OULUN AMMATTIKORKEAKOULU

VERKKOKAUPPA-ALUSTAN KEHITTÄMINEN

Juhani Koski, Antti Kurkinen ja Arttu Kääriäinen
Opinnäytetyö AMK
Kevät 2025
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä(t): Juhani Koski, Antti Kurkinen ja Arttu Kääriäinen
Opinnäytetyön otsikko: Verkkokauppa-alustan kehittäminen
Työn ohjaaja(t): Jari Kiiskinen
Työn valmistumislukukausi ja -vuosi: kevät 2025
Sivumäärä: 41

Opinnäytetyön tavoitteena oli kehittää monipuolinen verkkoalusta käytettyjen autojen ja niiden varaosien myyntiin. Sivuston avulla käyttäjät voivat helposti luoda uusia myynti-ilmoituksia, huomioiden monipuoliset vaihtoehdot. Autojen lisääminen edellyttää sekä perus- että teknisten tietojen antamista, minkä lisäksi käyttäjien tulee lisätä ilmoitukseen kuvia. Opinnäytetyön idea oli opiskelijoiden itsensä kehittämä.

Opinnäytetyö toteutettiin kehityspainotteisena projektina, jossa käytiin läpi koko kehitysprosessi suunnittelusta toteutukseen ja testaukseen. Aikataulullisten haasteitten takia lopullista versiota ei ehditty viimeistellä testikäyttäjien palautteen perusteella. Työ jakautuu teoria- ja toteutusosuuksiin. Teoriaosuudessa perustellaan valitut teknologiat, kuten ohjelmointikieli, tietokanta ja käyttöliittymän suunnitteluun käytetyt työkalut. Toteutusosuudessa kuvataan, kuinka sovellus yhdistettiin tietokantaan sekä miten eri käyttöliittymänäkymät rakennettiin. Lopuksi käydään läpi sovelluksen testausvaihe ja arvioidaan sen toimivuutta.

ABSTRACT

Oulu University of Applied Sciences
Degree Program in Information technology
Option of software development

Author(s): Juhani Koski, Antti Kurkinen and Arttu Kääriäinen
Title of thesis: Development of online marketplace
Supervisor(s): Jari Kiiskinen
Term and year when the thesis was submitted: Spring 2025
Number of pages: 41

The objective of this thesis was to develop a comprehensive online platform for selling used cars and their spare parts. The platform enables users to create new sales listings effortlessly while considering a wide range of options. Adding a car to the platform requires providing both basic and technical details, along with uploading relevant images. The concept for this project was independently developed by the students.

The thesis was conducted as a development-oriented project, covering the entire process from planning and implementation to testing. Due to time constraints, the final version could not be refined based on user feedback. The work is divided into theoretical and practical sections. The theoretical section justifies the chosen technologies, including the programming language, database, and tools used for designing the user interface. The practical section details how the application was integrated with the database and how the various user interface views were implemented. Finally, the testing phase is discussed, along with an evaluation of the application's functionality.

SISÄLLYS

SANASTO

Blob Storage	Kuvien tallennusratkaisu ulkoisessa palvelussa
JWT	Käyttäjän tunnistamiseen käytetty tuvallinen token-muoto
Next.js	React-pohjainen ohjelmistokehys käyttöliittymän rakentamiseen
Ohjelmistokehys	Valmiita työkaluja ja rakenteita sisältävä alusta ohjelmistojen rakentamiseen
PostgreSQL	Relaatiotietokanta, jossa säilytetään auton ja käyttäjän tietoja
Railway	Kehitysalusta backendin ja tietokannan isännöintiin
Relaatiotietokanta	Tietokantarakenne, joka tallentaa tiedot taulukoihin ja niiden välisiin suhteisiin
REST API	Rajapinta frontendin ja backendin väliseen tiedonsiirtoon
Vercel	Pilvipalvelu, jossa frontend-sovellus on julkaistu

1 JOHDANTO

Opinnäytetyöprojektin tavoitteena on kehittää skaalautuva ja toimiva web-sovellus, toisin sanoen verkkokauppa-alusta, joka mahdollistaa uusien ja käytettyjen autojen sekä niiden varaosien myynnin. Alustan tarkoituksena on tarjota sekä yksityisille myyjille että yrityksille markkinapaikka, joka takaa miellyttävän ja sujuvan ostokokemuksen ostajalle. Projekti sisältää kaikki verkkokaupan olennaiset ominaisuudet, kuten selaamisen, hakutoiminnot, myynti-ilmoitusten hallinnan, maksamisen sekä käyttäjätilit. Maksujärjestelmä toteutetaan Stripe-maksualustan avulla, ja ylläpitoympäristönä käytetään Microsoft Azurea.

Projektin tavoitteena on kehittää responsiivinen, käyttäjäystävällinen ja esteettömyysstandardit täyttävä verkkokauppa, joka ”mahdollistaa tuotteiden lisäämisen, hallinnan ja myynnin sekä yksityisille asiakkaille että yrityksille”. Verkkokaupan tulee sisältää nopea ja tarkka hakutoiminto, joka tukee kategorioiden ja suodattimien käyttöä. Maksujärjestelmä toteutetaan turvallisesti ja helppokäyttöisesti Stripeä hyödyntäen. Alustan skaalautuvuus ja luotettavuus varmistetaan pilvipalveluiden avulla. Tulevaisuudessa verkkokauppaan voidaan lisätä lisäominaisuuksia, kuten personointityökaluja ja analytiikkaa.

Backend rakentuu Node.js- ja Express.js-teknologioiden varaan RESTful API:n toteuttamiseksi, tarjoten hyvän skaalautuvuuden ja suorituskyvyn. Express toimii palvelun ytimenä ja tarjoaa API-endpointit. Tietokantana käytetään Microsoft Azure SQL Databasea relaatiotietorakenteiden hallintaan, mikä mahdollistaa monimutkaisten kyselyjen tehokkaan suorittamisen. Tietoturvaa vahvistetaan käyttämällä JSON Web Tokens (JWT) -autentikointia sekä Helmet- ja rate-limiting-työkaluja API-kutsujen suojaamiseksi

Frontend kehitetään Reactilla dynaamisen ja responsiivisen käyttöliittymän aikaansaamiseksi. Hyödynnetään kirjastoja, kuten Material-UI tai Tailwind CSS, nopeuttamaan suunnitteluprosessia. React Routeria käytetään navigoinnin hallitsemiseksi, ja Axiosia API-kutsujen toteuttamiseen ja hallintaan.

Stripe mahdollistaa turvalliset korttimaksut ja tukee useita maksutapoja. Microsoft Azure tarjoaa sovelluksen ylläpidon ja skaalautuvuuden. Azure Kubernetes Serviceä käytetään sovelluksen isännöintiin, ja Azure SQL database huolehtii tietokannan hallinnasta. Azure Monitor auttaa suorituskyvyn ja vianetsinnän seuraamisessa.

Dockerin avulla sovellus kontitetaan, mikä helpottaa asennusta ja testausympäristöjen hallintaa. Docker Compose yhdistää palvelut (backend, frontend, tietokanta) yhdeksi kokonaisuudeksi kehitysympäristöä varten. Laadunvarmistuksessa käytetään Jest-ohjelmaa automaattiseen yksikkötestaukseen sekä Cypressiä end-to-end-testaamiseen käyttäjäkokemuksen laadun varmistamiseksi.

Tuotehallinnan avulla myyjät voivat lisätä, muokata ja poistaa tuotteita. Jokainen tuote sisältää kuvauksen, kuvat, hinnan ja varastotiedot. Hallintapaneeli myyjille mahdollistaa myynnin seurannan. Hakutoiminnot tukevat tuotteiden hakemista avainsanojen perusteella ja suodatusta hinnan, kategorian tai sijainnin mukaan. Elasticsearch voidaan ottaa käyttöön nopeiden ja joustavien hakutoimintojen toteuttamiseksi.

Käyttäjätilit mahdollistavat rekisteröitymisen ja profiilien hallinnan. Ostohistoria ja suosikit tallennetaan käyttäjäprofiiliin. Maksutapahtumat käsitellään turvallisesti Stripe-alustan kautta, ja käyttäjä voi seurata tilaustensa etenemistä tilausseurannan avulla.

Varmistetaan automaattisten testien kattavuus sekä backend- että frontend-komponenteissa. Integraatiotestit, erityisesti maksutoimintojen osalta ovat kriittisiä. Käyttäjätiedot on suojattava asianmukaisesti GDPR-säädöksiä noudattaen. Tietoturvan vahvistamiseksi hyödynnetään työkaluja, kuten OWASP Dependency-Check.

Projektin eteneminen ja aikataulu alkaa suunnitteluvaiheella, jossa määritellään tavoitteet, teknologiat ja arkkitehtuuri. Kehitysvaiheessa rakennetaan backend- ja frontend-komponentit sekä toteutetaan integraatiot. Testausvaiheessa suoritetaan automaattiset ja manuaaliset testit, ja lopuksi sovellus otetaan käyttöön Azure-ympäristöön.

2 TAUSTATEORIA

2.1 Web-sovellus

Web-sovellukset ovat ohjelmistoja, jotka toimivat internetin välityksellä käyttäen verkkoselainta käyttöliittymänä. Toisin kuin perinteiset työpöytäsovellukset, web-sovellukset eivät vaadi erillistä asennusta käyttäjän laitteelle. Ne perustuvat asiakas-palvelin-arkkitehtuuriin, jossa käyttäjän selain toimii asiakasohjelmana ja palvelin vastaa sovelluksen logiikasta ja tietojen hallinnasta. Tällainen arkkitehtuuri mahdollistaa sen, että käyttäjät voivat käyttää sovelluksia millä tahansa internetyhteydellä varustetulla laitteella. (Volle 11.3.2025.)

Web-sovellusten keskeinen ominaisuus on niiden saavutettavuus ja joustavuus. Koska ne toimivat selaimen kautta, ne ovat riippumattomia käyttöjärjestelmästä ja laitetyypistä, mikä tekee niistä helposti saatavilla olevia eri tilanteissa. Web-sovelluksia käytetään monilla elämänalueilla, kuten verkkokaupassa, pankkitoiminnassa, viihteessä ja yritysten sisäisissä järjestelmissä. Erityisen hyödyllisiä ne ovat yrityksille, sillä ne tarjoavat alustan palveluiden ja tuotteiden skaalautuville ja kustannustehokkaille ratkaisuille. Lisäksi web-sovellusten ylläpito on vaivatonta, sillä päivitykset tehdään keskitetysti palvelimella, eikä käyttäjien tarvitse erikseen asentaa niitä. (Yasar 1.11.2024.)

2.2 Frontend

Frontend tarkoittaa verkkosovelluksen tai -sivuston näkyvää ja käyttäjän kanssa vuorovaikutuksessa olevaa osaa. Se koostuu HTML:stä, CSS:stä ja JavaScriptistä, jotka määrittävät sivuston rakenteen, tyylin ja toiminnallisuuden. Frontendiä käytetään varmistamaan, että käyttäjät voivat selata, käyttää ja olla vuorovaikutuksessa sovelluksen kanssa sujuvasti ja visuaalisesti miellyttävällä tavalla. Se on keskeinen osa verkkokehitystä, ja sitä hyödynnetään erityisesti käyttöliittymien suunnittelussa, responsiivisten sivustojen toteutuksessa ja interaktiivisten elementtien luomisessa.

Hyvä frontend suunnittelu noudattaa tiettyjä periaatteita. Esimerkiksi typografian huolellinen valinta parantaa luettavuutta ja käyttäjäkokemusta. Lisäksi suunnittelussa tulisi välttää ”tunnelinäköä” eli liiallista keskittymistä yhteen yksityiskohtaan kokonaisuuden kustannuksella. On myös tärkeää ymmärtää elementtien hierarkia ja suhteet, jotta käyttäjät voivat helposti hahmottaa sivun sisällön ja navigoida siinä. (Grezeszak s.a.)

Design patternit eli suunnittelumallit ovat toistuvia ratkaisuja yleisiin ongelmiin frontend kehityksessä. Ne auttavat organisoimaan koodia ja parantavat sen uudellenkäytettävyyttä ja ylläpidettävyyttä. Esimerkiksi ”strategy” -malli mahdollistaa erilaisten algoritmien kapseloinnnin yhteen rajapintaan, joilloin sovelluksen käyttäytymistä voidaan muuttaa dynaamisesti ajon aikana. (Ramotion 28.4.2023.)

Frontend-arkkitehtuuri käsittää käytäntöjä ja työkaluja, jotka parantavat sovelluksen koodin laatua. Hyvin suunniteltu arkkitehtuuri mahdollistaa modulaarisen ja skaalautuvan koodin, mikä helpottaa kehitystä ja ylläpitoa. Tämä sisältää esimerkiksi komponenttipohjaisen lähestymistavan, jossa sovelluksen osat jaetaan itsenäisiin, uudellenkäytettäviin komponentteihin. (Dhaduk 31.7.2023.)

2.3 Backend

Backend eli sovelluksen taustajärjestelmä on ohjelmistosovelluksen osa, joka toimii käyttäjän näkymän ulkopuolella ja vastaa sovelluksen toiminnallisuuden toteuttamisesta. Se huolehtii käsittelylogiikasta, liiketoimintasäännöistä, tietoturvasta sekä eri järjestelmien välisestä tiedonvaihdosta. Taustajärjestelmä varmistaa, että sovellus toimii suunnitellusti ja että sen eri osat kommunikoivat tehokkaasti keskenään. Liiketoimintasäännöt määrittelevät, miten sovelluksen on tarkoitus toimia eri tilanteissa. Taustajärjestelmä toteuttaa nämä säännöt varmistaen, että sovellus toimii oikein ja turvallisesti. Taustajärjestelmä on keskeinen osa sovelluksen tietoturvaa. Sen tehtäviin kuuluu käyttäjän tunnistaminen, oikeuksien hallinta, tiedon salaus sekä suojaus erilaisia kyberuhkia vastaan. (Ken 2024.)

Monet sovellukset perustuvat ulkoisiin palveluihin, kuten maksujärjestelmiin tai analytiikka-alustoihin. Taustajärjestelmä toimii välikätenä näiden palveluiden ja sovelluksen välillä. Taustajärjestelmä koostuu useista yhteen liitetyistä komponenteista, jotka mahdollistavat järjestelmän tehokkaan toiminnan. Taustajärjestelmä toimii palvelimella, joka vastaanottaa pyyntöjä, prosessoi ne ja palauttaa vastaukset. Palvelin voi olla fyysinen laite, pilvipalvelu tai virtuaalikone. Rajapinnat mahdollistavat tiedonvaihdon eri sovellusten välillä. REST- ja GraphQL-rajapinnat ovat yleisesti käytettyjä ratkaisuja. (Codeacademy Team 7.11.2022.)

2.4 Pilvipalvelut

Pilvipalvelut ovat IT-infrastruktuurin, alustojen ja ohjelmistojen kokonaisuus, joita kolmannen osapuolen palveluntarjoajat ylläpitävät ja tarjoavat käyttäjille internetin välityksellä. Pilvipalvelut mahdollistavat sovellusten joustavan käytön ilman, että käyttäjän tarvitsee hallita tai ylläpitää omaa fyysistä infrastruktuuria. Pilvipalveluiden avulla organisaatiot voivat skaalata resurssejaan omien tarpeiden mukaan, parantaa kustannustehokkuutta ja hyödyntää tarvittavia teknologioita ilman suuria alkuinvestointeja. (Liimatta 21.5.2021.)

Pilvipalvelut luokitellaan usein kolmeen pääpalvelumalliin: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) ja Software as a Service (SaaS). Näiden lisäksi on olemassa Function as a Service (FaaS), joka tarjoaa kehittäjille tapahtumapohjaisen suoritusalustan ilman tarvetta hallita taustalla olevaa infrastruktuuria (Red Hat 14.3.2022.)

- IaaS (Infrastructure as a Service) tarjoaa käyttäjille laskentatehoa, verkkoyhteyksiä ja tallennuskapasiteettia, jotka voidaan ottaa käyttöön ja skaalata tarpeen mukaan.
- PaaS (Platform as a Service) tarjoaa sovellusten kehittämiseen ja ajamiseen tarvittavan ympäristön, jossa on valmiiksi määritellyt ohjelmointirajapinnat, konttien hallinnat ja muut kehittäjille hyödylliset palvelut.
- SaaS (Software as a Service) tarjoaa valmiita ohjelmistoja pilvipalveluja, jolloin käyttäjät voivat käyttää sovelluksia suoriaan ilman erillistä asennusta tai infrastruktuurin hallintaa.

- FaaS (Funktion as a Service) mahdollistaa koodin suorittamisen pieninä yksittäisinä funktioina ilman palvelinympäristön hallintaa, mikä soveltuu hyvin skaalautuviin ja tapahtumapohjaisin sovelluksiin.

Pilvipalveluiden toimintaperiaatteisiin kuuluu myös resurssien virtualisointi, joka mahdollistaa tietoteknisten resurssien, kuten prosessointitehon, muistin ja tallennustilan erottamisen fyysisestä laitteistosta. Virtualisoinnin avulla resurssit voidaan jakaa tehokkaasti eri käyttäjien ja sovellusten kesken. (Red Hat 14.3.2022.)

Pilvipalvelut voiva olla julkisia, yksityisiä tai hybridipilviä. Julkiset pilvipalvelut, kuten Microsoft Azure, Google Cloud ja Amazon Web Services (AWS s.a. a.) tarjoavat palveluja käyttäjille ja mahdollistavat laajan skaalautuvuuden. Yksityispilvipalvelut ovat organisaation omassa hallinnassa ja tarjoavat enemmän tietoturvaa ja mukautusmahdollisuuksia. Hybridipilvet yhdistävät näiden kahden mallin ominaisuuksia mahdollistaen esimerkiksi arkaluonteisten tietojen säilyttämisen yksityispilvessä ja muiden sovellusten ajamisen julkipilvessä (Liimatta 21.5.2021.)

Pilvipalveluiden avulla organisaatiot voivat lisätä ketteryyttä ja innovaatio kykyä sekä optimoida IT-infrastruktuurin hallinnan. Keskeisiä etuja ovat skaalautuvuus, resurssien optimointi ja kustannustehokkuus. Lisäksi automatisointi ja keskitetty hallinta mahdollistavat järjestelmien tehokkaamman ylläpidon sekä vahvan tietoturvan. (Liimatta 21.5.2021.)

2.4.1 Sovellusten isännöinti

Sovellusten isännöinti tarkoittaa ohjelmistojen ajamista ja hallintaa ulkoisessa infrastruktuurissa, kuten pilvipalveluissa, sen sijaan että ne toimisivat paikallisilla palvelimilla. Se on välttämätöntä useimmille verkkosovelluksille, jotka edellyttävät palvelinpuolen toimintaa, kuten tietokannan hallintaa, käyttäjätunnistusta ja reaaliaikaista tiedonsiirtoa, koska ilman isännöintiä sovellukset rajoittuisivat paikalliseen käyttöön ilman internet-yhteyttä. (Sharma 27.11.2024.)

Pilvipalvelut tarjoavat useita isännöintiratkaisuja, jotka eroavat toisistaan resurssien hallinnan ja suorituskyvyn osalta. Jaetussa palvelimessa (Shared Hosting)

useat sovellukset jakavat saman palvelimen resurssit, mikä on kustannustehokas vaihtoehto pienille sovelluksille. Virtuaalipalvelimessa (VPS Hosting) on jaettu fyysinen palvelin osiin, mikä tarjoaa enemmän suorituskykyä ja hallintamahdollisuuksia. Pilvipohjainen palvelin (Cloud Hosting) on skaalautuva ja joustava ratkaisu, jossa resurssit jaetaan useiden palvelinten välillä käyttöasteen mukaan. Dedikoidussa palvelimessa (Dedicated Hosting) käyttäjä saa koko palvelimen käyttöönsä, mikä soveltuu suurille sovelluksille. (AWS s.a. a.)

Sovellusten isännöinnin valinnassa keskeisiä tekijöitä ovat suorituskyky, tietoturva, käytettävyys ja asiakastuki. Pilvipohjainen isännöinti tarjoaa mahdollisuuden keskittyä sovellusten kehittämiseen ja käyttöön ilman infrastruktuurin hallinnan haasteita. (Sharma 27.11.2024.)

2.4.2 Tietokanta

Tietokanta on sähköisesti tallennettu, järjestelmällinen kokoelma dataa, joka voi sisältää esimerkiksi tekstiä, numeroita, kuvia, videoita ja tiedostoja. Tietokannat mahdollistavat suurten tietomäärien hallinnan ja analysoinnin, ja niiden käyttö on keskeistä monilla eri aloilla. Tietokantojen hallintaan käytetään hallintajärjestelmiä (DBMS), joiden avulla voidaan tallentaa, hakea ja muokata tietoa tehokkaasti. (AWS s.a. b.)

Tietokannat ovat elintärkeitä yrityksille, sillä ne tukevat liiketoiminnan sisäisiä prosesseja, asiakas- ja toimittajatietojen hallintaa sekä datan analysointia. Ne mahdollistavat tietojen eheyden ja turvallisuuden, sillä niiden käyttöoikeuksia voidaan hallita ja asettaa rajoituksia käyttäjille tietojen muokkaamiselle. Lisäksi tietokantojen tietoja voidaan analysoida ja raportoida, mikä auttaa yrityksiä ja organisaatioita tekemään tietoon perustuvia päätöksiä. (AWS s.a. b.)

Tietokantoja voidaan luokitella useiden tekijöiden perusteella, kuten tietojen säilytystavan, käyttötarkoituksen ja rakenteen mukaan. Esimerkiksi relaatiotietokannat tallentavat tiedot taulukkomuotoon, jossa sarakkeet määrittelevät tietojen rakenteen ja rivit sisältävät yksittäiset tietueet. Relaatiotietokantojen käyttö perustuu kyselykieleen (SQL) ja ne soveltuvat hyvin rakenteellisen tiedon hallintaan. (AWS s.a. b.) NoSQL-tietokannat mahdollistavat joustavan tietomallin ilman

perinteisiä taulukko- ja sarakerakenteita. Ne ovat erityisen hyödyllisiä suurten, hajautettujen datamäärien käsittelyssä, kuten big data -sovelluksissa. (AWS s.a. b.; Gillis 2024.) Pilvipohjaiset tietokannat toimivat pilvipalveluissa ja tarjoavat skaalautuvuuden, korkean käytettävyyden sekä mahdollisuuden käyttää tietokantoja palveluna (Database as a Service, DBaaS). (Gillis 2024.) In-memory-tietokannat tallentavat tiedot suoraan keskusmuistiin, mikä mahdollistaa erittäin nopeat tietokantakyselyt. Tätä ratkaisua käytetään erityisesti telekommunikaatiosovelluksissa, joissa vasteaika on kriittinen. (AWS s.a. b.)

Tietokantojen kehitys on johtanut modernien tietokantaratkaisujen syntymiseen, joissa hyödynnetään tekoälyä, koneoppimista ja automaatiota. Pilvipohjaiset ja itseohjautuvat tietokannat vähentävät hallinnollista työtä ja mahdollistavat tietojen skaalautuvan käsittelyn reaaliajassa. Tietoturvan merkitys on myös kasvanut, ja pilvipalveluiden toimittajat tarjoavat kehittyneitä tietoturvaratkaisuja, kuten automaattisia varmuuskopioita, salattua tiedonsiirtoa ja roolipohjaista pääsynhallintaa. (Gillis 2024.)

Pilvipalveluiden kehittyessä tietokantojen rooli sovellusten taustajärjestelmissä on entistä tärkeämpi. Ne mahdollistavat tehokkaan tietojen hallinnan, analysoinnin ja skaalautuvuuden, jotka ovat välttämättömiä nykyaikaisille web-sovelluksille ja yritysratkaisuille. (AWS s.a. b.)

2.4.3 Konttiteknologia ja orkestrointi

Konttiteknologia mahdollistaa sovellusten ja niiden riippuvuuksien pakkaamisen yhdeksi itsenäiseksi yksiköksi eli kontiksi. Kontti sisältää kaiken tarvittavan sovelluksen suorittamiseen, kuten koodin, kirjastot ja asetustiedostot, mikä tekee niistä helposti siirrettäviä ja käyttövalmiita eri ympäristöissä. Kontit ovat kevyempi ratkaisu kuin perinteiset virtuaalikoneet, sillä ne jakavat isäntäkäyttöjärjestelmän ytimen sen sijaan, että jokaisella sovelluksella olisi oma käyttöjärjestelmä. Tämä parantaa suorituskykyä ja vähentää resurssien kulutusta. (Google Cloud s.a.; Red Hat 6.4.2023.)

Kontit mahdollistavat nopeamman ohjelmistokehityksen ja jatkuvat toimituksen (CI/CD), sillä ne tarjoavat vakaan ja yhdenmukaisen toimintaympäristön eri

kehitysvaiheissa. Yksi suosituimmista konttiteknologioista on Docker, joka on avoimen lähdekoodin alusta sovellusten kontittamiseen ja hallintaan. Muita vaihtoehtoja ovat esimerkiksi CRI-O ja Containerd, jotka tarjoavat palveluja konttien ajamiseen. (SolarWinds s.a.)

Konttien hallinta ja skaalaus vaativat tehokasta orkesterointia, sillä suuret järjestelmät voivat koostua sadoista tai tuhansista konteista. Kubernetes on suosituin avoimen lähdekoodin konttien hallintajärjestelmä, joka mahdollistaa niiden automaattisen käyttöönoton, skaalaamisen ja ylläpidon. Kubernetes hallitsee kontteja ryhmittämällä ne solmuiksi (nodes), joita voidaan ajaa sekä virtuaalikoneilla että fyysisillä palvelimilla. (Red Hat 6.4.2023.)

Orkesterointialustoilla, kuten Kubernetesilla esimerkiksi sovellusten suorituskykyä voidaan säätää dynaamisesti kuormituksen mukaan ja jakaa liikenne automaattisesti useiden konttien kesken, mikä parantaa suorituskykyä ja saatavuutta. Jos jokin konteista kaatuu, Kubernetes käynnistää sen automaattisesti uudelleen ja varmistaa palvelun jatkuvuuden. (Red Hat 6.4.2023.)

2.4.4 Jatkuva integraatio ja toimitus

Jatkuva integraatio ja toimitus (CI/CD) on ohjelmistokehitysmenetelmä, joka automatisoi koodin integroinnin, testauksen ja toimituksen mahdollistaen nopeammat ja luotettavammat ohjelmistojulkaisut. Automaatio minimoi manuaalisen työn, joka vähentää virheiden määrää ja nopeuttaa uusien ominaisuuksien julkaisua. (Red Hat 12.12.2023.)

Jatkuva integraatio yhdistää kehittäjien koodimuutokset yhteiseen lähdekoodivarastoon usein ja automatisoidusti. Jokainen muutos käynnistää automaattiset testit, joiden avulla voidaan havaita virheet ja yhteensopivuusongelmat varhaisessa vaiheessa. Tämä vähentää koodin yhteen sulauttamisesta aiheutuvia ongelmia ja nopeutta virheiden korjaamista. CI-prosessi kattaa tyypillisesti lähdekoodin versionhallinnan, automaattisen käännöksen, yksikkötestauksen ja koodianalyysin. (GitLab s.a.)

Jatkuva toimitus automatisoi koodin validoinnin ja jakelun, jolloin ohjelmistopäivitykset voidaan julkaista tuotantoympäristöön nopeasti. CD-prosessissa kehittäjän muutokset testataan kattavasti ennen niiden siirtämistä tuotantoversioon. Tämä varmistaa, että ohjelmisto on jatkuvasti julkaisukelpoinen. (GitLab s.a.)

Jatkuva käyttöönotto on jatkuvan toimituksen laajennus, jossa hyväksytyt koodimuutokset siirretään automaattisesti tuotantoympäristöön ilman manuaalista hyväksyntää. Tämä tarkoittaa, että ohjelmistopäivitykset voivat mennä tuotantoon minuuteissa niiden kehittämisestä, mikä mahdollistaa nopean palautteen saamisen käyttäjiltä ja sovelluksen jatkuvan parantamisen. Jatkuvassa käyttöönotossa testiautomaatiojärjestelmä täytyy olla vahva ja tarkasti toteutettu, koska ohjelmistoversiot julkaistaan ilman erillistä valvontaa. (GeeksforGeeks 2025.)

Pilvipalveluiden tarjoama skaalautuva infrastruktuuri mahdollistaa CI/CD käytön eri ympäristöissä. Esimerkiksi Microsoft Azure, Google Cloud ja Amazon Web Services (AWS) tarjoavat hallittuja CI/CD-palveluita, jotka helpottavat ohjelmistokehitysprosessin automatisointia ja nopeuttavat tuotantovalmiiden ohjelmistojen toimitusta. (GitLab s.a.)

3 TEKNOLOGIA

3.1 Node.js

Node.js on avoimen lähdekoodin palvelinpuolen javascript-ympäristö, joka on suunniteltu erityisesti skaalautuvien ja suorituskykyisten verkkosovellusten kehittämiseen. Noden keskeisimpiä vahvuuksia on sen asynkroninen toimintamalli, joka perustuu tapahtumapohjaiseen arkkitehtuuriin ja ei-blokkaavaan I/O-malliin. Tämä mahdollistaa suuren määrän samanaikaisia operaatioita ilman, että palvelin joutuu odottamaan yksittäisten tehtävien valmistumista. Node.js sisältää npm:n eli Node Package Manager, joka on maailman suurin ekosysteemi avoimen lähdekoodin kirjastoille. Se mahdollistaa lisäosien ja kirjastojen helpon asentamisen. (Node.js s.a.)

3.2 Express.js

Express.js on kevyt ja nopea web-kehys, joka on rakennettu Node.js:lle. Se tarjoaa yksinkertaisen mutta tehokkaan työkalupaketin HTTP-palvelimien ja websovellusten rakentamiseen. Express on minimalistinen, mutta erittäin laajennettavissa ja sen avulla kehittäjät voivat luoda monimutkaisia web-sovelluksia ja RESTful-rajapintoja helposti. Express tarjoaa perusrungon web-sovelluksille, joihin voi lisätä tarvittavat lisäosat ja ominaisuudet kolmannen osapuolen moduuleilla helpottamaan sovelluksen kehitystä. Koska express on rakennettu Node.js:n päälle se hyödyntää Noden asynkronisia ominaisuuksia ja tapahtumapohjaista luonnetta mikä tekee siitä erittäin suorituskykyisen. (MDN Web Docs 20.12.2024.)

3.3 React

React on JavaScript-kirjasto, jota käytetään käyttäjärajapintojen rakentamiseen verkkosovelluksissa. Se mahdollistaa tehokkaan ja joustavan tavan luoda interaktiivisia käyttöliittymiä jakamalla ne pieniin uudelleen käytettäviin

komponentteihin. React hyödyntää virtuaalista asiakirjan objektimalli DOM-rakennetta, mikä nopeuttaa käyttöliittymän päivityksiä ja parantaa suorituskykyä. Sitä käytetään erityisesti sovelluksissa, joissa vaaditaan dynaamista ja reaaliaikaista vuorovaikutusta käyttäjän kanssa, kuten verkkosivustoilla, mobiilisovelluksissa ja monimutkaisissa verkkosovelluksissa. Reactin avulla kehittäjät voivat hallita käyttöliittymän tilaa tehokkaasti ja luoda moderneja, responsiivisia ratkaisuja. (Carrol & Hanlon 14.2.2025)

3.4 Next.js

Next.js on moderni Reactiin perustuva JavaScript-kehys, joka tarjoaa työkaluja verkkosovellusten ja -sivustojen rakentamiseen. Sen tarkoitus on tehdä ohjelmistokehittäjien työstä helpompaa tarjoamalla valmiita ratkaisuja, kuten palvelin puolen renderöinti, staattisen sivun luominen ja API-reititystä. Näiden tekniikoiden avulla sovellukset ja sivut latautuvat nopeammin, tarjoavat paremman käyttökokemuksen ja hakukoneilla on helpompaa ymmärtää sisältöä.

Kehys on erityisen hyödyllinen dynaamisten verkkosivustojen ja sovellusten rakentamisessa, joissa tarvitaan tehokasta suorituskykyä, että joustavuutta sisällön hallinnassa. Kehystä käytetään esimerkiksi verkkokauppoihin, blogeihin, monimutkaisiin verkkosovelluksiin ja muuhun, missä yhdistyy nopeus, skaalautuvuus ja nykyaikainen käyttökokemus. (Next.js s.a.)

3.5 TailwindCSS

Tailwind CSS on käyttöliittymäkirjasto, joka tarjoaa valmiita käyttöön otettavia tyyliluokkia verkkosivujen ja sovellusten ulkoasun rakentamiseen. Sen idea on tehdä tyylittelystä tehokasta ja joustavaa käyttämällä suoraan HTML-elementeissä määriteltäviä luokkia, jotka vastaavat CSS-ominaisuuksia. Tämä lähestymisentapa poistaa tarpeen kirjoittaa perinteisiä CSS-tiedostoja ja mahdollistaa nopean iteroinnin sekä komponenttien visuaalisen hallinnan suoraan koodissa.

Tailwind CSS sopii erityisesti projekteihin, joissa halutaan yhdistää nopea kehitys, tarkka visuaalinen hallinta ja responsiivisuus. Sitä käytetään laajasti

kaikenlaisissa verkkosovelluksissa, alkaen yksinkertaisista sivustoista aina monimutkaisiin käyttöliittymiin, koska se tekee tyylien luomisesta selkeää ja modulaarista. (Tailwind CSS s.a.)

3.6 DaisyUI

DaisyUI on Tailwind CSS:n päälle rakennettu komponenttikirjasto, joka tarjoaa valmiita, tyyliteltyjä käyttöliittymäelementtejä, kuten painikkeita, lomakkeita, kortteja ja muita UI-komponentteja. Sen avulla kehittäjät voivat luoda nopeasti yhtenäisiä ja responsiivisia käyttöliittymiä ilman, että heidän tarvitsee kirjoittaa paljon muokautettua CSS:ää. DaisyUI helpottaa myös teeman hallintaa ja mukauttamista, mikä tekee siitä hyödyllisen erityisesti projekteissa, joissa halutaan säilyttää selkeä visuaalinen ilme ja nopeuttaa kehitysprosessia. (daisyUI s.a.)

3.7 Microsoft Azure

Microsoft Azure on pilvipalvelualusta, joka tarjoaa laajan valikoiman palveluita sovellusten ja palveluiden rakentamiseen, testaamiseen, käyttöönottoon ja hallintaan maailmanlaajuisen tietokeskusinfrastruktuurinsa kautta. Alusta tukee erilaisia palvelumalleja, kuten ohjelmisto palveluna (SaaS), alusta palveluna (PaaS) ja infrastruktuuri palveluna (IaaS), sekä monia ohjelmointikieliä, työkaluja ja kehityksiä. (McCoy s.a.)

3.8 Vercel

Vercel on yhdysvaltalainen teknologiayritys, joka tarjoaa pilvipalvelualustan (Platform as a Service) ja ylläpitää avointa lähdekoodia olevaa Next.js-web-kehystä. Vercelin alusta on suunniteltu helpottamaan kehittäjien työtä tarjoamalla työkaluja ja infrastruktuuria nopeampien ja henkilökohtaisempien verkkosivustojen rakentamiseen, skaalaamiseen ja suojaamiseen. Yritys on saanut rahoitusta useissa rahoituskierroksissa, mukaan lukien 102 miljoonan dollarin Series C -kierros kesäkuussa 2021, ja sen arvo oli toukokuussa 2024 noin 3,25 miljardia dollaria. (Vercel s.a.)

3.9 Railway

Railway on pilvipohjainen alusta, joka mahdollistaa sovellusten nopean käyttöönoton ja hallinnan tarjoamalla kehittäjille yksinkertaisen tavan yhdistää infrastruktuurinsa, kuten tietokantoihin, ilman monimutkaisia asetuksia. Yksi palveluista on PostgreSQL-tietokanta, jonka avulla käyttäjät voivat helposti ottaa käyttöön ja hallita PostgreSQL-instansseja projekteissaan. Railway integroituu saumattomasti eri sovelluskehyksiin ja tarjoaa valmiita malleja, kuten ExpressJS:n ja PostgreSQL:n yhdistelmän, mikä helpottaa kehittäjien työtä. (Railway Docs s.a.)

3.10 GitHub

Git on versiohallintajärjestelmä, joka auttaa kehittäjiä hallitsemaan ja seuraamaan koodimuutoksia tehokkaasti. Se mahdollistaa yhteistyön useiden kehittäjien kesken, säilyttää tallenteet eri kehitysvaiheista ja tarjoaa työkalut muutosten yhdistämiseen ja hallintaan. Gitin avulla voidaan projekti palauttaa aiempiin versioihin, haarauttaa uusia kehityslinjoja ja yhdistää ne myöhemmin ilman tietojen menetystä. Sitä käytetään ohjelmistokehityksessä kaikenkokoisissa projekteissa, erityisesti avoimen lähdekoodin hankkeissa ja tiimityössä, jossa tarvitaan selkeästi hallittua ja turvallista tapaa käsitellä koodia. (Git s.a.)

GitHub on pilvipohjainen alusta, joka tarjoaa Git-versiohallintajärjestelmään perustuvan koodin tallennus- ja yhteistyöympäristön. Sen avulla kehittäjät voivat hallita, jakaa ja tehdä yhteistyötä ohjelmistoprojekteissa tehokkaasti. GitHub mahdollistaa koodivarastojen versionhallinnan, tiimityön, muutosten tarkastelun ja yhdistämisen sekä avoimen lähdekoodin kehittämisen. Sitä käytetään erityisesti ohjelmistokehityksessä, jossa tarvitaan turvallinen ja keskitetty paikka koodin säilyttämiseen, muutosten hallintaa ja projektien julkaisuun. (GitHub Docs s.a.)

3.11 Visual Studio Code

Visual Studio Code (VS Code) on Microsoftin kehittämä avoimen lähdekoodin tekstieditori, joka on suunniteltu erityisesti ohjelmistokehitykseen. Se on kevyt, mutta samalla monipuolinen työkalu, joka tukee useita eri ohjelmointikieliä ja tarjoaa laajan valikoiman ominaisuuksia kehitystyön tehostamiseen. Vs Code sisältää IntelliSense-ominaisuuden, joka tarjoaa älykästä koodin täydennystä ja ehdotuksia. Tämä ominaisuus nopeuttaa ohjelmointia ja vähentää virheiden määrää. Lisäksi editori tukee monien ohjelmointikielten virheenkorjausta, mikä mahdollistaa ohjelmakoodin tehokkaan tarkastelun ja testaamisen. VS code sisältää sisäänrakennetun terminaalin, jonka avulla käyttäjät voivat suorittaa komentorivikäskyjä suoraan editorissa. Lisäksi VS Code tukee Git-versionhallintaa, jonka avulla kehittäjät voivat hallita ohjelmistoprojektin eri versioita ilma ulkoisia työkaluja. (Visual Studio Code s.a.)

4 TYÖN TOTEUTUS

4.1 Käyttäjien autentikointi ja roolipohjainen pääsynhallinta verkkosovelluksessa

Tässä projektissa on toteutettu käyttäjärekisteröinti ja roolipohjainen pääsynhallinta hyödyntäen moderneja teknologioita kuten PostgreSQL -tietokantaa, Node.js:ää sekä JWT-pohjaista autentikointia. Tavoitteena oli luoda turvallinen ja skaalautuva järjestelmä, joka mahdollistaa erilaisten käyttäjien (esim. tavalliset käyttäjät ja admin-käyttäjät) hallinnan ja rajoittaa pääsyä sovelluksen eri osiin roolin mukaan. Tietoturva oli keskeinen osa järjestelmän suunnittelua ja toteutusta, ja se takasi sen, että käyttäjien tiedot olivat suojattuina sekä rekisteröinnin että kirjautumisen aikana.

Rekisteröintiprosessi alkaa käyttäjän syöttäessä henkilökohtaiset tietonsa verkkosovellukseen, jossa tarkistetaan, että kaikki vaaditut kentät (nimi, sähköpostiosoite, puhelinnumero ja salasana) ovat oikein täytetty. Tiedot tallennetaan PostgreSQL -tietokantaan käyttäen POST-pyyntöä ja pg-kirjastoa Node.js-palvelimella. Salasana on tärkeä tietoturvatekijä, ja sen vuoksi se on suojattu bcrypt-hajautusalgoritmilla ennen tallennusta tietokantaan (KUVA 1). Tämä varmistaa, että vaikka tietokanta vuotaisikin, käyttäjien salasanat ovat turvassa.

```
Koodi alkaa.

// Insert user into database
  await pool.query(
    `INSERT INTO users (name, email, phone, passwordhash, role)
    VALUES ($1, $2, $3, $4, $5)`,
    [name, email, phone, hashedPassword, role]
  );

Koodi päättyy.
```

KUVA 1. Salasanan hajautus bcrypt-algoritmilla (kuvakaappaus koodista)

Tietokannan rakenteessa on otettu huomioon myös mahdollisuus tulevaisuudessa laajentaa rekisteröintitietoja tai lisätä uusia käyttäjärooleja. Käyttäjälle asetetaan oletuksena rooli user, mutta järjestelmässä on myös mahdollisuus

määrittää eri rooleja, kuten admin, jotka antavat laajemmat käyttöoikeudet sovelluksen hallintapaneeliin.

Rekisteröinnin yhteydessä suoritetaan tiukat tarkistukset, kuten sähköpostiosoitteen oikeellisuus ja salasanan vaatimusten täyttäminen (esimerkiksi vähintään 8 merkkiä, sisältää isoja kirjaimia, numeroita ja erikoismerkkejä).

SQL-kyselyiden rakenteelliset virheet, kuten sarakkeiden ja arvojen täsmäys, tunnistettiin ja korjattiin. Virheiden käsittelyssä käytettiin try-catch-lohkoja ja palautettiin asianmukaiset virheilmoitukset.

4.2 Roolipohjainen pääsynhallinta

Tässä projektissa hyödynnettiin roolipohjaista pääsynhallintaa (RBAC, Role-Based Access Control), jonka avulla käyttäjien oikeuksia voidaan hallita tarkasti heidän roolinsa mukaan. Käyttäjille annetaan rooli, joka määrittää heidän pääsynsä tietyille resursseille ja toiminnoille sovelluksessa. Tämä toteutettiin JWT-tunnisteiden avulla, jotka sisälsivät käyttäjän roolin ja muut tunnistetiedot.

Admin-roolia varten luotiin erillinen pääsy hallintapaneeliin, joka mahdollistaa käyttäjien ja ajoneuvojen hallinnan. Kun käyttäjä yrittää päästä /admin/dashboard-reitille, palvelin tarkistaa, onko hänellä admin-rooli JWT-tunnisteessa. Jos rooli ei ole admin, pääsy estetään, ja käyttäjä saa virheilmoituksen Access Denied. Admins only (KUVA 2).

```
Koodi alkaa.

//Admin Dashboard Route (Protected)
router.get("/dashboard", authenticateToken, async (req, res) => {
  try {
    // Check if the user is in the admin database
    const result = await pool.query(
      "SELECT role FROM users WHERE userid = $1",
      [req.user.userId]
    );

    if (result.rows.length === 0 || result.rows[0].role !== "admin") {
      return res.status(403).json({ message: "Access denied. Admins only." });
    }

    res.json({ message: "Welcome to the admin dashboard!" });

  } catch (error) {
```

```
    console.error("Admin dashboard error:", error);
    res.status(500).json({ message: "Internal server error." });
  }
});

Koodi päättyy.
```

KUVA 2. Admin-käyttäjän tunnistaminen ja pääsynhallinta (kuvakaappaus koodista)

Käyttäjäroolit: Tavallisille käyttäjille (user-roolilla) on rajattu pääsy vain tavanomaisiin toimintoihin, kuten ajoneuvojen selaamiseen ja omien tietojen hallintaan. Admin-käyttäjät puolestaan voivat hallita myös muiden käyttäjien tietoja ja tehdä muutoksia sovelluksen rakenteeseen.

4.3 Roolien hallinta ja käyttöoikeudet

Järjestelmässä on mahdollista lisätä uusia rooleja ja määrittää niiden käyttöoikeudet helposti. Esimerkiksi moderaattorirooli voisi saada pääsyn tiettyihin sovelluksen osiin, mutta ei hallintapaneeliin. Roolien luominen ja hallinta toteutetaan tietokannan kautta ja järjestelmänvalvoja voi muokata niitä hallintapaneelin kautta, mikä mahdollistaa joustavan ja skaalautuvan käyttöoikeuksien hallinnan.

Pääsynhallinnan ja autentikointijärjestelmän toimivuus testattiin kattavasti postmanin avulla. Rekisteröintiä testattiin POST-pyynnöllä /register-reitille varmistaen, että käyttäjätiedot rekisteröidään oikein ja kaikki kentät validoidaan asianmukaisesti. Kirjautumisen varmistettiin lähettämällä POST-pyyntö /auth/login-reitille, jossa oikeilla tunnuksilla palvelin palautti JWT-tunnisteen. Suojattujen reittien toiminta testattiin lähettämällä GET-pyyntö /admin/dashboard-reitille, jossa JWT-tunniste sisällytettiin Authorization-headeriin. Pääsynhallintaa testattiin sekä admin- että user-käyttäjärooleilla, ja testauksen tulokset osoittivat, että admin-käyttäjä pääsi hallintapaneeliin, kun taas tavallinen käyttäjä estettiin onnistuneesti ja sai virheilmoituksen ”Access Denied”.

Testauksen perusteella järjestelmän autentikointi ja pääsynhallinta toimi odotetusti, ja vain oikeilla käyttöoikeuksilla varustetut käyttäjät voivat käyttää hallintatoimintoja. JWT-tunnisteiden ja roolipohjaisen pääsynhallinnan ansioista järjestelmä on tehokas turvallinen ja helposti skaalautuva. Tämä ratkaisu mahdollistaa

selkeän käyttöoikeuksien hallinnan varmistaen, että vain valtuutetut käyttäjät voivat hallita myynti-ilmoituksia ja asiakastietoja.

4.4 Tietoturva ja käyttäjätunnistus verkkosovelluksessa

Tietoturva on keskeinen osa verkkosovellusten kehityksessä, ja se tulee ottaa huomioon koko sovelluksen elinkaaren ajan. Erityisesti käyttäjätietojen suojaaminen ja autentikointi ovat osa-alueita, joihin kiinnitetään erityistä huomiota. Tässä osiossa tarkastellaan tarkemmin käyttäjän autentikointia ja valtuutusta JWT (JSON Web Token) -pohjaisella järjestelmällä, joka mahdollistaa turvallisen ja skaalautuvan käyttäjähallinnan.

JWT on standardi, joka mahdollistaa käyttäjän tunnistamisen ja pääsyn valtuuttamisen turvallisesti ilman, että palvelimen tarvitsee säilyttää käyttäjien sessiotietoja. Tämä helpottaa sovelluksen skaalaamista ja parantaa sen suorituskykyä, sillä JWT:t ovat itsessään varmenteita ja niiden käsittely ei vaadi tiedon hakemista palvelimen muistista.

Autentikointiprosessi alkaa, kun käyttäjä kirjautuu sisään syöttäen sähköpostiosoitteen ja salasanan. Palvelin tarkistaa, että syötetyt tiedot ovat oikein, ja jos ne ovat oikeat, luo JWT-tunnisteen, joka palautetaan asiakkaalle. Tämä tunniste sisältää käyttäjän tiedot, kuten UserID, rooli ja sähköpostiosoite. Tunniste on allekirjoitettu salaisella avaimella (JWT_SECRET), mikä varmistaa sen aitouden ja estää tunnisteen manipuloinnin (KUVA 3).

```
Koodi alkaa.

//Create JWT-token
    const token = jwt.sign(
      {
        userId: user.userid,
        role: user.role,
        email: user.email,
        name: user.name
      },
      process.env.JWT_SECRET,
      { expiresIn: "1h" }
    );

Koodi päättyy.
```

KUVA 3. JWT tokenin luonti käyttäjätiedoilla (kuvakaappaus koodista)

JWT:n käyttö vähentää tarpeen säilyttää sessiotietoja palvelimella, mikä parantaa sovelluksen suorituskykyä ja skaalautuvuutta. Tunnisteet voidaan säilyttää asiakkaan selaimessa (localStorage) tai HTTP-only cookie -muodossa, joka suojaa niitä mahdollisilta XSS-hyökkäyksiltä.

4.5 Suojatut reitit ja pääsynhallinta

Suojattujen reittien toteutus on tärkeä osa turvallista verkkosovellusta. Suojatut reitit edellyttävät käyttäjän JWT-tunnisteen validointia ennen pääsyä suojattuihin resursseihin. Jos tunniste on kelvollinen, käyttäjän tiedot purkavat tunnisteesta ja lisätään pyyntöön. Jos tunniste on vanhentunut tai virheellinen, käyttäjä ohjataan ulos ja pyyntö estetään.

Suojattuja reittejä voidaan käyttää varmistamaan, että vain tietyn roolin omaavat käyttäjät voivat päästä käsiksi tiettyihin toiminteisiin, kuten järjestelmän hallintaan tai sensitiivisiin tietoihin.

4.6 Autolistausjärjestelmän tietokantaintegraatio ja API-rakenne

Tietokantaintegraatio on rakennettu Azure SQL -tietokannan ja Node.js/Express-pohjaisen backend-palvelimen varaan. Autolistausjärjestelmään kuuluu backend-palvelin, joka käsittelee pyyntöjä ja vastaa frontend-sovelluksen tarpeisiin tietokannan kanssa kommunikoinnissa.

Tietokannassa on cars-taulu, johon tallennetaan autolistausten tiedot, kuten auton merkki, malli, vuosimalli, hinta, kuvaus ja kuvan URL-osoite. Taulun rakenne mahdollistaa kaikkien tarvittavien autolistaustietojen tallentamisen (KUVA 4).

Koodi alkaa.

```
// Add a new car
router.post("/", async (req, res) => {
  const {
    typeid, makeid, modelid, year, colorid, shadeid,
    registrationnumber, vin, price, description,
    firstregistration, inspectiondate, numberofowners,
    userid, countryid, regionid, cityid, subtypeid,
    roadworthy, sold, lastupdated, views
  } = req.body;
```

```
  if (!typeid || !makeid || !modelid || !year || !colorid || !shadeid ||
    !registrationnumber || !vin || !price || !userid || !countryid ||
    !regionid || !cityid || !subtypeid || roadworthy === undefined) {
    return res.status(400).json(
      { message: "All required fields must be filled" });
  }

  try {
    const result = await pool.query(
      `INSERT INTO cars
      (typeid, makeid, modelid, year, colorid, shadeid, registrationnumber,
      vin, price, description, firstregistration, inspectiondate,
      numberofowners, userid, countryid, regionid, cityid, subtypeid,
      roadworthy, sold, lastupdated, views)
      VALUES
      ($1, $2, $3, $4, $5, $6, $7, $8, $9,
      $10, $11, $12, $13, $14, $15,
      $16, $17, $18, $19, $20, $21, $22)
      RETURNING carid`,
      [typeid, makeid, modelid, year, colorid, shadeid, registrationnumber,
        vin, price, description || null, firstregistration || null,
        inspectiondate || null, numberofowners || null, userid, countryid,
        regionid, cityid, subtypeid, roadworthy || false, sold || false,
        lastupdated || new Date(), views || 0]
      );

      res.status(201).json(
        { message: "Car added successfully", carid: result.rows[0].carid });
    } catch (error) {
      console.error("Error adding car:", error);
      res.status(500).json({ message: "Internal server error." });
    }
});

Koodi päättyy.
```

KUVA 4. Autolistaustietojen tallentaminen tietokantaan (kuvakaappaus koodista)

Backend-palvelimelle on luotu API-reitti, joka vastaanottaa frontendiltä lähetettyjä POST-pyyntöjä ja tallentaa autolistaustiedot tietokantaan. Tämä reitti vastaanottaa tietoja kuten auton merkki, malli, vuosimalli, hinta, kuvaus ja kuva-URL, ja tallentaa ne SQL-tietokantaan. Ennen tallentamista se tarkistaa, että kaikki tarvittavat tiedot ovat mukana. Mikäli tiedot ovat puutteellisia, palautetaan virheilmoitus.

Frontend-sovellus lähettää tietoja backend-palvelimelle JSON-muodossa POST-pyynnöllä. Tämä pyyntö sisältää autolistaustiedot, kuten auton merkin, mallin, hinnan, kuvauksen ja kuvan URL-osoitteen. Frontend voi käyttää tätä API:ta uusien autojen lisäämiseksi tietokantaan. Postmanin avulla voidaan testata, että

API toimii oikein ja tietoja voidaan lisätä tietokantaan. Testauksessa varmistetaan, että oikeat tiedot lähetetään ja että tietojen lisäys onnistuu ilman virheitä.

Auton teknisten tietojen hallintaan liittyy kaksi päätoimintoa: auton teknisten tietojen hakeminen ja niiden tallentaminen tietokantaan. GET-pyyntö hakee auton tekniset tiedot, kun taas POST-pyyntö lisää niitä tietokantaan. Virheiden käsittely on olennainen osa näitä toimintoja, erityisesti silloin, kun pyynnöissä tulee virheitä, kuten virheellisiä parametreja tai puutteellisia tietoja.

Tässä Express.js-sovelluksessa on rakennettu API, joka yhdistää PostgreSQL-tietokannan autohakujärjestelmään. Sovellus tarjoaa useita reittejä, joiden avulla voidaan hakea ja suodattaa ajoneuvojen valmistajia, malleja, värejä, polttoainetyyppejä sekä teknisiä tietoja. Tietokantaoperaatiot suoritetaan käyttämällä pool-objektia, joka hallitsee PostgreSQL-yhteyksiä ja varmistaa yhteyksien tehokkaan käytön ilman tarvetta luoda uutta yhteyttä jokaista pyyntöä varten.

Kaikilla API-reiteillä on virheenkäsittelylogiikka, joka takaa palvelimen luotettavan toiminnan myös mahdollisten virheiden sattuessa. Jos tietokantakysely epäonnistuu, palvelin palauttaa virheilmoituksen ja HTTP-virhekoodin, joka auttaa käyttäjää ymmärtämään ongelman syyn. Tämä virheenkäsittely parantaa sovelluksen luotettavuutta ja käyttäjäkokemusta.

Sovelluksen hakutoiminto on rakennettu joustavaksi dynaamisella SQL-kyselyrakenteella. Käyttäjät voivat hakea ajoneuvoja useiden suodattimien, kuten merkin, mallin, värin ja hinnan, perusteella. SQL-kysely muodostetaan dynaamisesti riippuen siitä, mitkä suodattimet käyttäjä on määrittänyt. Tämä lähestymistapa mahdollistaa hakujen joustavan mukauttamisen ja suodattimien lisäämisen ilman, että erillisiä kyselyitä täytyy kirjoittaa jokaiselle suodattimelle. Kyselyissä käytetyt parametrisoidut syötteet estävät SQL-injektiohyökkäykset, mikä parantaa tietoturvaa ja estää käyttäjien syötteiden väärinkäytön.

Dynaaminen SQL-kysely ei ainoastaan mahdollista suodattimien lisäämistä lennossa, vaan se voi myös helpottaa kyselyn laajentamista uusilla suodattimilla. Tällöin voidaan käsitellä uusia hakukriteerejä ilman, että kyselyitä tarvitsee kirjoittaa kokonaan uusiksi. Tämä parantaa sovelluksen skaalautuvuutta ja suorituskykyä, erityisesti suurilla tietomäärillä.

Tulevaisuudessa voidaan parantaa SQL-kyselyn tehokkuutta ja tietoturvaa muun muassa lisäämällä indeksejä käytettäville sarakkeille ja käyttämällä parametrisoituja kyselyjä SQL-injektioiden estämiseksi. Tämä lähestymistapa tarjoaa joustavan, skaalautuvan ja tehokkaan tavan käsitellä hakutuloksia, jotka perustuvat käyttäjän antamiin suodattimiin.

4.7 Johtopäätökset

JWT tarjoaa turvallisen ja skaalautuvan tavan autentikoida käyttäjiä verkkosovelluksissa ja hallita pääsyä suojattuihin resursseihin. Sen käyttäminen mahdollistaa käyttäjätunnistuksen ja roolipohjaisen pääsynhallinnan toteuttamisen tehokkaasti ja turvallisesti.

Rekisteröinti, kirjautuminen ja roolipohjainen pääsynhallinta toimivat odotetusti ja täyttivät turvallisuusvaatimukset. Käyttäjien roolit tunnistettiin ja validoitiin oikein, ja pääsy suojattuihin reitteihin oli rajattu ainoastaan niille käyttäjille, joilla oli asianmukaiset oikeudet.

4.8 React-frontendin rakentaminen ja alkuvaiheen kehitystyö

Projektin alkuvaiheessa frontendin kehitystä suunniteltiin toteutettavaksi perinteisen Reactin avulla, käyttäen Create React Appia (CRA). Nopeasti kuitenkin huomattiin, että React-kehittäjät suosittelevat frameworkien, kuten Next.js:n tai Remixin, käyttöä perinteisen CRA:n sijaan. Tässä osuudessa käydään läpi frontend-kehityksen alkuvaiheet, Next.js:n valinnan syyt ja kehitykseen liittyvät ratkaisut.

Frontend-kehitys aloitettiin luomalla React-projekti käyttäen CRA:ta. Se tarjoaa yksinkertaisen tavan aloittaa React-sovellus ilman monimutkaista konfigurointia. Se luo valmiin sovelluspohjan, joka sisältää Webpackin, Babelin ja hot reload -ominaisuuden, mikä helpottaa kehitystyötä. Ensimmäisessä vaiheessa toteutettiin perusrakenne, jossa määriteltiin komponenttirakenne, tilanhallinnat ja tyylitykset. Sovelluksen eri osat, kuten lomakkeet, hakutoiminnot ja navigaatio, jaettiin erillisiin komponentteihin. Reactin useState- ja useEffect-hookeja käytettiin tilojen

hallintaan ja erilaisiin API-kutsuihin. Daisyui tailwindcss:n kanssa mahdollisti responsiivisen ja modulaarisen ulkoasun rakentamisen nopeasti ja vaivatta.

Kehitysvaiheen alussa ilmeni useita haasteita, jotka olivat merkittäviä sovelluksen skaalautumisen ja suorituskyvyn kannalta. Perus React-sovellus käyttää client-side rendering (CSR) -menetelmää, mikä tarkoittaa, että selain lataa ensin JavaScriptin ja vasta sen jälkeen renderöi sivun sisällön. Hakukoneoptimoinnin kannalta tämä on puutteellista, sillä selain näyttää tyhjän html -dokumentin oikean sisällön sijaan. React sovellukset myös latautuu aluksi hitaasti, koska kaikki sisältö renderöidään selainpuolella, joka vaikuttaa erityisesti ensimmäisen latauksen nopeuteen. Reactin tarjoama reititysjärjestelmä tarvitsee enemmän manuaalista konfigurointia frameworkeihin verrattuna, joissa se hoituu automaattisesti tiedostorakenteen perusteella. Näiden ongelmien vuoksi React-kehittäjien suositukset frameworkien käytöstä nousivat esille, ja aloimme vertailla Next.js:n ja Remixin tarjoamia vaihtoehtoja.

4.8.1 Next.js:n ja Remixin vertailu

Next.js ja Remix ovat moderneja React-pohjaisia frameworkeja, jotka on kehitetty ratkaisemaan monia perinteisen React -sovelluksen haasteita. Molemmat tarjoavat server-side renderöinnin (SSR), joka mahdollistaa sisällön esilataamisen ennen kuin se näytetään käyttäjälle. Tämä parantaa merkittävästi sovelluksen latausnopeutta ja hakukoneoptimointia CRA-sovellukseen verrattuna.

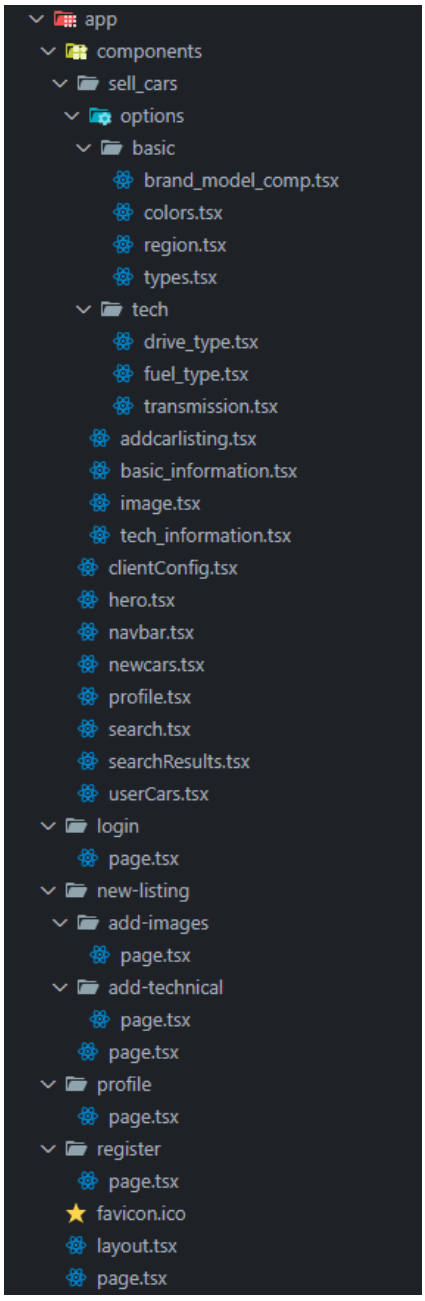
Next.js tarjoaa useita etuja, jotka tekevät siitä houkuttelevan vaihtoehdon verkkosovellusten kehittämiseen. Sen tiedostopohjainen reititysjärjestelmä yksinkertaistaa sovelluksen navigoinnin hallintaa, sillä reitit muodostuvat automaattisesti hakemistorakenteen perusteella. Lisäksi Next.js tukee sekä server-side renderingiä että static site generationia (SSG), mikä mahdollistaa nopeamman sisällön latauksen ja vähentää palvelimen kuormitusta. Sovelluksen suorituskykyä tehostavat myös erilaiset automaattiset optimoinnit, kuten kuvanlatauksen parantaminen Next.js Image -komponentilla. Erityisesti Vercelin kanssa Next.js integroituu saumattomasti, mikä helpottaa sovelluksen julkaisemista ja skaalaamista.

Remix puolestaan tarjoaa erinomaiset työkalut datan latauksen hallintaan, mikä mahdollistaa tehokkaamman lomakekäsittelyn ja interaktiivisten sovellusten kehittämisen. Se antaa tarkemman hallinnan server-renderöinnin ja datan haun suhteen, mikä voi olla erityisen hyödyllistä monimutkaisemmissa sovelluksissa, joissa tarvitaan joustavuutta tietokantakyselyiden ja sivulatausten optimointiin.

Vaikka Remix tarjoaa kiinnostavia ominaisuuksia, sen ekosysteemi on tällä hetkellä pienempi verrattuna Next.js:ään. Next.js:n laaja käyttäjäyhteisö ja vankka dokumentaatio tekevät siitä luotettavan valinnan kehittäjille, jotka haluavat rakentaa suorituskykyisiä ja helposti ylläpidettäviä React-sovelluksia. Tästä syystä Next.js valittiin projektin frontend-teknologiaksi, sillä se mahdollisti skaalautuvan ja tehokkaan sovellusarkkitehtuurin luomisen ilman ylimääräistä konfigurointia.

4.8.2 Komponenttirakenne ja käyttöliittymän suunnittelu

Tavallisen React-sovelluksen mukaisesti projekti on jaettu uudelleenkäytettäviin komponentteihin, jotka mahdollistavat sovelluksen modulaarisen rakenteen. Samalla voidaan määrittää mitkä sovelluksen osat renderöidään palvelimella ja mitkä käyttäjän puolella. (Kuva 5).



KUVA 5. Frontend komponenttirakenne (kuvakaappaus koodista)

4.8.3 Perustietojen valinnat ilmoituksen jättämisessä

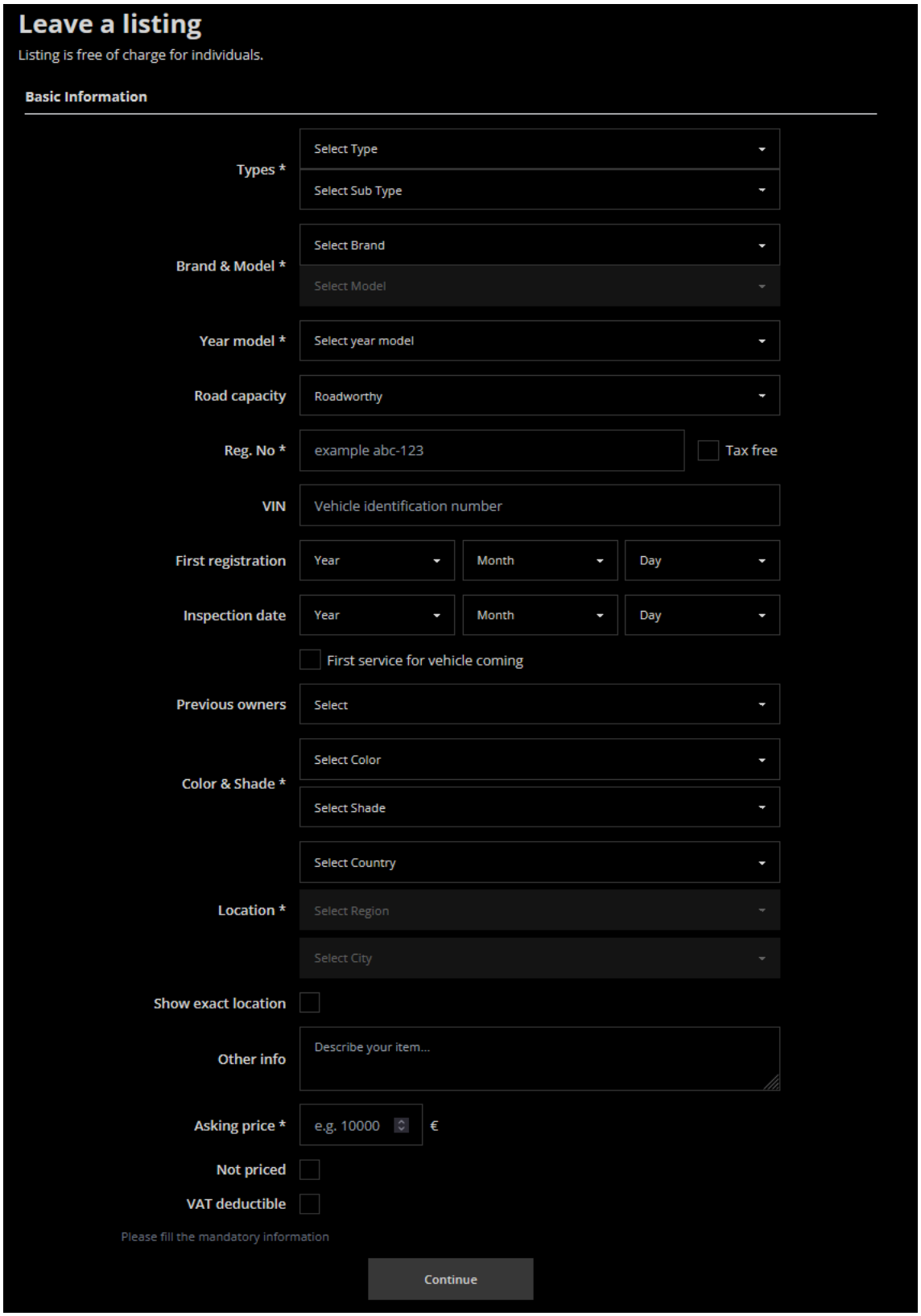
Ilmoituksen jättäminen käytetyille autoille ja varaosille edellyttää käyttäjältä useiden perustietojen täyttämistä. Näiden tietojen avulla voidaan varmistaa, että ilmoitus on kattava ja sisältää kaikki olennaiset tiedot auton ominaisuuksista ja sijainnista. Ensimmäiseksi käyttäjän tulee valita ajoneuvon tyyppi ja alityyppi, joiden avulla määritellään, onko kyseessä esimerkiksi henkilöauto, pakettiauto tai

moottoripyörä. Seuraavaksi valitaan merkki ja malli, mikä auttaa ostajia löytämään tarkasti haluamansa ajoneuvotyypin. Ajoneuvon valmistusvuosi on olennainen tieto, sillä se vaikuttaa suoraan auton arvoon ja houkuttelevuuteen markkinoilla. Lisäksi käyttäjän on ilmoitettava, onko ajoneuvo tieliikennekelpoinen (roadworthy), mikä kertoo ostajalle, voiko autoa käyttää välittömästi liikenteessä. (Kuva 6).

Ajoneuvon tunnistamisen varmistamiseksi ilmoitukseen lisätään rekisterinumero ja ajoneuvon valmistenumero (VIN). Rekisterinumero on pakollinen kenttä, mutta käyttäjä voi myös ilmoittaa, että auto on verovapaa (tax free). Tämä on erityisen tärkeää tietyissä tapauksissa, kuten yrityskäytössä tai maahantuoduissa ajoneuvoissa. (Kuva 6).

Myyjän tulee valita ajoneuvon sijainti, mikä sisältää maan, alueen ja kaupungin. Demo versiossa on maina Suomi ja Ruotsi, josta Suomesta jokainen maakunta ja Ruotsista vain muutama. Jokaisella maakunnalla vain yksi kaupunki, mutta niitä saa lisättyä suoraan tietokantaan, jos tarve sen vaatii. Myyjä voi myös valita, haluaako hän näyttää tarkan sijainnin, mikä voi olla hyödyllistä nopeiden kaupantekojen kannalta. (Kuva 6).

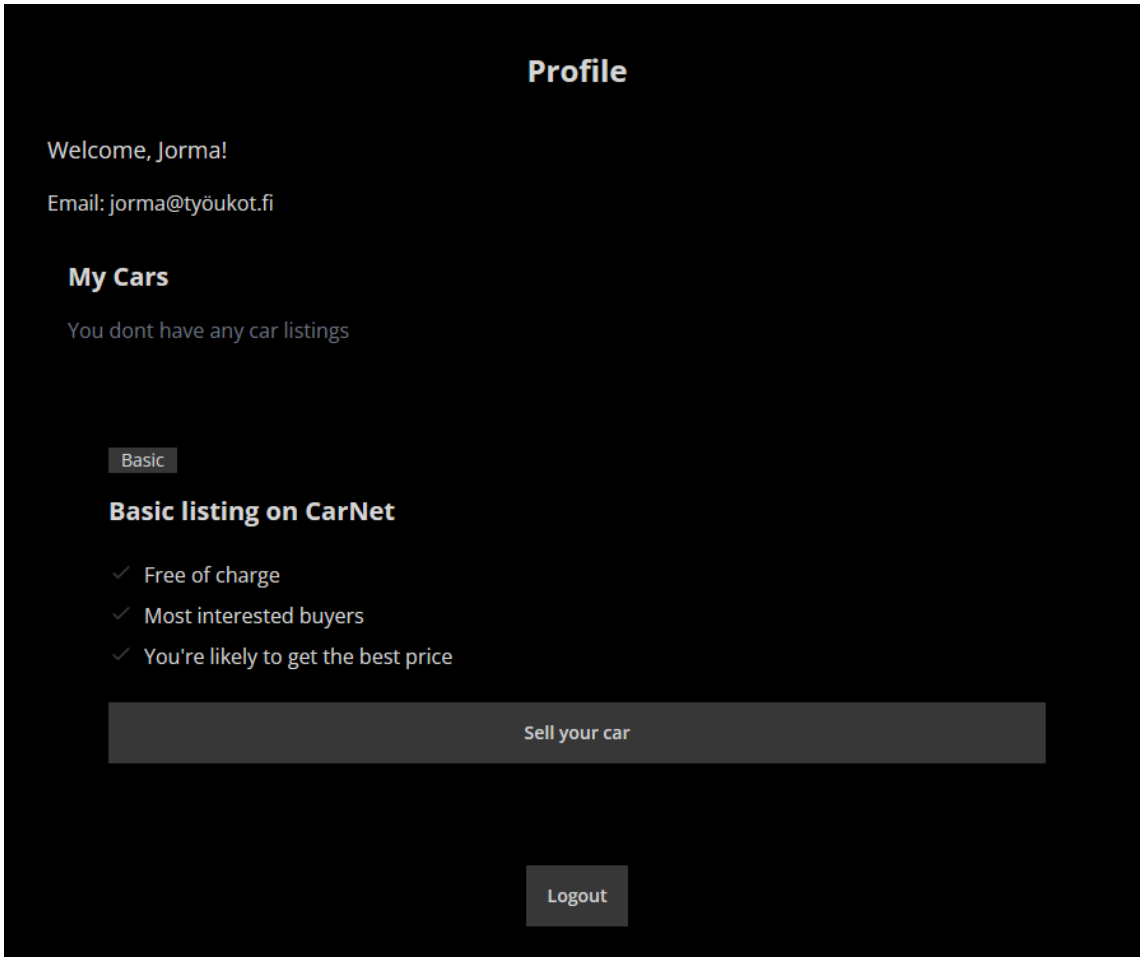
Perustietojen huolellinen täyttäminen varmistaa, että ostajat saavat kattavat tiedot ajoneuvosta ennen yhteydenottoa. Selkeä ja tarkka ilmoitus vähentää turhia kyselyitä ja nopeuttaa kaupantekoa.



KUVA 6. Myynti-ilmoituksen perustietojen lisääminen (kuvakaappaus sivustolta)

4.8.4 Käyttäjäprofiilin sisältö

Profiilisivulla käyttäjä näkee tervetuloviestin, joka sisältää hänen nimensä ja sähköpostiosoitteensa. Lisäksi sivulla näytetään käyttäjän julkaistut autolistaukset, mutta jos listauksia ei ole, käyttäjälle ilmoitetaan asiasta selkeästi. Sivulla on myös mahdollisuus aloittaa auton myynti ilmaiseksi, mikä korostaa palvelun hyötyjä, kuten laajaa ostajakuntaa ja mahdollisuutta saada paras mahdollinen hinta. Käyttäjä voi myös kirjautua ulos palvelusta helposti, mikä lisää käyttömukavuutta ja turvallisuutta. (Kuva 7).



KUVA 7. Käyttäjäprofiili näyttää omat myynti-ilmoitukset ja mahdollisuus uuden luomiseen (Kuvakaappaus sivustolta)

5 POHDINTA

5.1 Suunnitteluvaihe

Projektin alkuvaiheessa määriteltiin verkkosovelluksen päämäärät ja tekniset vaatimukset. Tavoitteena oli kehittää käytettyjen autojen ja varaosien myyntiin soveltuva alusta, joka tarjoaisi käyttäjille helppokäyttöisen ilmoitusten hallinnan, hakutoiminnot sekä visuaalisesti miellyttävän ja responsiivisen käyttöliittymän. Alkuperäisen suunnitelman mukaan sovellus oli tarkoitus rakentaa perinteisellä Reactilla, mutta jo varhaisessa vaiheessa ilmeni tarve paremmin optimoidulle ja skaalautuvalle ratkaisulle. Tämän seurauksena päädyttiin käyttämään Next.js-kehystä, joka mahdollisti server-side renderöinnin ja paransi hakukoneoptimointia (SEO).

Tässä vaiheessa valittiin myös sovelluksen keskeiset teknologiat. Backend päätettiin rakentaa Node.js:n ja Express.js:n avulla RESTful API -arkkitehtuurilla, ja tietokannaksi valittiin PostgreSQL, koska se tarjosi skaalautuvan ja tehokkaan tavan hallita relaatiotietoja. Maksujärjestelmän toteuttamiseksi valittiin Stripe, ja pilvipalveluna käytettiin Microsoft Azurea, joka tarjosi luotettavan infrastruktuurin sovelluksen isännöintiin. Myöhemmin kuitenkin ilmeni, ettei Azuren CI/CD putkea voitu toteuttaa puutteellisten oikeuksien takia. Siirsimme pilvipalvelut ja tietokannan Verceliin.

5.2 Toteutusvaihe

Sovelluksen kehitys eteni iteratiivisesti siten, että aluksi toteutettiin perustoiminnot, kuten käyttäjäautentikointi ja ilmoitusten lisääminen. Autentikointiin käytettiin JWT-pohjaista tunnistautumista, joka varmisti käyttäjien turvallisen kirjautumisen ja pääsynhallinnan. Ilmoitusten lisäämiseksi luotiin dynaaminen lomake, jossa käyttäjä pystyi syöttämään autonsa tiedot, kuten merkin, mallin, hinnan ja tekniset tiedot. Lisäksi toteutettiin kuvien latausominaisuus, jossa kuvat tallennettiin Vercel Blob Storageen, ja tietokantaan tallennettiin niiden URL-osoitteet.

Frontendin osalta hyödynnettiin Tailwind CSS:ää ja DaisyUI:ta, jotka mahdollistivat nopean ja yhtenäisen käyttöliittymän rakentamisen. Hakutoiminnallisuus toteutettiin dynaamisilla SQL-kyselyillä, joiden avulla käyttäjät pystyivät suodattamaan autoja eri kriteerien, kuten hinnan, vuoden ja merkin perusteella.

Järjestelmän testaus suoritettiin useassa vaiheessa. Automatisoitujen Jest- ja Cypress-testien avulla varmistettiin, että sovelluksen eri komponentit toimivat odotetusti. Manuaalisessa testauksessa käyttäjäpalautteen perusteella tehtiin parannuksia käyttöliittymään ja hakutoiminnallisuuteen, jotta ne palvelisivat paremmin loppukäyttäjiä.

5.3 Haasteet ja ratkaisut

Projektin aikana kohdattiin useita haasteita, kuten kuvien hallinta ja dynaaminen hakutoiminnallisuus. Yksi merkittävä haaste oli Next.js:n kuvankäsittelyn konfigurointi, koska ulkoisten kuvapalveluiden, kuten Vercelin blob-tallennuksen, käyttö vaati lisäasetuksia `next.config.js`-tiedostossa. Tämä ratkaistiin lisäämällä sallitut kuva-hostit asetuksiin, mikä mahdollisti kuvien lataamisen suoraan tietokannasta.

Toinen merkittävä haaste liittyi käyttäjäkohtaisen suodatuksen toteutukseen hakutoiminnossa. Aluksi kaikki autot haettiin tietokannasta, mutta myöhemmin lisättiin mahdollisuus suodattaa autot käyttäjä-ID:n perusteella, jotta käyttäjät pystyivät hallinnoimaan omia ilmoituksiaan. Tämä toteutettiin lisäämällä `userid`-parametri hakupyyntöihin ja mukauttamalla SQL-kyselyitä tukemaan tätä suodatusta.

Käyttäjärekisteröinnissä ilmeni ongelma, jossa käyttäjän roolikenttä jäi NULL-arvoiseksi, ellei roolia määritetty erikseen. Tämä saattoi aiheuttaa käyttöoikeuksiin liittyviä epäselvyyksiä ja rajoituksia. Haaste ratkaistiin lisäämällä SQL-kyselyihin oletusarvoinen rooli, mikä varmisti, että jokainen rekisteröity käyttäjä sai automaattisesti määritellyn käyttöoikeustason.

Autentikointiin liittyi myös useita haasteita, erityisesti JWT-tunnisteiden turvallinen käsittely. Yksi suurimmista riskeistä oli tunnisteiden varastaminen, erityisesti XSS-hyökkäysten kautta. Tämän vuoksi JWT-tunnisteiden säilyttäminen HTTP-only cookieissa valittiin ratkaisuksi, sillä se estää asiakaspuolen skriptien pääsyn

tunnisteisiin. Lisäksi tokenien vanheneminen vaati tehokkaan refresh token -mekanismin, jotta käyttäjä ei joutuisi kirjautumaan sisään jatkuvasti, mutta samalla turvallisuus säilyisi korkealla tasolla.

5.4 Lopputulos ja tavoitteiden saavuttaminen

Vaikka projektiin liittyi aikataulullisia haasteita, suurin osa tavoitteista saavutettiin. Sovelluksesta rakennettiin toimiva verkkokauppa-alusta autojen myyntiin, ja se mahdollistaa ilmoitusten luomisen, muokkaamisen sekä hakemisen tehokkaasti. Käyttäjäkokemus optimoitiin responsiivisella käyttöliittymällä, ja turvallisuus varmistettiin JWT-autentikoinnilla sekä tietoturva-auditoiduilla API-pyynnöillä.

Joidenkin ominaisuuksien, kuten kehittyneempien analytiikka- ja suositusjärjestelmien, toteuttaminen jäi aikataulusyistä myöhemmäksi kehitysvaiheeksi. Jatkokehityksessä keskityttäisiin erityisesti parempaan käyttäjäprofiilien hallintaan, lisäominaisuuksien, kuten ilmoitusten suosikeiksi merkitsemisen, sekä monipuolisemman hakualgoritmin kehittämiseen.

Projektin lopputuloksena syntyi skaalautuva ja käyttäjäystävällinen verkkosovellus, joka täyttää alustan perusvaatimukset ja tarjoaa pohjan jatkokehitykselle. Tämä työ osoitti, kuinka modernit web-teknologiat voivat tehostaa markkinapaikan rakentamista, ja se tarjosi arvokasta oppia erityisesti Next.js:n, PostgreSQL:n ja pilvipalveluiden käytöstä verkkokehityksessä.

LÄHTEET

AWS s.a. a. What is Web Hosting? Luettavissa: https://aws.amazon.com/what-is/web-hosting/ Luettu 3.2.2025.

AWS s.a. b. What is a Database? Luettavissa: https://aws.amazon.com/what-is/database/ Luettu 6.2.2025.

Carrol, M. & Hanlon, R. 14.2.2025. Sunsetting Create React App. Luettavissa: https://react.dev/blog/2025/02/14/sunsetting-create-react-app Luettu 25.1.2025.

Codeacademy Team 7.11.2022. Back-End Web Architecture. Luettavissa: https://www.codecademy.com/article/back-end-architecture Luettu 25.1.2025.

Dhaduk, H. 31.7.2023. Frontend Architecture and How to Improve its Design. Luettavissa: https://www.simform.com/blog/frontend-architecture Luettu 24.1.2025.

DaisyUI s.a. Install daisyUI as a Tailwind plugin. Luettavissa: https://daisyui.com/docs/install/ Luettu 1.2.2025.

Railway Docs s.a. Railway documentation. Luettavissa: https://docs.railway.com/ Luettu 23.2.2025.

Grezeszak, B. s.a. Top 10 Front-end Design Principles for Developers. Luettavissa: https://www.toptal.com/front-end/front-end-design-principles Luettu 25.1.2025.

Gillis, A. 2024. What is a database (DB)? Luettavissa: https://www.techtarget.com/searchdatamanagement/definition/database Luettu 6.2.2025.

GeeksForGeeks 2025. What is CI/CD? Luettavissa: https://www.geeksforgeeks.org/what-is-ci-cd/ Luettu 13.2.2025.

Google Cloud s.a. What are Containers? Luettavissa: https://cloud.google.com/learn/what-are-containers Luettu 11.2.2025.

GitLab s.a. What is CI/CD? Luettavissa: https://about.gitlab.com/topics/ci-cd/ Luettu 13.2.2025.

Git s.a. Documentation. Luettavissa: https://git-scm.com/doc Luettu 7.2.2025.

Github Docs s.a. Help for wherever you are on your GitHub journey. Luettavissa: https://docs.github.com/en Luettu 8.2.2025.

Ken, A. 2023. Backend: What is it and what is it used for? Luettavissa: https://www.gluo.mx/en-US/blog/backend-que-es-y-para-que-sirve Luettu 26.1.2025.

Liimatta, A. 21.5.2021. Pilvipalvelut: tiedä tärkeimmät termit. Luettavissa: https://www.tietoevry.com/fi/blogi/2021/05/pilvipalvelut-tieda-tarkeimmat-termit/ Luettu 3.2.2025.

McCoy, L. s.a. Microsoft Azure Explained: What It Is and Why It Matters. Luettavissa: https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/ Luettu 24.1.2025.

MDN Web Docs 20.12.2024. Express/Node introduction. Luettavissa: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction Luettu 6.2.2025.

Node.js s.a. Introduction to Node.js. Luettavissa: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs Luettu 6.2.2025

Next.js s.a. What is Next.js? Luettavissa: https://nextjs.org/docs Luettu 8.2.2025.

Ramotion 28.4.2023. The Most Common Frontend Design Patterns Explained. Luettavissa: https://www.ramotion.com/blog/frontend-design-patterns Luettu 25.1.2025.

RedHat 14.3.2022. What are cloud services? Luettavissa: https://www.redhat.com/en/topics/cloud-computing/what-are-cloud-services Luettu 3.2.2025.

RedHat 6.4.2023. Understanding containers. Luettavissa: https://www.redhat.com/en/topics/containers Luettu 11.2.2025.

RedHat 12.12.2023. What is CI/CD? Luettavissa: https://www.redhat.com/en/topics/devops/what-is-ci-cd Luettu 13.2.2025.

Sharma, M. 27.11.2024. What is application hosting? Luettavissa: https://www.techradar.com/how-to/what-is-application-hosting Luettu 3.2.2025.

Solarwinds s.a. What is Container Technology? Luettavissa: https://www.solarwinds.com/resources/it-glossary/container Luettu 11.2.2025.

Tailwind CSS s.a. Get started with Tailwind CSS. Luettavissa: https://tailwindcss.com/docs/installation/framework-guides Luettu 1.2.2025.

Visual Studio Code s.a. Your code editor. Redefined with AI. Luettavissa: https://code.visualstudio.com Luettu 8.2.2025.

Vercel s.a. Your complete platform for the web. Luettavissa: https://vercel.com/ Luettu 22.2.2025.

Volle, A. 11.3.2025. Web application. Luettavissa: https://www.britannica.com/topic/Web-application Luettu 24.1.2025.

Yasar, K. 1.11.2024. What is web application (web apps) and its benefits? Luettavissa: https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app Luettu 24.1.2025.