään tuotu TinyMCE-tekstieditori

Kuvasta 7 huomataan, että TinyMCE:llä on paljon mahdollisuuksia laajennusten suhteen. TinyMCE:n laajennuksilla voidaan toteuttaa esimerkiksi merkkauskielen käsitteleminen, jolloin haluttua dataa voidaan säilyttää tietokannassa muistin kannalta pienemmässä muodossa.

4.3.3 Käyttöliittymän suunnittelu Figmalla

Figma on käyttöliittymien suunnitteluun tarkoitettu työkalu. Sen avulla voidaan luoda malleja ja prototyyppejä verkkosivuston ulkoasusta ja painikkeiden toiminnallisuudesta sivustolla navigoitaessa. Useampi käyttäjä voi samanaikaisesti tarkastella ja muokata näkymää, mikä tekee siitä kätevän työkalun ryhmässä työskentelemiseen. Figmasta löytyy kattava kirjasto yleisesti käytössä olevista käyttöliittymä-komponenteista, sekä mahdollisuus toiminnallisuuden laajentamiseen eri integraatioiden avulla.



Kuva 8. Sovelluskehityksen virtaus suunnittelusta tuotantoon (Locofy s.a.)

Kuvasta 8 voidaan toteuttaa esimerkkinä tilanne, jossa sovelluksen käyttöliittymän suunnittelu on päätetty toteuttaa Figmalla React Material UI:n komponenttien avulla:

1. UI-suunnittelija tekee käyttöliittymän prototyypin Figmaan.
2. Suunnittelija käyttää Material UI:n visuaalisia komponentteja Figmassa.
3. Valmis malli voidaan ulkonäön ja toiminnallisuuden osalta hyväksyttää.
4. Luodaan hyväksytystä mallista valmis koodipohja integraation avulla.

Esimerkkiä voidaan viedä pidemmälle aina tuotantoon saakka, mutta keskeisenä ajatuksena haluan tuoda esiin työn tehostamisen mahdollisuuden jo suunnittelusta lähtien.

4.4 Palvelinpuolen suunnitelma

4.4.1 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokanta, joka tukee sekä SQL (relaatio) että- JSON (ei-relaatio) -kyselyjä. Se on erittäin vakaa tietokannan hallintajärjestelmä, jota tukee yli 20-vuotinen yhteisöllinen kehitys. PostgreSQL:ää käytetään tietovarastona monille verkko-, mobiili-, analytiikka- ja geospatiaalisille sovelluksille. (Amazon AWS s.a.)

PostgreSQL on tehokas ja laajasti käytetty relaatiotietokanta, joka tarjoaa vahvan tuen transaktioille, kattavan turvallisuusominaisuuksien joukon sekä laajan muokattavuuden esimerkiksi lisäosien ja ohjelmointikielien kautta. Sen parametrit ovat joustavasti säädettävissä eri tasoilla, ja järjestelmä soveltuu hyvin monimutkaisten sovellusten taustalle. (Dhruv 2024.) Haittapuoliksi Dhruv (2024) mainitsee rakenteen jäykkyyden verrattuna NoSQL-tietokantoihin, mahdolliset suorituskykyongelmat suurilla tietomäärillä, varmuuskopioinnin haasteet sekä ajoittaisen tarpeen erikoistuneille ohjelmisto- tai laitteistovaatimuksille.

4.4.2 MongoDB

MongoDB on suosittu avoimen lähdekoodin dokumenttitietokanta, jota käytetään laajalti nykyaikaisissa verkko- ja mobiilisovelluksissa. Se on luokiteltu NoSQL-tietokannaksi, mikä tarkoittaa, että se käyttää dokumenttilähtöistä lähestymistapaa tietojen tallentamiseen perinteisen taulukkopohjaisen relaatiomenetelmän sijaan. (Erickson 2024b.)

Dokumenttitietokannalla Erickson (2024b) tarkoittaa MongoDB:n tapaa säilyttää, hakea ja muokata tietoja. Tieto tallennetaan JSON-tyyppisen tiedon binääriversio BSON:nä kokoelmiin.

Ohjelmointikielet, joita MongoDB tukee:

- Node.js
- C
- C++

- C#
- Go
- Java
- Perl
- PHP
- Python
- Ruby
- Rust
- Scala
- Swift

MongoDB tarjoaa joustavan skeeman, korkean suorituskyvyn ja skaalautuvuuden. Lisäksi se tukee reaaliaikaista datankäsittelyä, monimutkaisia tietorakenteita ja integroituu hyvin eri ohjelmointikieliin ja kehitysalustoihin. Näiden ominaisuuksien ansiosta se soveltuu erityisesti sovelluksiin, joissa vaaditaan nopeutta, joustavuutta ja suurten tietomäärien käsittelyä. Haittapuolina ovat rajoitettu ACID-tuki useiden dokumenttien välillä, suuri muistin kulutus, operaatioiden tehottomuus sekä monimutkainen sharding-konfigurointi. (GeeksforGeeks 2025.)

4.4.3 GraphQL

GraphQL on ohjelmointirajapintojen kyselykieli ja suorituspalvelu näiden kyselyjen täyttämiseen olemassa olevilla tiedoilla. GraphQL tarjoaa ymmärrettävän kuvauksen API:n tiedoista, sekä antaa asiakkaille mahdollisuuden pyytää juuri sitä tietoa, mitä he tarvitsevat. (Hygraph s.a.)

Hygraphin mukaan GraphQL-kyselyt eivät pääse käsiksi vain yksittäisen resurssin kenttiin, vaan myös seuraavat niiden välisiä viittauksia. Vaikka tyypilliset REST-ohjelmointirajapinnat edellyttävät lataamista useista URL-osoitteista, GraphQL-sovellusliittymät saavat kaikki tiedot yhdellä pyynnöllä, mikä tekee sovelluksista nopeita myös hitaissa mobiiliverkkoyhteyksissä.

GraphQL ja REST ovat kaksi suosituinta API-kehityksen ja -integroinnin arkkitehtuuria, jotka helpottavat tiedonsiirtoa asiakkaiden ja palvelimien välillä.

REST-arkkitehtuurissa asiakas tekee HTTP-pyyntöjä eri päätepisteisiin, ja tiedot lähetetään HTTP-vastauksena, kun taas GraphQL:ssä asiakas pyytää tietoja kyselyillä yhteen päätepisteeseen. (Herrera 2025.)

Herrera (2025) kertoo GraphQL:n vahvuuksiksi muun muassa seuraavat ominaisuudet:

- **Vähentää ali- ja ylihakua**: Voidaan pyytää vain haluttua dataa ja vähentää siten yli- ja alihaun ongelmaa. Tämä optimointi lisää tehokkuutta ja vähentää samalla tarpeettomia kustannuksia.

- **Joustavuus ja mukautuvuus**: Nopea kenttien muokkaus ja lisäys rikkomatta olemassa olevia asiakkaita.

- **Eliminoi useita API-pyyntöjä**: Tarvittavia tietoja voidaan käyttää yhdellä kyselyllä. Tämän ansiosta useita API-pyyntöjä ei tarvitse tehdä. Tämä ei ainoastaan nopeuta ohjelmaa, vaan myös vähentää palvelimen kuormitusta ja tekee siitä skaalautuvamman.

- **Itsedokumentoituva skeema**: GraphQL sisältää tyyppijärjestelmän ja skeeman, joiden avulla kehittäjät ja tiimit voivat helposti tarkastella ja ylläpitää tietojen rakennetta.

Heikkouksiksi Herrera (2025) luettelee esimerkiksi:

- **Välimuistin haasteet**: Tavallinen HTTP-välimuisti on vähemmän tehokas, ja mukautetun välimuistin määrittäminen voi olla vaikeaa.

- **Turvallisuusongelmat**: Joustavat kyselyominaisuudet voivat aiheuttaa tietoturvaongelmia, jos niitä ei käsitellä oikein.

- **Jyrkkä oppimiskäyrä**: Vaikka GraphQL:llä on etunsa, sen käyttöönotto ja hallinta voi olla monimutkaisempaa verrattuna REST-sovellusliittymiin. Uudet kehittäjät saattavat tarvita jonkin aikaa ymmärtääkseen sen käsitteet ja parhaat käytännöt.

- **Osittain standardoitu**: GraphQL:ssä ei ole sisäänrakennettuja standardeja esimerkiksi virheiden ja moniosaisten lomatietojen käsittelyssä. Joissakin tapauksissa oman logiikan toteuttaminen tai kolmannen osapuolen kirjastojen käyttäminen voi olla tarpeen näiden puutteiden korjaamiseksi.

Tästä voidaan todeta, että GraphQL soveltuu tehokkaaseen ja joustavaan tiedonhakuun monimutkaisissa järjestelmissä. Haasteita voivat kuitenkin aiheuttaa välimuistin hallinta, tietoturva sekä teknologian omaksumisen vaativuus.

4.4.4 Autentikointi

Autentikointi on prosessi, jonka avulla varmistetaan, että ihmiset, palvelut ja sovellukset, joilla on oikeat käyttöoikeudet, voivat saada pääsyn organisaation resursseihin (Microsoft s.a). Microsoftin mukaan autentikointiprosessi sisältää kolme päävaihetta:

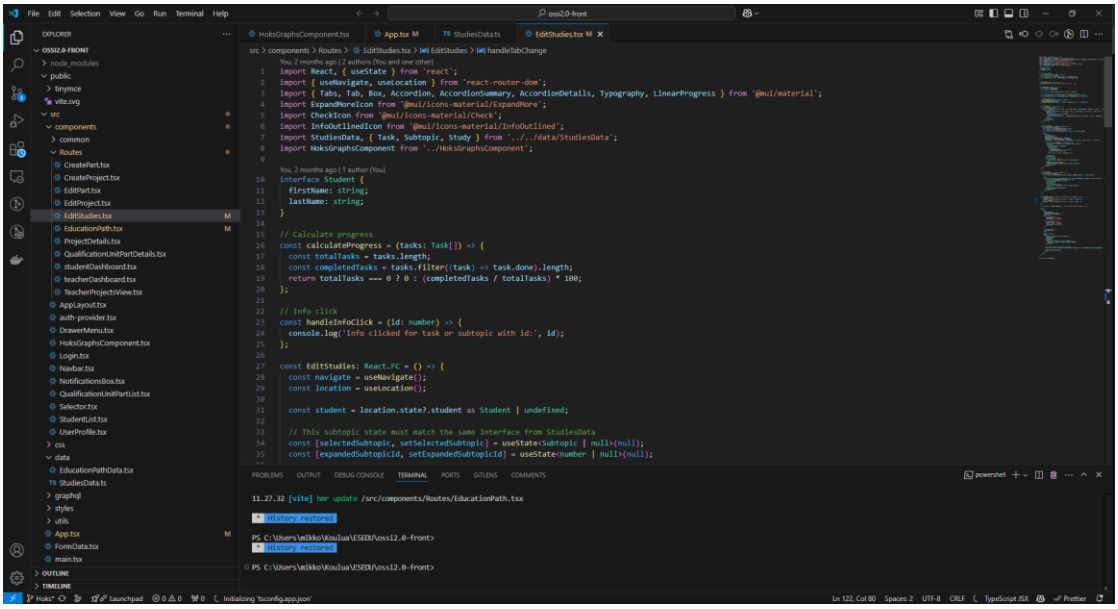
- **Tunnistamisen**: Käyttäjät määrittävät yleensä käyttäjätunnuksen avulla, keitä he ovat.
- **Todennuksen**: Tyypillisesti käyttäjä todistaa olevansa se, joka hän sanoo olevansa syöttämällä salasanan, mutta turvallisuuden vahvistamiseksi monet organisaatiot vaativat myös, että käyttäjä todistaa henkilöllisyytensä mobiilisovelluksen koodilla tai vastaavalla lisävarmenteella (MFA).
- **Valtuutuksen**: Järjestelmä tarkistaa, että käyttäjillä on oikeudet järjestelmään, jota he yrittävät käyttää.

Kun puhumme todennuksesta, tarkoitamme prosessia, jossa varmistetaan, että joku on se, joka hän sanoo olevansa. Valtuutus puolestaan on prosessi, jolla varmistetaan, että henkilöllä on oikeus tehdä jotain. (Netguru 2025.)

5 KEHITYSTYÖKALUT, VERSIONHALLINTA & PILVIPALVELUT

5.1 Visual Studio Code

Visual Studio Code (VS Code) on Microsoftin kehittämä kehitysympäristö sovelluskehittäjiä varten. Sen avulla esimerkiksi kirjoitetaan koodia, hallinnoidaan sovelluksen tiedostoja ja ajetaan komentoja terminaalissa. Kuvassa 9 on Visual Studio Coden kehitysympäristön näkymä, jossa ohjelmointikoodia kirjoitetaan.

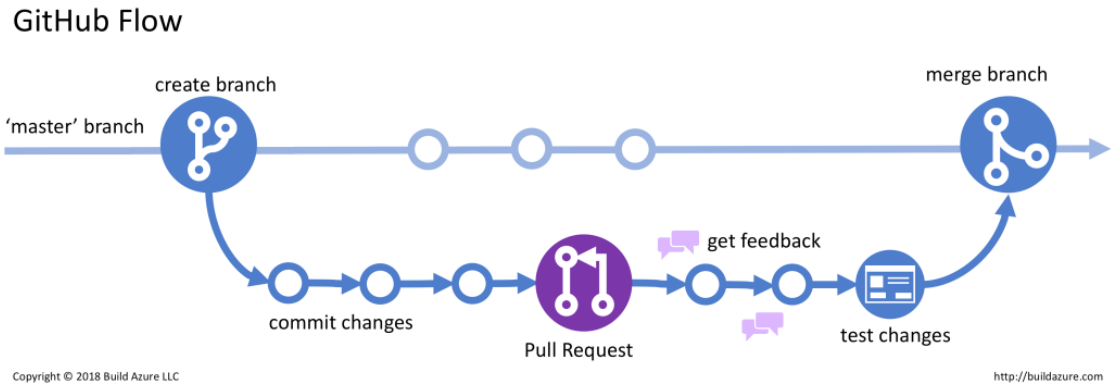


Kuva 9. Näkymä Visual Studio Codesta

Kuvassa 9 käyttöliittymässä käytössä on tummatila. Vasemmalla sivulla nähdään projektin tiedostorakenne, oikealla suuremmassa ikkunassa muokattava ohjelmointitiedosto ja alhaalla ohjelman terminaali, josta voidaan syöttää komentoja.

5.2 Versionhallinta – GIT ja GitHub

GIT on versionhallintajärjestelmä, joka tallentaa muutokset tiedostoihin ja mahdollistaa usean kehittäjän työskentelyn saman projektin parissa (GitHub Docs s.a). Tyypillinen sovellusprojekti rakentuu haarojen (branch) ympärille, mikä tarkoittaa, että päähaarasta (main/master branch) luodaan oma haara, jossa muutoksia voi tehdä itsenäisesti vaikuttamatta päähaaraan. Tiimityöskentelyssä kehitetään usein useampaa ominaisuutta samanaikaisesti, jolloin jokaiselle ominaisuudelle voidaan luoda oma haara. Kun ominaisuus on saatu valmiiksi, yhdistetään haara päähaaraan. GIT tallentaa versiot, jotta myöhemmin projektin eri versioita voidaan vertailla tai palauttaa käyttöön.



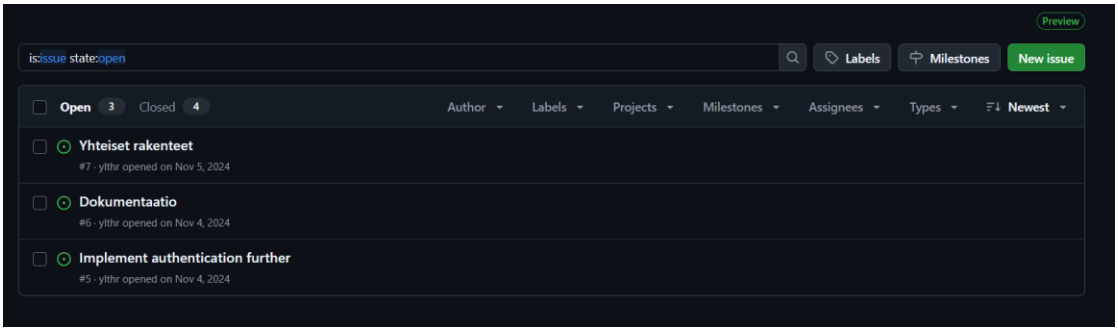
Kuva 10. GIT-versionhallinnan työnkulun hallinta (Pietschmann 2018)

Kuvassa 10 esitetään GitHub Flow -menetelmä, joka kuvaa ohjelmistokehityksen versionhallinnan tyypillistä työnkulkua käyttäen GitHubia. Prosessi auttaa hallitsemaan muutoksia selkeästi, parantaa yhteistyötä, ja varmistaa, että koodi pysyy vakaana ja toimivana koko ajan.

GitHub

GitHub on pilvipohjainen alusta koodin tallennukseen, jakamiseen ja työskentelyyn muiden kanssa (GitHub Docs s.a.). GitHub tarjoaa alustana kattavan valikoiman erilaisia työkaluja kehitystiimin käyttöön. Code-näkymä näyttää projektin tiedostot, kansiorakenteen sekä mahdollistaa navigoinnin ja tarkastelun eri haarojen välillä, kun taas Issues toimii tehtävienhallintajärjestelmänä, johon voidaan kirjata esimerkiksi bugeja ja kehitysehdotuksia sekä osoittaa niitä kehitystiimin jäsenille. Pull requests -toiminnallisuus mahdollistaa muutosten yhdistämisen eri haarojen välillä. Muut kehittäjät voivat kommentoida, hyväksyä tai hylätä koodimuutokset ennen niiden yhdistämistä haarasta toiseen. Automaatio puolestaan keskittyy GitHub Actions -työkaluun, joka mahdollistaa jatkuvan integroinnin ja testien ajamisen aina, kun luodaan uusi koodi tiedoston commit.

Kuvassa 11 esitellään OSSIn GitHub arkiston Issues-näkymä. Välilehdelle on nostettu muutamia esille nousseita ongelmia.



Kuva 11. GitHubin Issues-näkymä oppi-api-arkistossa

Projektien päivittäiseen hallintaan GitHub tarjoaa Projects-ominaisuuden, joka muistuttaa kanban-taulua. Se auttaa esimerkiksi tehtävien ja vaiheiden seurannassa ja suunnittelussa. Dokumentointiin on käytössä Wiki, johon voi luoda ja päivittää ohjeita tai muuta dokumentaatiota suoraan selaimella. Tietoturvapuolta hoitaa Security-osio, joka voidaan konfiguroida esimerkiksi skannaamaan projektin riippuvuudet (dependencies) ja varoittamaan, jos niissä havaitaan tunnettuja haavoittuvuuksia. Insights-näkymässä voidaan tarkastella projektin analytiikkaa esimerkiksi commit-taajuuden tai projektin aktiivisuuden osalta. Lopuksi Settings-työkalu mahdollistaa arkiston laajemman hallinnan, kuten käyttöoikeuksien ja näkyvyyden säätämisen, haarojen suojauksen ja ympäristömuuttujien ylläpidon.

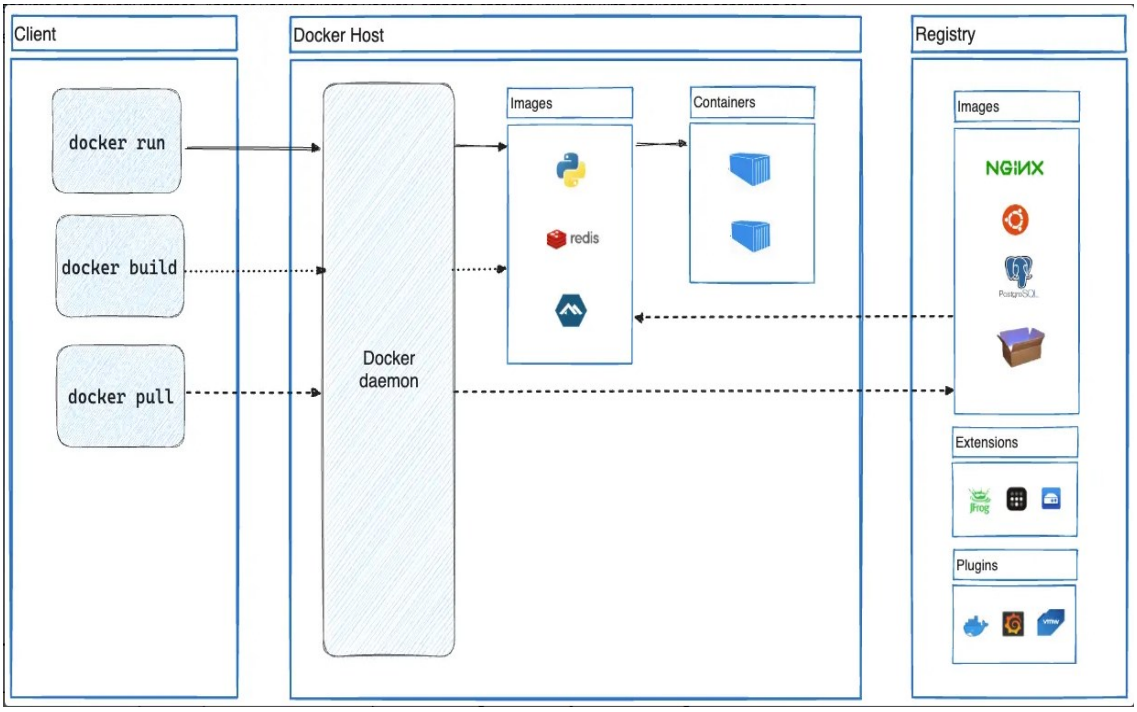
5.3 Docker-konttiteknologia

Kontittaminen on sovelluksen pakkausprosessi. Kontit ovat ohjelmistopaketteja, jotka sisältävät kaiken, mitä sovellus tarvitsee toimiakseen missä tahansa ympäristössä. Toisin kuin perinteiset virtuaalikoneet, jotka ovat resursseja vaativia, kontit jakavat isäntäjärjestelmän kernelin toimiessaan eristetyissä ympäristöissä, mikä tekee niistä sekä kevyitä että tehokkaita. Uuden kontin luomiseen tarvitaan kontti-image, joka on staattinen, suoritettava tiedosto ja sisältää kaikki sovelluksen riippuvuudet. Nämä siirrettävät kontti-imaget voivat käynnistää uusia kontteja sekä Linux- että Windows-alustoilla. (Sukhdeep 2025.)

Kuvassa 12 on esimerkki tyypillisestä Docker-työnkulusta:

- Käyttäjä antaa komennon (docker pull) hakeakseen valmiin imagen Registry-palvelimelta. Docker-daemon noutaa tämän imagen rekisteristä Docker Hostiin.

- docker build -komennolla käyttäjä rakentaa uuden imagen Dockerfilesta.
- Lopuksi docker run -komennolla Docker-daemon käynnistää kontin, joka perustuu olemassa olevaan imageen.



Kuva 12. esimerkki tyypillisestä Docker-työnkulusta (Docker Docs s.a.)

Docker tarjoaa joitain palveluita, jotka ovat hyödyllisiä kehittäjille ja järjestelmänvalvojille. Se on avoin alusta, jota voidaan käyttää sovellusten rakentamiseen, jakeluun ja suorittamiseen kannettavassa, kevyessä ajonaikaisessa ja pakkaustyökalussa, joka tunnetaan nimellä Docker Engine. Se tarjoaa myös Docker Hubin, joka on pilvipalvelu sovellusten jakamiseen. Kustannuksia voidaan vähentää korvaamalla perinteinen virtuaalikone Docker-kontilla. Se vähentää runsaasti pilvikehitysalustan uudelleenrakentamisen kustannuksia. (Bashari Rad 2017.)

Docker käyttää asiakas–palvelin-arkkitehtuuria. Docker-asiakas keskustelee Docker-daemonin kanssa, joka tekee raskaan työn Docker-konttien rakentamisessa, suorittamisessa ja jakelussa. Docker-asiakas ja daemon voivat toimia samassa järjestelmässä tai ne voivat muodostaa etäyhteyden asiakkaan ja Docker-daemonin välille. Docker-asiakas ja daemon kommunikoivat REST API:n, UNIX-sokettien tai verkkorajapinnan kautta. Toinen Docker-asiakas on

Docker Compose, jonka avulla voidaan työskennellä konteista koostuvien sovellusten kanssa. (Docker Docs s.a.)

Dockerin hyödyt verrattuna perinteiseen virtuaalikoneeseen

Perinteisessä hypervisor-pohjaisessa virtualisoinnissa jokainen virtuaalikone sisältää kokonaisen käyttöjärjestelmän, mikä vie runsaasti muistia, levytilaa ja lisää käynnistysaikaa. Dockerin kontit sen sijaan jakavat käyttöjärjestelmän ytimen, joten ne vievät vähemmän resursseja. Konttipohjainen ratkaisu voikin olla huomattavasti kevyempi kuin virtuaalikoneisiin perustuva ratkaisu. Jha (2016) toteaa, että konttien avulla samalla isännällä voidaan ajaa satoja eristettyjä instansseja, ja konttien uudelleenkäynnistys on nopeaa, koska vain sovellusprosessi käynnistyy uudelleen, ei koko käyttöjärjestelmä.

5.4 Azure

Azure on Microsoftin julkinen pilvipalvelualusta. Se tarjoaa laajan valikoiman pilvipalveluita, kuten laskentatehoa, analytiikkaa, tallennustilaa ja verkkoratkaisuja. Käyttäjät voivat valita näistä palveluista kehittääkseen ja skaalatakseen uusia sovelluksia tai käyttääkseen olemassa olevia sovelluksia julkisessa pilvessä. (Yasar & Bigelow 2025.) Yksinkertaisesti selitettynä pilvipalvelut tarjoavat käyttäjille mahdollisuuden ulkoistaa fyysisen laitteiston ja niiden hallinnan palveluntarjoajalle.

Azure soveltuu nykypäivän sovelluskehittämiseen Yasarin ja Bigelowin (2025) mukaan mainiosti esimerkiksi seuraavissa tapauksissa:

- **Virtuaalikoneet ja kontit** – Verkkopalvelut ja niiden ylläpitäminen voidaan toteuttaa ilman fyysisiä omia laitteita.
- **Tietokantojen isännöinti** – Relaatio- ja ei-relaatiotietokantojen isännöinti ilman omaa palvelininfraa.
- **Sovellusten kehitys ja julkaisu** – Sovelluksia voidaan kehittää, testata ja skaalata ilman huolta palvelinten hallinnasta.

- **Varmuuskopiointi ja katastrofista palautuminen** – Azurea käytetään pitkäaikaiseen datan säilytykseen. Pahimman sattuessa järjestelmän palauttaminen aikaisempaan ajankohtaan on mahdollista.

- **Koneoppiminen ja tekoäly** – Azure tarjoaa työkalut tekoälyn kouluttamiseen, rakentamiseen sekä käyttöönottoon.

Tästä listauksesta voidaan todeta Azuren palveluiden soveltuvan kattavasti erityyppisille sovelluksille.

Azure AD

Microsoft kehitti Azure AD:n alun perin Office 365:n kirjautumisia varten. Ilman Azure AD:ta käyttäjät joutuisivat kirjautumaan erillisillä käyttäjätunnuksilla sähköpostiin, SharePointiin, Teamsiin ja muihin Microsoftin palveluihin. Azure AD:ta voisikin kuvailla globaaliksi kirjautumisalustaksi – mutta sen lisäpalvelut mahdollistavat paljon muutakin. Azure AD kerää tietoa muun muassa siitä, millä työvälineellä, mihin kellonaikaan ja mistä päin maailmaa käyttäjät kirjautuvat palveluihin. (Tietokeskus s.a.)

Tietokeskus (s.a) kertoo esimerkiksi tietoturvan näkökulmasta, että Azure AD:n kirjautumistietoja voidaan hyödyntää estämään epäilyttävistä sijainneista tulevat kirjautumiset ja tunnusten väärinkäyttö. Datamurskauksen ja koneälyn avulla voidaan havaita poikkeavaa käyttäytymistä, kuten epärealistinen sijainnin vaihtuminen. Tällöin järjestelmä voi estää pääsyn tai vaatia lisävahvistuksia.

6 TOTEUTUSVAIHE

Tämän kappaleen painopiste on sovelluksen asiakaspuolen toteutuksissa. Sovellusta lähdettiin toteuttamaan suunnitteluvaiheessa luodun arkkitehtuuridokumentin pohjalta. Sovelluskehittäjät jaettiin asiakaspuolen ja palvelinpuolen tiimeihin, joiden ensimmäiseksi tehtäväksi tuli toteuttaa käyttäjien autentikoiminen Microsoft Office 365 -tunnuksilla. Kehittämisen helpottamiseksi opettaja- ja oppilaskäyttäjät päätettiin ohjata sovelluksessa oikeisiin näkymiin sähköpostiosoitteen perusteella. Alkuperäisessä suunnitelmassa moni asia oli ideatasolla ilman tarkempaa kuvausta halutusta toteutuksesta. Autentikointiin

ja sen toiminnallisuuteen panostettiin alkuvaiheessa runsaasti aikaa ja resursseja, koska sen tiedettiin olevan tärkeä kokonaisuus sovelluksen tietoturvan suhteen.

Autentikointi

Autentikoinnin toteutuksessa päädyttiin Azure AD:n (MSAL) ja palvelinpuolen tokenin muokkauksen (mutation) yhdistelmään. Alla kuvattuna autentikointivirtaus asiakaspuolen, palvelinpuolen ja Azure AD:n välillä seuraavasti:

1. Käyttäjä → Asiakaspuoli (aloittaa kirjautumisen)
2. Asiakaspuoli → Azure AD (uudelleenohjaus kirjautumissivulle)
3. Käyttäjä → Azure AD (kirjautuminen O365-tunnuksilla)
4. Azure AD → Asiakaspuoli (uudelleenohjaus koodin kanssa)
5. Asiakaspuoli → Palvelinpuoli (lähettää koodin palvelimelle)
6. Palvelinpuoli → Azure AD (vaihtaa koodin tunnisteisiin)
7. Azure AD → Palvelinpuoli (lähettää alkuperäisen ID-tokenin)
8. Palvelinpuoli (vahvistaa ID-tokenin ja luo oman muokatun tokenin)
9. Palvelinpuoli → Asiakaspuoli (lähettää muokatun tokenin)
10. Asiakaspuoli → Selain (tallentaa muokatun tokenin istuntovarastoon)
11. Asiakaspuoli → Palvelimen API (lähettää muokatun tokenin pyynnön mukana)
12. Palvelimen API (vahvistaa muokatun tokenin jokaisella pyynnöllä)

Tämän toteutuksen vahvuuksina voidaan nähdä Azure AD:n käsittelemä todennus Microsoftin suojattua infrastruktuuria hyödyntäen sekä palvelinpuolen tokenin muokkaus, mikä lisää sovelluksen tietoturvaa. Lisäksi asiakaspuoli pysyy kevyenä palvelinpuolen käsitellessä tietoturvaan liittyviä toimintoja ja logiikkaa. Haittapuolina on hyvä huomioida selaimen instuntovarastoon tallennettujen tietojen mahdollinen altistuminen XSS-hyökkäyksille (Cross Site Scripting). Lisäksi palvelimen kuormitusta ajatellen tulee tiedostaa sovelluksen ja käyttäjämäärien mahdollisesti kasvaessa tokenin ylimääräinen kierros palvelimella.

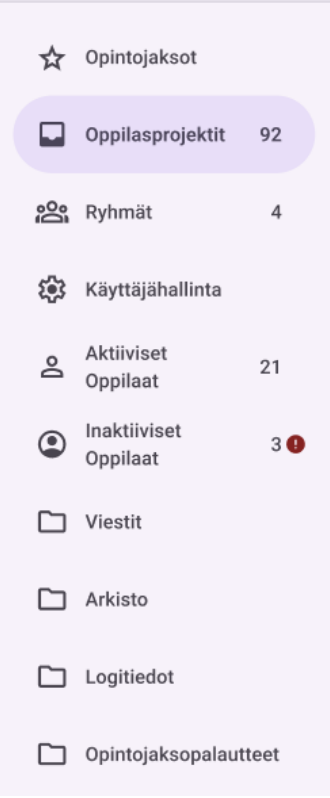
Sovelluksen UI ja UX

Painikkeisiin ja soluihin luotiin efektejä selkeyttämään käyttökokemusta. Hiiren osoittimen ollessa solun tai painikkeen päällä muuttui värisävy tummemmaksi. Lisäksi painikkeet, joista kävi ilmi tilan aktiivisuus, muutettiin tummemmiksi.

Valmiiseen sovellukseen on tarkoitus tuoda Etelä-Savon Koulutus Oy:n organisaation mukainen teemoitus sekä luoda käyttäjälle mahdollisuus tummantilan valintaan.

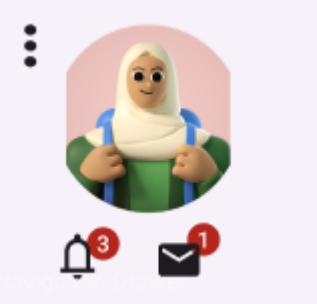
Kaikkien käyttäjien yleisnäkymä

Asiakaspuolen kehityksessä käytettiin mahdollisuuksien mukaan Reactin Material UI:n (MUI) MIT-lisenssin alaisia komponentteja. Mikäli tarvittavilla ominaisuuksilla olevaa komponenttia ei ollut saatavilla, luotiin komponentti itse. Käyttökokemusta ja sivustolla navigointia parantamaan haluttiin sovellukselle lisätä yhtenäisiä komponentteja, jotka sijoitettiin sivuston vasemmalle laidalle. Nämä komponentit ovat läsnä missä tahansa sivuston URL-polussa. Toteutukseen valikoitui kuvan 13 MUI:n drawer ja kuvan 14 box-komponentit.



Kuva 13. React MUI:n drawer-komponentti

Kuvan 13 toteutunut sisältö piti sisällään navigoinnin etusivulle, opiskelijoiden tietoihin, projektitietoihin, teemoihin, työharjoittelupaikkatietoihin ja tutkintotietoihin. Kuvassa 14 nähdään box-komponentin suunniteltu sisältö.

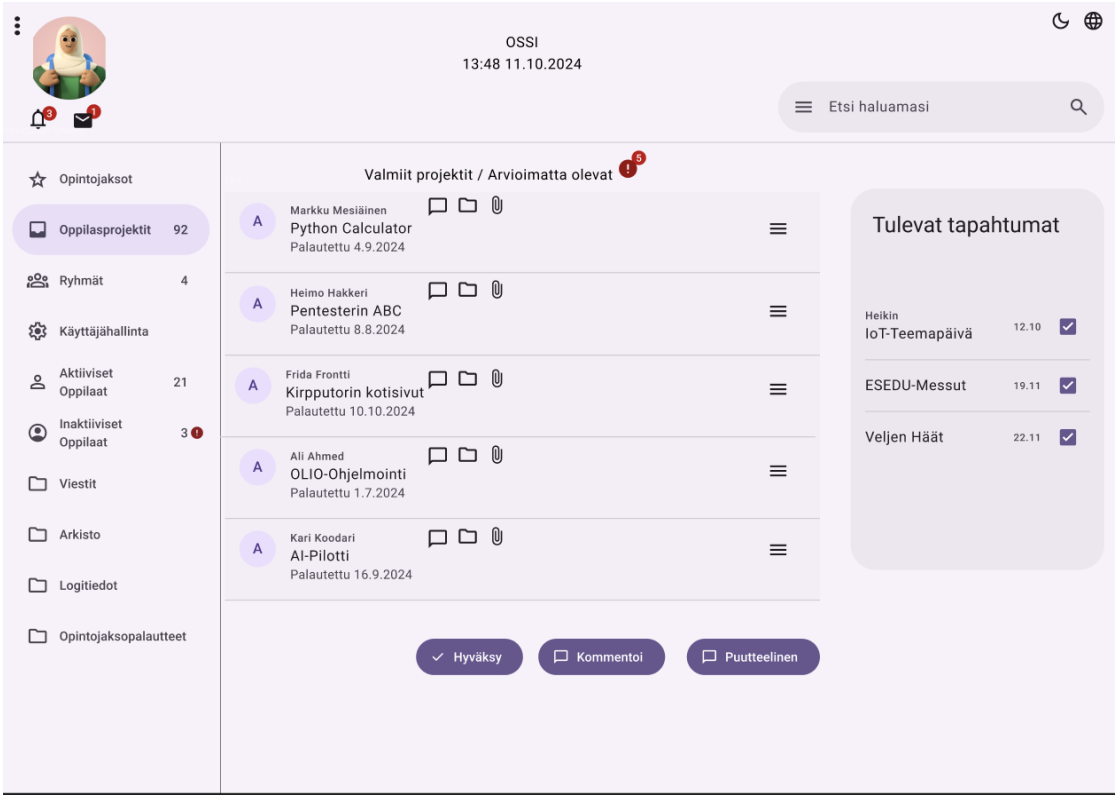


Kuva 14. Box-komponentin sisältö

Kuvan 14 komponenttiin haluttiin toteutuksessa notifikaatiot, asetusvalikko, sekä käyttäjän etu- ja sukunimi. Profiilikuva jätettiin toteutuksesta pois.

Sovelluksen opettajakäyttäjä

Sovelluksen kehittämisessä ehdittiin keskityttyä pääasiassa opettajakäyttäjän toiminnallisuuksiin, käyttöliittymään sekä käyttökokemukseen. Oppilaskäyttäjän kehittämistä ei tämän työn aikana ehditty aloittamaan. Kuvassa 15 on Figma-luonnos opettajakäyttäjän etusivusta.



Kuva 15. Figma-luonnos opettajakäyttäjän etusivusta

Luonnoksen kuvan 15 rakenne voidaan kuvata näin:

- Vasen sivupalkki
  Sisältää päävalikon, jonka avulla käyttäjä voi siirtyä eri osioihin, kuten:
  - Oppilasprojektit
  - Ryhmät
  - Käyttäjähallinta
  - Viestit
  - Arkisto
  - Opintojaksopalaute

  Merkintä "Oppilasprojektit (92)" kertoo aktiivisten projektien määrän. Punainen ilmoitusikoni kertoo arvioimattomista projekteista (esim. "3").

- Yläpalkki
  - Näyttää nykyisen päivämäärän ja kellonajan.
  - Oikealla puolella on hakukenttä "Etsi haluamasi", jolla voidaan hakea opiskelijoita tai projekteja.
  - Käyttäjäasetusten pikanäppäimet kielen ja tummantilan vaihtamiseen.
- Projektien lista (keskellä)
  - Otsikko: "Valmiit projektit / Arvioimatta olevat" Listassa näkyy opiskelijoiden palauttamat projektit:
    - Opiskelijan nimi
    - Projektin nimi
    - Palautuspäivämäärä

Kunkin projektin vieressä on erilaisia toimintopainikkeita, kuten:

- Kommentointi
- Liitetiedostojen tarkastelu
- Arviointityökalut (hyväksy, kommentoi, puutteellinen)

- Oikea reuna – Tapahtumat
  - Tulevat tapahtumat -lista näyttää käyttäjän merkitsemiä tapahtumia, kuten:
    - IoT-Teemapäivä
    - ESEDU-Messut
    - Veljen Häät

Tapahtumien perässä on päivämäärät ja checkboxit, joita voidaan käyttää merkitsemiseen tai muistutukseen. Tästä luonnoksesta oikean reunan tapahtumat-laatikko ei päätynyt lopulliseen toteutukseen.

Tietojenhaku GraphQL-kyselyillä

Kuvan 15 ”Valmiit projektit / Arvioimatta olevat” -laatikon, kuten muidenkin sovelluksessa tarvittujen tietojen noutamiseen tietokannasta käytettiin GraphQL:aa. Kuvassa 16 on sovelluksessa toteutunut projektitietojen hakukysely.

```
import { gql } from '@apollo/client';

export const GET_PROJECT_BY_ID = gql`
    query GetProjectById($id: ID!) {
        project(id: $id) {
            id
            name
            description
            duration
            materials
            isActive
            includedInQualificationUnitParts {
                id
                name
            }
            tags {
                id
                name
            }
        }
    }
`;
```

Kuva 16. GraphQL-projektitietokysely

Teknisesti kuvattuna kuvassa 16 määritellään GraphQL-kysely nimeltä GetProjectById seuraavasti:

- Haetaan kyseisen projektin tiedot palvelimelta Apollo Clientin avulla.
- Kysely ottaa yhden parametrin $id, joka on tyyppiä ID! (pakollinen tunniste).
- Palvelimelta haetaan projektia, jonka tunnus (id) vastaa kyselyyn syötettyä $id-arvoa.
- Kentät id - isActive pyydetään kyselyn vastaukseen.
- includedInQualificationUnitParts ja tags -parametreillä on suhteita projektille kuuluviin osiin. Näistä kyselyssä pyydetään halutut id- ja name -parametrit.

Kuvan 16 kyselyn vastaus palautetaan JSON-formaatissa. Alla kuvassa 17 testikäytössä ollut esimerkkivastaus:

```
{
  "data": {
    "project": {
      "id": "55",
      "name": "Yksinkertaiset verkkosivut HTML-ohjelmoinnilla",
      "description": "Tutustutaan HTML-ohjelmointiin ja luodaan yksinkertaiset verkkosivut.",
      "duration": "4 viikkoa",
      "materials":"-",
      "isActive": true,
      "includedInQualificationUnitParts": [
        { "id": "unit1", "name": "Käyttöliittymät" },
        { "id": "unit2", "name": "Ryhmätyöskentely" }
      ],
      "tags": [
        { "id": "tag1", "name": "React" },
        { "id": "tag2", "name": "UI/UX" }
      ]
    }
  }
}
```


Kuva 17. Kyselyn vastaus palautuu JSON-formaatissa

Kuvassa 17 näkyy GetProjectById -kyselyn palauttama GraphQL-vastaus JSON-muodossa. Tämä tarkoittaa, että tiedot on esitetty asiakaspuolella käsiteltävässä muodossa.

Docker konttiteknologia sovelluskehityksessä

Sovelluksen kehittämistä helpotti suuresti palvelinpuolen palveluiden suorittaminen Docker-konteissa. Kontittamalla palvelut saatiin yhtenäinen, helposti hallittava ja toistettava kehitysympäristö, joka vähensi ympäristöjen välisiä eroavaisuuksia sekä nopeutti sovelluksen asentamista ja käyttöönottoa.

Jokaisen konttiin lisättävän palvelinpuolen palvelun yhteyteen lisättiin kolme keskeistä tiedostoa: Dockerfile, docker-compose.yml ja .dockerignore. Näiden tiedostojen tarkoitus ja toiminnallisuus on kuvattu tarkemmin alla:

Dockerfile

Dockerfile sisältää ohjeet siitä, miten Docker-kuva rakennetaan. Se määrittää pohjakuvan (base image), asentaa tarvittavat riippuvuudet ja määrittelee tarvittavat asetukset, kuten palvelun suoritusympäristön, käytettävät portit sekä käynnistettävät komennot.

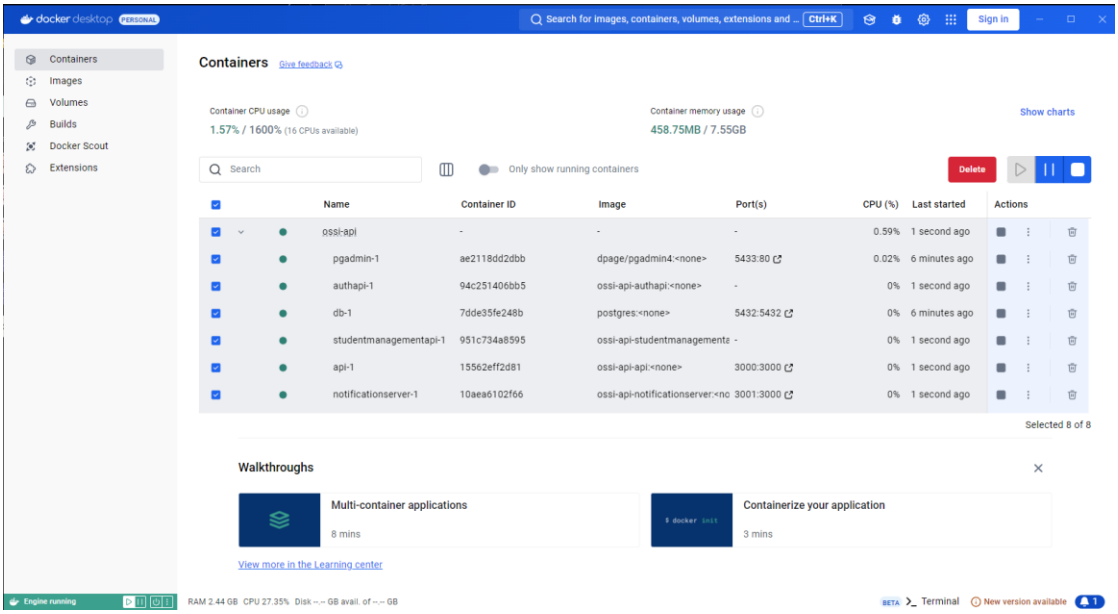
docker-compose.yml

docker-compose.yml mahdollistaa usean Docker-kontin hallinnan ja suorittamisen yhtäaikaisesti. Se yksinkertaistaa ympäristöjen konfigurointia määrittelemällä palveluiden väliset yhteydet, riippuvuudet, porttimääritykset sekä ympäristömuuttujat keskitetysti yhdessä tiedostossa.

.**dockerignore**

.dockerignore-tiedosto määrittelee tiedostot ja kansiot, joita ei sisällytetä Docker-kuvan rakennukseen. Tämän avulla pidetään Docker-kuvan koko pienenä ja vältetään ylimääräisten tai arkaluontoisten tiedostojen päätyminen Docker-kuviin.

Kuvassa 18 on docker desktop -sovellus, jonka avulla kontit voitiin suorittaa. Konttien ollessa ylhäällä, voitiin asiakaspuolen toiminnallisuutta ja tiedonhakua testata sujuvammin.



Kuva 18. Docker desktop

Alkuperäisessä suunnitelmassa myös asiakaspuolta oli tarkoitus suorittaa konteissa, mutta kehitystiimimme koki uusien konttikuvien luomisen jokaisen muutoksen yhteydessä hidastavan kehittämistä, joten tästä ajatuksesta luovuttiin.

7 JATKOKEHITYS

Jatkuvalla integraatiolla (CI) tarkoitetaan käytäntöä, jossa koodimuutokset integroidaan automaattisesti ja usein jaettuun lähdekoodivarastoon. Jatkuva toimitus ja/tai käyttöönotto (CD) on 2-osainen prosessi, joka viittaa koodimuutosten integrointiin, testaukseen ja toimittamiseen. Jatkuva toimitus pysähtyy ilman automaattista tuotannon käyttöönottoa, kun taas jatkuva käyttöönotto vapauttaa päivitykset automaattisesti tuotantoympäristöön. (Redhat 2023.)

Sovelluksen jatkokehityksessä tärkeitä kehityskohteita ovat esimerkiksi:

- CI/CD-putken (Continuous Integration / Continuous Delivery) toteuttaminen ja automatisointi. CI/CD-putken avulla voidaan vähentää sovelluksen ylläpitokuormaa, nopeuttaa julkaisuprosessia sekä vähentää virheiden määrää ja riskiä.
- Viestintämoduulin rakentaminen helpottamaan käyttäjien välistä kommunikointia sovelluksessa.
- Käyttäjähallinnan laajentaminen siten, että se kattaa myös työelämän edustajat. Tämä helpottaisi opiskelijoiden harjoittelujaksojen seurantaa ja raportointia sekä loisi helppokäyttöisen ja nopean viestintäkanavan opiskelijan, opettajan ja työelämän edustajan välille.
- Sovelluksen responsiivisuuden parantaminen, mikä lisää käytettävyyttä erilaisilla laitteilla ja eri resoluutioilla.
- Sovelluksen saavutettavuuden kehittäminen, jotta se ottaisi huomioon opiskelijoiden erilaiset taustat, esimerkiksi kulttuuriin tai terveydentilaan liittyvät erityistarpeet. Käytännön esimerkkejä saavutettavuuden parantamisesta ovat kielivaihtoehtojen lisääminen ja ruudunlukutoiminnon tarjoaminen sovelluksen käyttäjäasetuksissa.

Riippuen päätöksestä laajennetaanko sovelluksen käyttöä kattamaan myös tieto- ja viestintätekniikan ulkopuolisia koulutusaloja, tulee kehityskohteita mahdollisesti suuri määrä listan ulkopuolisia kohteita lisää.

8 PÄÄTÄNTÖ

Tämän työn toteutus oli mielenkiintoista ja haastavaa. Pääsin perehtymään kattavammin sovelluskehityksessä käytettyihin teknologioihin sekä ohjelmistokehityksen prosesseihin yksin- ja tiimityöskentelyn osalta. Sain ottaa osaa lukuisiin suunnittelupalavereihin ja työskennellä omien taitojeni pohjalta siellä, missä minua eniten tarvittiin.

Jälkikäteen ajateltuna prosessi olisi ollut järkevää pilkkoa pienemmäksi osaksi, kuten pelkkään sovelluksen asiakaspuoleen, tai vaihtoehtoisesti toteuttaa opinnäytetyö parityönä. Koin haastavaksi päättää, mitä tietoa olisi syytä sisällyttää ja mitä karsia esimerkiksi käytettyjen teknologioiden tai käyttöliittymän toteutuksen osalta.

Petrattavaakin jäi. Harjoittelujakson jälkeen menetin käyttäjätunnukseni sovellukselle olennaisessa Azure AD -palvelussa. Käytännössä tämä tarkoittaa siis sitä, että pääsyni sovellukseen ei ollut mahdollinen, eikä testaustunnuksia ollut projektille olemassa. Tämä näkyy työn toteutusvaiheessa käytettyjen Figma -luonnoskuvien osalta, joilla on pyritty korvaamaan oikean sovelluksen käyttöliittymän näkymiä. Yhteenvetona tästä prosessinkuvauksesta jäi kuitenkin positiivinen kuva, ja uskon työn parissa opituista taidoista olevan vielä hyötyä tulevaisuudessa. Tahdon omalta osaltani kiittää Etelä-Savon Koulutus Oy:tä mahdollisuudesta työskennellä OSSI-sovelluksen kehitystyön parissa.

LÄHTEET

Airfocus s.a. What is a backend (in a website)? WWW-dokumentti. Saatavissa: https://airfocus.com/glossary/what-is-a-back-end/ [viitattu 4.5.2025]

Amazon AWS s.a. What is PostgreSQL? WWW-dokumentti. Saatavissa: https://aws.amazon.com/rds/postgresql/what-is-postgresql/ [viitattu 13.4.2025]

Barić, M. 2024. Developing React Applications with TypeScript Using Online Development Environments Like Codeanywhere. Blogi. Päivitetty 27.8.2024. Saatavissa: https://codeanywhere.com/blog/developing-react-applications-with-type-script-using-online-development-environments-like-codeanywhere [viitattu 12.5.2025]

Bashari Rad, B. 2017. An Introduction to Docker and Analysis of its Performance. WWW-dokumentti. Saatavissa: https://www.academia.edu/32555833/An_Introduction_to_Docker_and_Analysis_of_its_Performance [viitattu 30.5.2025]

Bass, L., Clements, P. & Kazman R. 2003. Software architecture in practice. E-kirja. Saatavissa: https://finna.fi/Record/aalto.992986294406526?sid=5007772373 [viitattu 27.4.2025]

Carnegie Mellon University s.a. Software Architecture. WWW-dokumentti. Saatavissa: https://insights.sei.cmu.edu/software-architecture/ [viitattu 27.4.2025]

Dhruv, S. 2024. PostgreSQL Advantages and Disadvantages. Blogi. Päivitetty 12.7.2024. Saatavissa: https://www.aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/ [viitattu 13.5.2025]

Docker Docs s.a. What is Docker? WWW-dokumentti. Saatavissa: https://docs.docker.com/get-started/docker-overview/ [viitattu 6.4.2025]

Erickson, J. 2024a. MySQL: Understanding What It Is and How It's Used. WWW-dokumentti. Saatavissa: https://www.oracle.com/mysql/what-is-mysql/ [viitattu 11.05.2025]

Erickson, J. 2024b. What is MongoDB? Expert Guide. WWW-dokumentti. Saatavissa: https://www.oracle.com/fi/database/mongodb/ [viitattu 27.4.2025]

Full stack open s.a. Fundamentals of Web apps. WWW-dokumentti. Saatavissa: https://fullstackopen.com/en/part0/fundamentals_of_web_apps [viitattu 1.5.2025]

GeeksforGeeks. 2025. MongoDB Advantages & Disadvantages. Blogi. Päivitetty 23.1.2025. Saatavissa: https://www.geeksforgeeks.org/mongodb-advantages-disadvantages/ [viitattu 13.5.2025]

GitHub Docs s.a. About Github and Git. WWW-dokumentti. Saatavissa: https://docs.github.com/en/get-started/start-your-journey/about-github-and-git [viitattu 5.4.2025]

Google Cloud s.a. What is a relational database? WWW-dokumentti. Saatavissa: https://cloud.google.com/learn/what-is-a-relational-database [viitattu 5.4.2025]

Ha, D. 2024. 9 reasons Linux is a popular choice for servers. Blogi. Päivitetty 24.7.2024. Saatavissa: https://www.logicmonitor.com/blog/9-reasons-linux-is-a-popular-choice-for-servers [viitattu 4.5.2025]

Haverbeke, M. 2024. Eloquent Javascript. E-kirja. Saatavissa: https://eloquentjavascript.net/00_intro.html [viitattu 27.4.2025]

Herrera, E. 2025. GraphQL vs. REST APIs: What's the difference between them. Blogi. Päivitetty 5.3.2025. Saatavissa: https://blog.logrocket.com/graphql-vs-rest-apis/ [viitattu 12.5.2025]

Hess, A. 2023. Apache HTTP Server Overview: Features, Use Cases, and Alternatives. Blogi. Päivitetty 15.8.2023. Saatavissa: https://www.openlogic.com/blog/apache-http-server [viitattu 4.5.2025]

Hygraph s.a. What is GraphQL? WWW-dokumentti. Saatavissa: https://hygraph.com/learn/graphql [viitattu 12.5.2025]

IONOS. 2023. PHP Framework CodeIgniter. WWW-dokumentti. Saatavissa: https://www.ionos.com/digitalguide/websites/web-development/codeigniter-the-lean-php-framework/ [viitattu 11.5.2025]

Itewiki s.a. Frontend-kehittäjä. WWW-dokumentti. Saatavissa: https://www.itewiki.fi/opas/frontend-kehittaja/ [viitattu 1.5.2025]

Insight Software. 2023. Mitä eroa on? Relaatiotietokannat ja muut kuin relaatiotietokannat. Blogi. Päivitetty 15.5.2023. Saatavissa: https://insightsoftware.com/fi/blog/whats-the-difference-relational-vs-non-relational-databases/ [viitattu 19.4.2025]

Jha, J. 2016. Docker: An Emerging Container Technology. WWW-dokumentti. Saatavissa: https://www.academia.edu/113556398/Docker_An_Emerging_Container_Technology [viitattu 5.4.2025]

Kumar, S. 2020. What is a client, Server, Internal Data Types, and Variables in PHP? WWW-dokumentti. Saatavissa: https://www.devopsschool.com/blog/what-is-a-client-server-internal-data-types-and-variables-in-php/ [viitattu 1.5.2025]

Liimatta, A. 2021. Pilvipalvelut: tiedä tärkeimmät termit. Blogi. Päivitetty 21.5.2021. Saatavissa: https://www.tietoevry.com/fi/blogi/2021/05/pilvipalvelut-tieda-tarkeimmat-termit/ [viitattu 19.4.2025]

Locofy s.a. Frontend Development at Lightning Speed. WWW-dokumentti. Saatavissa: https://www.locofy.ai/ [13.4.2025]

MariaDB s.a. About MariaDB Server. WWW-dokumentti. Saatavissa: https://mariadb.org/about/ [viitattu 5.4.2025]

Microsoft s.a. What is authentication? WWW-dokumentti.Saatavissa: https://www.microsoft.com/en-us/security/business/security-101/what-is-authentication [viitattu 12.5.2025]

Microsoft Learn. 2025. Authentication flow support in the Microsoft Authentication Library (MSAL). WWW-dokumentti. Saatavissa: https://learn.microsoft.com/en-us/entra/identity-platform/msal-authentication-flows [viitattu 12.5.2025]

Microsoft Learn. 2023. Common web application architectures. WWW-dokumentti. Saatavissa: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures [viitattu 30.5.2025]

Netguru. 2025. Ultimate Guide to Web Authentication. Blogi. Päivitetty 22.2.2025. Saatavissa: https://www.netguru.com/blog/web-authentication-guide [viitattu 12.5.2025]

Osowski, R. 2024. What are microservices? WWW-dokumentti. Saatavissa: https://www.ibm.com/think/topics/microservices [viitattu 27.4.2025]

Ozkaya, M. 2023. When to Use and When NOT to Use Microservices: No Silver Bullet. WWW-dokumentti. Saatavissa: https://medium.com/design-microservices-architecture-with-patterns/when-to-use-and-when-not-to-use-microservices-no-silver-bullet-3ae293faf6d [viitattu 27.4.2025]

Pietschmann, C. 2018. Introduction to GitHub and Git Version Control Workflow. WWW-dokumentti. Saatavissa: https://build5nines.com/introduction-to-git-version-control-workflow/ [viitattu 1.5.2025]

Redhat. 2023. What is CI/CD? WWW-dokumentti. Saatavissa: https://www.redhat.com/en/topics/devops/what-is-ci-cd [viitattu 18.5.2025]

Reselman, B. & McKenzie, C. 2023. 10 disadvantages of microservices you'll need to overcome. WWW-dokumentti. Saatavissa: https://www.theserverside.com/answer/What-are-some-of-the-disadvantages-of-microservices [viitattu 27.4.2025]

Saini, S. 2025. Modernizing Software Testing: The impact of Container Technologies. Blogi. Päivitetty 2025. Saatavissa: https://www.computer.org/publications/tech-news/trends/modernizing-software-testing [viitattu 5.4.2025]

Shah, A. 2023. Frontend vs Backend in eCommerce: Key Differences & Importance. Blogi. Päivitetty 24.3.2023. Saatavissa:

https://brainspate.com/blog/frontend-vs-backend-in-ecommerce-unlocking-web-development-secrets/ [viitattu 4.5.2025]

Simplilearn. 2025. Guide To Using Typescript With React. WWW-dokumentti. Saatavissa: https://www.simplilearn.com/tutorials/reactjs-tutorial/react-typescript [viitattu 12.5.2025]

Sukhdeep, S. 2025. Modernizing Software Testing: The Impact of Container Technologies. Blogi. Päivitetty 24.3.2025. Saatavissa: https://www.computer.org/publications/tech-news/trends/modernizing-software-testing [viitattu 30.5.2025]

Tietokeskus s.a. Azure AD on globaali kirjautumisalusta – ja paljon muuta. Blogi. Saatavissa: https://www.tietokeskus.fi/blogi/azure-ad-on-globaali-kirjautumisalusta-ja-paljon-muuta/ [viitattu 13.5.2025]

Tiny s.a. Introduction to TinyMCE. WWW-dokumentti. Saatavissa: https://www.tiny.cloud/docs/tinymce/latest/introduction-to-tinymce/ [13.4.2025]

Toal, R. 2014. A Comprehensive Guide to PHP Programming: What You Need to Know. Blogi. Päivitetty 10.6.2014. Saatavissa: https://codeinstitute.net/global/blog/what-is-php-programming/ [viitattu 12.4.2025]

Visure Solutions s.a. Vaatimukset Määritelmä: Mikä se on ja miten sitä sovelletaan? Blogi. Saatavissa: https://visuresolutions.com/fi/blogi/vaatimusten-m%C3%A4%C3%A4ritelm%C3%A4/ [viitattu 5.4.2025]

Virtanen, J. 2016. UX-design ja UI-design: Mitä eroa niillä on? Blogi. Päivitetty 12.4.2016. Saatavissa: https://www.contrast.fi/blog/ux-design-ja-ui-design-mita-eroa-niilla-on [viitattu 18.5.2025]

W3Schools s.a. PHP Introduction. WWW-dokumentti. Saatavissa: https://www.w3schools.com/php/php_intro.asp [viitattu 1.5.2025]

Yasar, K. & Bigelow, S. J. 2025. What is Microsoft Azure and how does it work? WWW-dokumentti. Päivitetty 2025. Saatavissa: https://www.techtarget.com/searchcloudcomputing/definition/Windows-Azure [viitattu 19.4.2025]