Tuomas Mellin

Web-pohjainen työkalu IoT-laitteiden asennukseen ja käyttöönottoon

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

2.5.2025

Tiivistelmä

Tekijä:	Tuomas Mellin
Otsikko:	Web-pohjainen työkalu IoT-laitteiden asennukseen ja käyttöönottoon
Sivumäärä:	47 sivua + 4 liitettä
Aika:	2.5.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Älykkäät IoT-järjestelmät
Ohjaajat:	Saana Vallius, lehtori

Insinöörityössä suunniteltiin ja kehitettiin web-pohjainen käyttöönottotyökalu, jonka tavoitteena oli helpottaa IoT-laitteiden asennusta ja käyttöönottoa Skipperi Fleet -palvelussa. Käyttöönottotyökalun tavoitteena oli vähentää asennusprosessin virhealttiutta, nopeuttaa käyttöönottoa sekä parantaa asiakaskokemusta. Käyttöönottotyökalun avulla haluttiin varmistaa, että käyttäjät voivat suorittaa asennuksen itseohjautuvasti ja ilman ulkopuolista apua.

Työssä toteutettiin selkeä, vaiheittain etenevä käyttöliittymä Reactilla, joka ohjasi käyttäjän läpi koko käyttöönottoprosessin. Käyttöliittymä oli suunniteltu niin, että se oli intuitiivinen ja helppokäyttöinen. Lisäksi käyttöönottotyökalun oheen kehitettiin kattavat ohjeet, jotka auttoivat käyttäjää navigoimaan koko käyttöönottoprosessin läpi sekä ratkomaan mahdollisia virhetilanteita. Projektissa oli tarkkaan määritelty 10 viikon projektisuunnitelma, joka ohjasi kehitystyötä. Projektissa panostettiin merkittävästi tutkimus- ja suunnitteluvaiheisiin, joiden aikana tutustuttiin käytettäviin työkaluihin ja teknologioihin, määriteltiin järjestelmän arkkitehtuuri ja vaatimukset, suunniteltiin käyttöliittymän rakenne sekä testaus. Yksi projektin keskeisistä tavoitteista olikin perehtyä käytettäviin työkaluihin ja niiden tuottamaan dataan sekä laatia niistä dokumentaatiota myös jatkokehitystä varten. Työ toteutettiin moderneilla teknologioilla kuten Reactilla, Laravelilla sekä TypeScriptillä. Työkalu rakennettiin modulaariseksi, mikä mahdollisti sen helpon laajennettavuuden ja kehittämisen myös jatkossa.

Testaus sisälsi yksikkötestauksen, ominaisuustestauksen sekä kattavan käyttäjätestauksen. Tuloksista ilmeni, että työkalun avulla asennusprosessia saatiin selkeytettyä ja nopeutettua merkittävästi. Samalla työ paljasti tärkeitä havaintoja ja näkökulmia, joita ei olisi tullut ilmi ilman tämänkaltaista kokonaisvaltaista ratkaisua.

Työn katsottiin täyttävän kaikki sille asetetut tavoitteet ja luovan hyvän pohjan jatkokehitykselle. Työn tuloksia voidaan hyödyntää jatkokehityksessä.

Avainsanat: IoT, käyttöönottotyökalu, ohjelmistokehitys

Abstract

Author:	Tuomas Mellin
Title:	Web-based tool for IoT device installation and onboarding
Number of Pages:	47 pages + 4 appendices
Date:	2 May 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Smart IoT Systems
Supervisors:	Saana Vallius, Senior Lecturer

In this final year project, a web-based onboarding tool was designed and developed to simplify the installation and configuration of IoT devices for Skipperi Fleet. The aim was to reduce the error-proneness of the installation process, streamline the onboarding of new boats, and enhance overall customer experience. The tool was created to enable users to complete the installation independently without external assistance or guidance.

A clear, step-by-step user interface was developed using React. The web-interface was designed to be intuitive and easy to use. In addition, formal instructions were created to accompany the tool, helping users navigate the process and resolve potential issues. The project followed a carefully defined 10-week plan that guided the development work. Significant effort was dedicated to the research and planning stages, during which the necessary tools were evaluated and the system architecture, requirements, UI design, and testing protocols were defined. One of the key objectives of the project was to become familiar with the tools and technologies used and the data they generate, as well as to document them for future development. The tool was built using modern technologies such as React, Laravel, and TypeScript.

Testing included unit testing, feature testing, and comprehensive user testing. The results indicated that the onboarding process was significantly streamlined and accelerated because of the tool. Additionally, the project revealed valuable insights and findings that would not have emerged without such a comprehensive solution.

The project successfully achieved all its objectives and laid a strong foundation for future development. The results of the project can be used to improve future development efforts.

Keywords: IoT, Onboarding wizard, software development

Sisällys

Lyhenteet

HTTP: Hypertext Transfer Protocol. Protokolla, joka mahdollistaa verkkosivujen ja muiden resurssien tietojen siirtämisen selaimen ja palvelimen välillä.

URL: Uniform Resource Locator. URL on yksilöllinen tunniste, jota käytetään resurssien, kuten verkkosivujen, tiedostojen tai muiden verkossa olevien sisältöjen osoittamiseen. Se kertoo verkkoselaimelle, mistä resurssista on kyse ja miten siihen päästään käsiksi.

TDD: Test-Driven Development. Ohjelmistokehityksen lähestymistapa, jossa koodi kirjoitetaan testien avulla. Ennen ohjelmakoodin kirjoittamista kehittäjä luo testit, jotka määrittävät, mitä koodin tulee tehdä.

REST: Representational State Transfer. Verkkopalveluiden arkkitehtuurityyli, jossa resurssit tunnistetaan URL-osoitteilla ja niitä käsitellään standardoiduilla HTTP-operaatioilla.

SPA: Single Page Application. Verkkosovellus, joka toimii yhdellä sivulatauksella ja päivittää vain tarvittavat osat sivusta ilman koko sivun uudelleenlatausta.

DOM: Document Object Model. Verkkosivun rakenteen ohjelmointirajapinta, joka kuvaa HTML-dokumentin elementit puumaisena rakenteena. DOM mahdollistaa JavaScript ohjelmoinnin kautta sivun sisällön, rakenteen ja tyylien käsittelyn ja muokkaamisen dynaamisesti.

1 Johdanto

Teknologian kehittyessä IoT-laitteet ovat yhä keskeisemmässä roolissa eri toimialoilla. Ne mahdollistavat laitteiden etäseurannan, automatisoinnin ja tiedon keräämisen tavalla, joka ei aiemmin ole ollut mahdollista. IoT-teknologian potentiaali on valtava, mutta sen laajamittainen käyttöönotto tuo mukanaan myös omat haasteensa. Erityisesti käyttöönoton sujuvuus on kriittinen tekijä – monesti IoT-laitteita asentavat ja käyttöönottoprosessista vastaavat henkilöt eivät ole teknisiä asiantuntijoita, mikä voi aiheuttaa viiveitä, virheitä ja turhautumista. Tämän vuoksi tarvitaan ratkaisuja, jotka tekevät asennusprosessista mahdollisimman selkeän, intuitiivisen ja helpon myös ei-teknisille käyttäjille.

Tämän opinnäytetyön tarkoituksena on kehittää älykäs käyttöönottojärjestelmä helpottamaan IoT-laitteiden käyttöönottoa erityisesti ei-teknisen henkilön näkökulmasta. Keskeisenä osana opinnäytetyötä on kehittää web-pohjainen käyttöönottotyökalu, joka opastaa käyttäjää vaihe vaiheelta varmistaen, että kaikki asennusvaiheet suoritetaan oikein.

Opinnäytetyö toteutettiin projektina Harmaja 10 Oy:lle, jonka pääasiallinen tuote on jakamistalouden alusta, veneiden yhteiskäyttöpalvelu sekä vertaisvuokrausalusta Skipperi. Yrityksellä oli tarve kehittää ratkaisu, joka helpottaa ja nopeuttaa IoT-laitteiden käyttöönottoa. Ratkaisua tarvittiin erityisesti tilanteisiin, joissa asennuksista vastaavat jälleenmyyjät tai huoltoliikkeet, joiden tekninen osaaminen vaihtelee. Koska käyttöönotot ovat usein kiireellisiä ja kenttäolosuhteet vaihtelevat, haluttiin kehittää järjestelmä, joka toimii luotettavasti ja vähentää käyttäjän epävarmuutta.

Opinnäytetyössä esitellään toimeksiannon tausta, käytetyt teknologiat ja työkalut sekä lopputyöprojektin suunnittelu ja toteutus. Testaus ja laadunvarmistus ovat myös tärkeässä roolissa, sillä lopputuloksen on oltava sekä teknisesti toimiva, että loppukäyttäjälle helppokäyttöinen. Opinnäytetyönä kehitettiin olemassa olevaan web-portaaliin lisäosa, joka hyödyntää IoT-laitteiden dataa asennuksen

onnistumisen analysoimiseksi, tunnistaa mahdolliset virheet ja tarjoaa käyttäjälle selkeät ohjeet niiden korjaamiseksi. Toimeksiannon tavoitteena oli vähentää inhimillisiä virheitä, nopeuttaa prosessia sekä parantaa asiakaskokemusta. Lisäksi haluttiin kerätä IoT-laitteiden tuottamaa dataa jatkokehitystä ja suunnittelua varten, luoden pohjaa älykkäämmälle ja datalähtöiselle toimintamallille tulevaisuudessa.

2 Toimeksianto ja tausta

Tässä osiossa kuvataan, millaiseen toimintaympäristöön ja minkälaiselle yritykselle opinnäytetyön toimeksianto tehtiin. Tarkastelussa ovat sekä yrityksen toimiala että ne olosuhteet ja tarpeet, joiden pohjalta tarve kehitystyölle syntyi. Tavoitteena on antaa lukijalle selkeä kuva siitä, minkälaiseen ympäristöön ja käyttötarkoitukseen tässä opinnäytetyössä kehitettävä järjestelmä pohjautuu.

2.1 Skipperi

Skipperi on Suomessa vuonna 2017 perustettu veneilyalan kasvuyritys. Skipperin ensisijainen tuote on veneiden yhteiskäyttöpalvelu Skipperi Fleet. Skipperi käynnisti toimintansa vuonna 2017 veneiden vertaisvuokrauspalvelu Skipperi Rentillä. Skipperi Rentissä yksityishenkilöt pääsivät vuokraamaan omia veneitään toisille yksityishenkilöille. Vuonna 2018 Skipperi lanseerasi kaupunkivenepalvelun, joka tarjosi aluksi kymmenen soutuvenettä käytettäväksi kausimaksulla. Vuonna 2019 kaupunkivenepalvelu laajeni soutuveneistä 15 moottoriveneeseen. Moottoriveneitä tarjottiin alussa pääkaupunkiseudun, Tampereen sekä Turun alueella. Vuonna 2021 kaupunkiveneet kokivat brändiuudistuksen ja niistä tuli nykyisin tunnettu Skipperi Fleet. Nykyisin Skipperi Fleet toimii yhteensä seitsemässä eri maassa ja yhteiskäyttöveneitä on lähes 400. Nykyisellään Skipperin liiketoiminnan ytimeen kuuluu Skipperi Rent sekä Skipperi Fleet. Skipperin asiakkaita ovat pääasiassa kuluttaja-asiakkaat, mutta Skipperi Fleet on myös suosittu yritysten antama työsuhde-etuus. [1.]

Skipperi on kehittänyt veneilyn saralla useita innovatiivisia ratkaisuja. Skipperi on yhteiskäyttövenepalvelun sekä yksityisveneiden vertaisvuokrausalustan lisäksi kehittänyt useita veneilyyn liittyviä ominaisuuksia kuten Kohteet sekä Geofencing. Nämä ominaisuudet ovat alansa ensimmäisiä ja osoitus Skipperin sitoutumisesta veneilyn modernisointiin ja käyttäjäkokemuksen parantamiseen. [2.]

Geofencing eli aluerajaus on Skipperin kehittämä ominaisuus, joka hyödyntää GPS-dataa ja alueellisia rajoituksia turvallisuuden lisäämiseksi vesillä. Skipperin geofencing-ominaisuudessa veneen paikkatieto yhdistetään ennalta määriteltyihin aluerajoihin. Kun vene saapuu rajatun alueen sisälle, järjestelmä antaa hälytyksen, joka ohjeistaa esimerkiksi nopeuden alentamista tai varovaisempaa liikkumista. Ominaisuus on ensimmäinen laatuaan huviveneilyssä ja tarjoaa työkaluja sekä turvallisuuden että ympäristön huomioimiseen. [3.]

Skipperi Kohteet on ominaisuus, jossa Skipperi Fleet -jäsenet voivat lisätä suosikkiveneilykohteitaan kartalle ja jakaa niistä tietoa muille käyttäjille. Jäsenet voivat kirjata esimerkiksi rantautumispaikkoja, kiinnittymistapoja ja paikallisia palveluita, kuten tulipaikkoja tai käymälöitä kartalle. Näin tieto vesistöjen kohteista pysyy ajantasaisena ja uusien kohteiden löytäminen helpottuu, mikä tukee veneilyn suunnittelua ja turvallisuutta mutta auttaa myös uusia veneilijöitä harrastuksen piiriin. Molemmat näistä ominaisuuksista on kehitetty ainoastaan Skipperi Fleet -palvelun käyttäjille. [4.]

2.2 Skipperi Fleet

Skipperi Fleet on veneiden yhteiskäyttöpalvelu, joka toimii Suomessa, Ruotsissa, Norjassa, Tanskassa, Uudessa-Seelannissa, Kanadassa sekä Yhdysvalloissa. Fleet-palvelu on niin sanottu yhteiskäyttöpalvelu, jossa asiakas ei omista veneettä vaan käyttää sitä yhdessä muiden palvelun asiakkaiden kanssa. Tämä eroaa perinteisestä venevuokrauksesta siten, että käytöstä ei makseta käyttökertojen perusteella, vaan palvelussa on kiinteä kuukausimaksu, joka mahdollistaa rajattoman veneilyn ilman lisäkustannuksia. Tämä malli mahdollistaa joustavaa käyttöä ilman omistajuuden tuomia velvoitteita ja kustannuksia. Yhteiskäyttöpalvelu eroaakin oman veneen omistamisesta pääasiassa kustannusten ja huoltovastuiden osalta. Vastuu veneiden huolloista, vakuutuksista ja saatavuudesta on palvelua tarjoavalla yrityksellä, ja asiakas voi keskittyä itse veneilystä nauttimiseen ilman huolia ylläpidosta tai varaosista.

Tämän opinnäytetyön kirjoitushetkellä Skipperillä on yhteensä noin 230 Yamarin-venettä Pohjoismaissa sekä hieman yli 300 venettä maailmanlaajuisesti. Kaikissa Skipperin veneissä on Yamahan perämoottori, jonka koko vaihtelee veneen koon mukaan. Lisäksi jokaisessa veneessä on Nextfourin kehittämä Q Display 2 -kart-taplotteri sekä Igloohomen Keybox v3 -lukkoboksi.

2.3 Skipperi App

Skipperi App on React Nativella sekä Expolla toteutettu mobiilisovellus, joka on suunnattu Skipperi Fleetin jäsenille. Sovellus on ladattavissa iOS-käyttäjille App Storesta ja Android-käyttäjille Google Play -kaupasta.

Sovelluksen kautta käyttäjät voivat varata yhteiskäyttöveneitä, hallita omia varauksiaan, hyödyntää Kohteet-ominaisuutta sekä ilmoittautua tapahtumiin, kuten koulutuksiin ja ohjatuille retkille.

2.4 Skipperi Console

Skipperi Console on yrityksen henkilökunnalle ja yhteistyökumppaneille kehitetty web-pohjainen hallintatyökalu. Se kokoaa yhteen kaikki keskeiset toiminnot, joita tarvitaan Fleet-veneiden, satamien ja operatiivisen työn hallintaan.

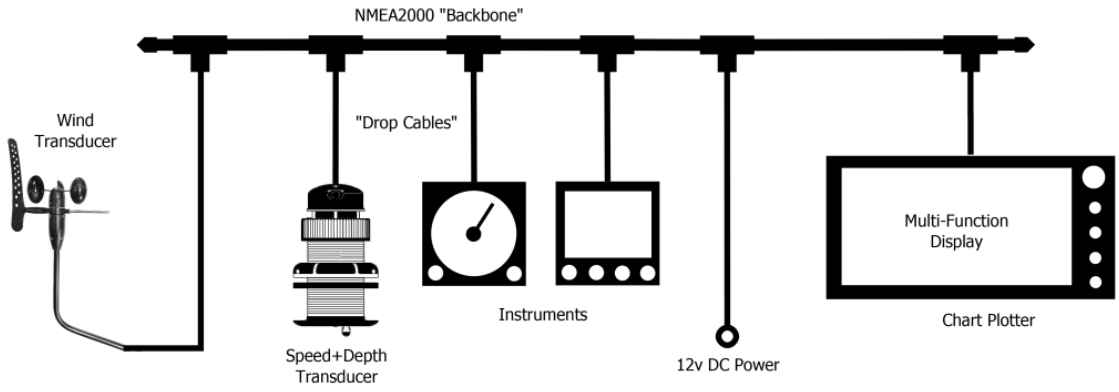
Skipperi Consolen avulla voi muun muassa hallita veneitä ja satamia, varauksia, asiakastietoja, lukkobokseja, Geofence-alueita sekä käyttäjäraportteja. Työkalu on suunniteltu korvaamaan erilliset järjestelmät tarjoamalla keskitetyn ja tehokkaan tavan operoida palvelua.

3 Tutkimus ja teknologiat

Tässä osiossa käsitellään opinnäytetyössä käytettyjä työkaluja ja teknologioita. Yksi opinnäytetyön keskeisimmistä tavoitteista on ollut tutustua näihin teknologioihin syvällisemmin, ymmärtää niiden tarjoamat mahdollisuudet sekä tunnistaa niihin liittyvät rajoitteet. Teknologioiden tutkiminen on ollut olennainen osa kehitysprosessia, ja sen kautta on pyritty luomaan kokonaiskuva siitä, kuinka valittuja ratkaisuja voisi hyödyntää parhaiten opinnäytetyössä sekä saada niistä kaiken irti.

3.1 NMEA 2000

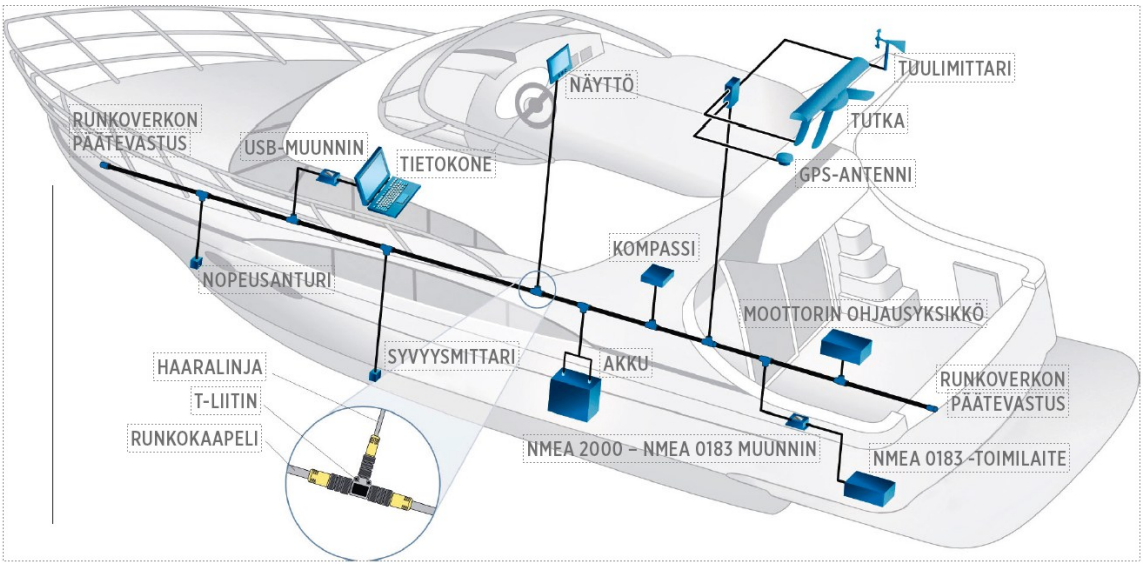
NMEA 2000 on nykyaikainen, standardoitu plug-and-play-tiedonsiirtoprotokolla, joka on mullistanut vene-elektroniikan verkottamisen. Se on suunniteltu mahdollistamaan useiden erilaisten laitteiden, esimerkiksi tutkien, syvyysmittareiden, nopeusmittareiden ja navigointilaitteiden saumattoman kommunikaation yhteisessä verkossa. Käytännössä NMEA 2000 yhdistää veneen eri elektroniset järjestelmät siten, että kaikki laitteet voivat jakaa dataa toisilleen reaaliaikaisesti. Kuvassa 1 esitellään yksinkertainen esimerkkikaavio NMEA 2000 -verkosta ja siihen liitetyistä laitteista.



Kuva 1. NMEA 2000-verkon esimerkkikaavio [5]

NMEA 2000 perustuu CAN-väyläteknologiaan (Controller Area Network), joka on laajalti käytössä muun muassa autojen tiedonsiirrossa. Autoissa CAN-väylä yhdistää esimerkiksi moottorinohjauksen, jarrujärjestelmän, ilmastoinnin ja mittariston, jolloin eri järjestelmät voivat jakaa tietoa keskenään ilman erillisiä kaapelointeja. Veneissä vastaava periaate mahdollistaa esimerkiksi moottorin, polttoaineanturin, autopilotin ja karttaplotterin tiedonsiirron samassa verkossa. NMEA 2000 on suunniteltu plug-and-play-periaatteella, joka tarkoittaa sitä, että uusi laite voidaan lisätä verkkoon ilman monimutkaisia asennusprosesseja. Käytännössä tämä tarkoittaa sitä, että uuden laitteen liittäminen verkkoon onnistuu yksinkertaisesti kytkemällä se yhdellä drop-kaapelilla suoraan runkoverkkoon – ilman muita asennusvaatimuksia. Jokaisella laitteella on oma yksilöllinen tunnistenumeronsa, millä varmistetaan, että lähetetty data voidaan erottaa ja yhdistää oikeaan laitteeseen.

NMEA 2000 -verkko rakentuu yhdestä pääkaapelista eli runkolinjasta, joka toimii järjestelmän selkärankana. Runkolinjaan voidaan lisätä jopa 50 laitetta drop-kaapeleiden avulla. Drop-kaapeliksi kutsutaan sitä kaapelia, millä yhdistetään laiteen NMEA-portti NMEA-runkolinjaan. Kuvassa 2 esitellään esimerkki huviveneeseen rakennetusta NMEA 2000 -verkosta, johon on liitetty erilaisia veneilyyn liittyviä lisälaitteita. [5; 6.]



Kuva 2. NMEA 2000 -verkko huviveneessä [7]

Juuri tämän suunnittelun takia NMEA 2000 -verkon suurin etu on sen helppokäyttöisyys ja skaalautuvuus. NMEA 2000 -verkko vaatii toimiakseen 12V tasavirran sekä runkolinjan molempiin päihin päätevastukset.

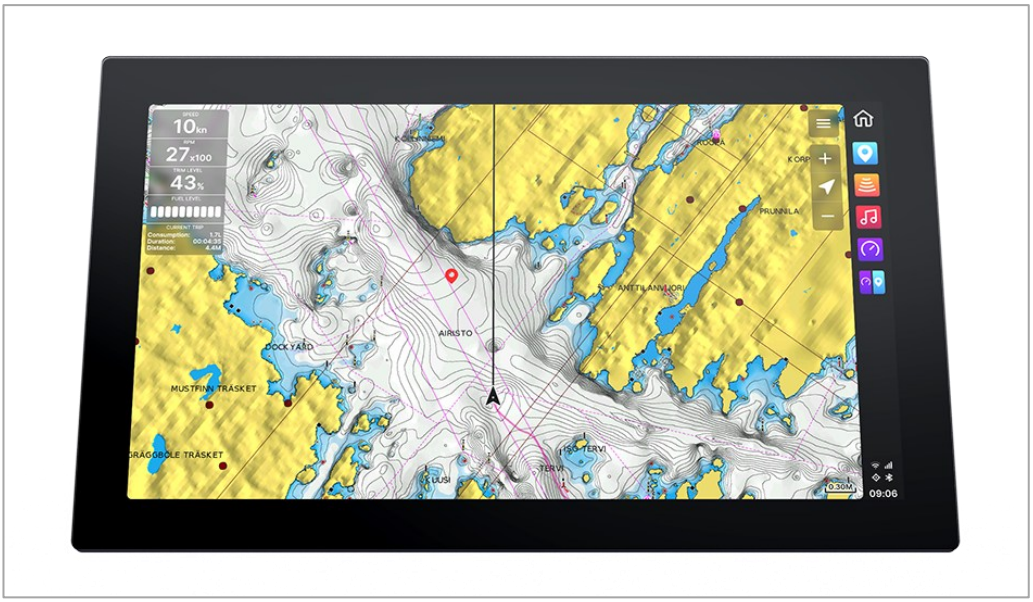
3.2 Yamaha Command Link

Yamaha Command Link on Yamahan oma, sulautettu kommunikaatiojärjestelmä, joka on suunniteltu yhteensopivaksi Yamahan perämoottoreiden kanssa. Tämä järjestelmä mahdollistaa moottorin eri toimintaparametrien kuten kierrosnopeuden, lämpötilan sekä polttoaineenkulutuksen reaaliaikaisen seurannan. Yamaha Command Link on omalla protokollallaan toimiva itsenäinen järjestelmä, joka huolehtii moottorin tiedonkeruusta ja -käsittelystä.

Yamaha Command Link -järjestelmä on liitettävissä NMEA 2000 -verkkoon erillisen yhdyskäytävämoduulin avulla. Tämä yhdyskäytävä toimii käännöspisteenä, joka muuntaa Yamahan oman tiedonsiirtoprotokollan NMEA 2000 -standardin ymmärtämään muotoon. Näin moottorin tiedot, kuten kierrosluku, lämpötila ja polttoainetiedot, voidaan siirtää NMEA 2000 -verkkoon ja jakaa veneen muiden laitteiden, kuten karttaplotterin kanssa. Järjestelmä toimii kuitenkin itsenäisesti, eli sitä ei tarvitse kytkeä osaksi NMEA 2000 -verkkoa toimiakseen. [8; 9.]

3.3 Q Display 2 Series

Q Display 2 Series on turkulaisen Nextfour Oy:n kehittämä älykäs karttaplotteri. Se toimii ikään kuin veneen ohjauskeskuksena, mihin voi liittää erilaisia lisälaitteita hallitakseen niitä. Se kokoaa yhteen useita toimintoja, kuten karttanäytön, moottori- ja anturidatan, syvyysmittauksen sekä viihdeominaisuuksia. Laitetta on saatavilla kuudessa eri koossa: 10”, 12”, 16”, 22”, 24” sekä tupla 10”. Q-näyttö saa tietonsa muun muassa sisäänrakennetusta GPS-anturista, ja se voi vastaanottaa dataa myös NMEA 2000 -verkosta, joka yhdistää veneen eri elektroniset laitteet kuten perämoottorin. Näytön voi yhdistää internetiin joko SIM-kortin avulla tai WiFi-verkon kautta. [10.] Kuvassa 3 esitetään havainnekuva Q-näytöstä.



Kuva 3. Q Display 2 Series [11]

Q-näyttö on yhteydessä veneen perämoottoriin NMEA 2000 -verkon kautta ja välittää tietoa moottorin tilasta, kuten polttoaineenkulutuksesta ja kierrosluvusta. Lisäksi Q-näyttö tallentaa veneiden moottoreihin tehtävät rutiinihuollot, näyttää syvyystiedot sekä polttoainetankin tason. [12.]

Q-näyttö päivittää tietojaan verkkoon jatkuvasti laitteen ollessa päällä ja 10 minuutin välein laitteen ollessa sammutettuna, mutta kytkettynä herätevirtaan. Virallisten Nextfourin asennusohjeiden mukaan Q-näyttö on päällä, kun veneen virta-avaimesta on käännetty virrat päälle. Herätevirrassa karttaplotteri on taas silloin, kun virta-avain on käännetty pois päältä. Voitaisiin siis sanoa, että Q-näyttö on päällä, kun veneen moottori on päällä, ja herätevirrassa, kun veneen moottori on sammutettuna.

Kuva 4. Q-näyttö käytössä veneessä [11]

Kohdeyrityksessä Q-näyttö oli asennettuna jokaiseen veneeseen, ja sen pääasiallinen tehtävä oli toimia navigointilaitteena sekä välittää tietoa veneen sijainnista sekä polttoaineenkulutuksesta verkkoon. Lisäksi se toimi Geofencing-järjestelmän näyttönä, jolloin paikkatietoon perustuvat hälytykset näkyivät suoraan näytöllä ja antoivat tarvittaessa myös äänimerkin. Kuvassa 4 havainnollistetaan Q-näyttöä käytössä veneessä.

3.4 Igloohome Keybox v3

Igloohome Keybox v3 on igloocompany Pte:n kehittämä älykäs avainlaatikko, joka on suunniteltu helpottamaan avainten säilyttämistä ja pääsynhallintaa erilaisissa ympäristöissä. Yksi sen yleisimmistä käyttötarkoituksista on Airbnb-asuntovuokraus, jossa tarvitaan kätevä ja turvallinen tapa hallinnoida asunnon avainten luovutusta ilman fyysistä avaintenvaihtoa vuokraajien välillä. [13.] Kuvassa 5 esitetään havainnekuva Keybox v3 -lukkoboksista.

Kuva 5. Igloohome Keybox v3 -lukkoboksi [14]

Kohdeyrityksessä Igloohomen kehittämiä lukkobokseja käytetään veneiden perämoottorien virta-avainten säilytykseen. Lukkoboksit kiinnitettiin veneeseen näkyvälle paikalle, josta ne olivat helposti havaittavissa ja käytettävissä. Kuvassa 6 havainnollistetaan lukkoboksin käyttöä kohdeyrityksen veneessä.

Kuva 6. Lukkoboksi käytössä kohdeyrityksen veneessä

Lukkoboksi kykenee generoimaan PIN-koodeja ilman jatkuvaa verkkoyhteyttä tai Bluetooth-yhteyttä käyttämällä Igloohomen kehittämää algoPIN-teknologiaa.

algoPIN-teknologia toimii siten, että lukkoboksin parituksen yhteydessä laite jakaa salaisen siemenluvun Igloohomen pilvipalvelimen kanssa. Kun käyttäjä haluaa luoda uuden PIN-koodin Igloohomen omalla mobiilisovelluksella, sovellus lähettää pyynnön palvelimelle, joka hyödyntää aiemmin synkronoituja siemenlukuja sekä aikaleimaa tuottaakseen satunnaisen, määräaikaisesti voimassa olevan PIN-koodin. Tämä teknologia perustuu samankaltaiseen menetelmään, kuin pankkien käyttämät kertakäyttöiset salasanat. Koska lukko toimii offline-tilassa, se on suojattu haavoittuvuuksilta ja yhteysongelmilta. Kohdeyrityksessä lukkoboksin aikaleima päivittyi aina lukon Bluetooth-avauksen yhteydessä. Näin PIN-koodeja voitiin siis luoda täysin etänä ilman minkäänlaista yhteyttä lukkoboksiin ja hallita asiakkaiden pääsyä lukkoboksiin varauskohtaisesti. [15.]

Igloohomen kehittämä Keybox v3 tukee myös Bluetooth-avauksia, jolloin lukon voi avata Bluetooth yhteyden avulla ilman pitkien PIN-koodien syöttämistä. Kohdeyrityksessä asiakkaat voivat avata lukkobokseja sekä syöttämällä generoidun PIN-koodin että Bluetooth-yhteyden avulla.

Lisäksi PIN-koodit voidaan peittää, jolloin käyttäjä voi syöttää jopa 16 numeroa, joista vain viimeiset numerot muodostavat varsinaisen PIN-koodin. Tämä lisää turvallisuutta ja suojaa PIN-koodia muiden katseilta. [16.]

4 Käytettävät työkalut ja ohjelmointikielet

Tässä osiossa esitellään tarkemmin ne työkalut ja ohjelmointikielet, joita opinnäytetyön toteutuksessa hyödynnettiin. Valinnat tehtiin pitkälti sen pohjalta, mitä teknologioita yrityksellä oli jo valmiiksi käytössä. Samalla oli tärkeää miettiä, kuinka nämä työkalut sopivat juuri tämän tyyppisen käyttöönottotyökalun kehittämiseen ja mitä hyötyjä tai haasteita niiden käyttö voisi tuoda mukanaan.

Taulukko 1. Kuvaus projektissa käytetyistä työkaluista ja teknologioista

Komponentti	**Teknologia / Työkalu**
Käyttöliittymä	React, Typescript
Välimuisti	Redis
Palvelinpuoli	PHP – Laravel (Framework) – Eloquent (ORM)
Mobiilisovellus	React Native, Expo, Typescript
Testaus	PHPUnit
Muut työkalut	Q Display 2, Igloohome Keybox v3

Taulukossa 1 esitellään tässä opinnäytetyössä käytetyt työkalut ja teknologiat sekä niiden käyttötarkoitus.

4.1 TypeScript

TypeScript on Microsoftin kehittämä ja ylläpitämä ohjelmointikieli, joka laajentaa suosittua JavaScriptiä lisäämällä siihen staattisen tyypityksen. Se on suunniteltu pääasiassa suurten sovellusten kehittämiseen. Se mahdollistaa muuttujien ja funktioiden tietotyyppien määrittämisen jo koodin kirjoitusvaiheessa, mikä tekee koodista ennakoitavampaa ja helpottaa virheiden etsintää.

JavaScript on erittäin suosittu ohjelmointikieli, jota käytetään laajasti nykyaikaisessa web-kehityksessä. Sen suurin heikkous on kuitenkin se, että muuttujien tyypit määritellään vasta ajon aikana, mikä voi johtaa odottamattomiin virheisiin. Esimerkiksi, jos muuttujaan tallennetaan arvo, joka ei ole odotettua tyyppiä, virhe ilmenee vasta silloin, kun koodia suoritetaan. TypeScript ratkaisee tämän ongelman tarjoamalla mahdollisuuden määritellä muuttujien ja funktioiden tyypit jo koodin kirjoitusvaiheessa, mikä johtaa siihen, että virheet voidaan havaita ja korjata jo ennen koodin ajamista. TypeScript onkin erittäin suosittu modernissa web-kehityksessä, sillä se voidaan kääntää tavalliseksi JavaScriptiksi, joka toimii kaikilla moderneilla verkkoselaimilla. [17.]

4.2 React, React Native ja Expo

React on Facebookin kehittämä avoimen lähdekoodin JavaScript-kirjasto, jolla luodaan interaktiivisia ja dynaamisia käyttöliittymiä verkkosivuihin. Sitä käytetään erityisesti yksisivuisten verkkosovellusten (Single Page Application, SPA) rakentamiseen. React perustuu komponenttipohjaiseen arkkitehtuuriin, jossa sovellus jaetaan pieniin, uudelleenkäytettäviin osiin. Näin käyttäjät voivat rakentaa sovelluksia yhdistämällä itsenäisiä komponentteja, kuten nappeja, lomakkeita tai ikkunoita, ilman, että koodia tarvitsee toistaa. Yksi Reactin keskeisistä vahvuuksista on sen virtuaalinen DOM (Document, Object Model), joka pitää sisällään käyttöliittymän virtuaalisen kopion. Kun käyttöliittymässä tapahtuu muutoksia, React päivittää vain muuttuneet osat, mikä parantaa suorituskykyä ja tekee sovelluksesta nopeamman. Täten React onkin yksi suosituimpia tapoja kehittää moderneja verkkosivustoja. [18.]

React Native on Reactiin pohjautuva avoimen lähdekoodin ohjelmistokehys, joka mahdollistaa mobiilisovellusten kehittämisen sekä Androidille että iOS:lle yhdellä JavaScript-koodipohjalla. Tämä poistaa tarpeen kehittää erillisiä sovelluksia kummallekin alustalle, nopeuttaen kehitystä ja vähentäen ylläpitotyötä. Toisin kuin perinteisessä Reactissa, jossa käyttöliittymä rakennetaan HTML:n ja CSS:n avulla, React Nativessa hyödynnetään natiivikomponentteja, jotka vastaavat suoraan kunkin käyttöjärjestelmän omia käyttöliittymäelementtejä. Natiivikomponentit tarkoittavat käyttöjärjestelmään sisäänrakennettuja elementtejä, kuten painikkeita, listoja ja näkymiä, jotka käyttäytyvät ja näyttävät samalta kuin käsin koodatut Android- tai iOS-komponentit. [19.]

Expo on osa React Native -ekosysteemiä, joka helpottaa kehitystä tarjoamalla työkaluja sovelluksen nopeaan testaamiseen ja käyttöönottoon ilman erillistä natiivia kehitysympäristöä. Expon avulla kehittäjät voivat keskittyä sovelluksen toiminnallisuuteen ilman, että heidän tarvitse huolehtia alustan yksityiskohdista. [20.]

4.3 PHP ja Laravel

PHP (Hypertext Preprocessor) on laajasti käytetty avoimen lähdekoodin palvelinpuolen ohjelmointikieli, jota käytetään erityisesti web-palvelinympäristössä. PHP on tulkattu kieli, mikä tarkoittaa, että ohjelmakoodi suoritetaan vasta ohjelman ajon aikana, eikä sitä tarvitse kääntää erikseen ennen käyttöönottoa. PHP:tä käytetään yleisesti dynaamisten verkkosivustojen ja sovellusten luomiseen, ja sen avulla voidaan käsitellä esimerkiksi lomaketietoja, tietokantahakuja ja käyttäjän istuntoja. PHP on kehittynyt vuosien varrella merkittävästi pelkästä skriptikielestä täysiveriseksi olio-ohjelmointia tukevaksi kieleksi, jossa on parannettu tyyppijärjestelmää ja suorituskykyä. [21.]

Yksi tunnetuimmista sekä suosituimmista PHP-ohjelmistokehyksistä on Laravel. Se perustuu MVC-arkkitehtuuriin (Model-View-Controller), joka auttaa jäsentämään koodia selkeämmin ja parantamaan sen ylläpidettävyyttä. Laravel tarjoaa valmiita ratkaisuja esimerkiksi reititykseen, tietokantamallinnukseen (Eloquent ORM) ja käyttäjäautentikointiin. Laravelin tavoitteena on helpottaa ja nopeuttaa PHP-kehitystä tarjoamalla selkeitä ja uudelleenkäytettäviä työkaluja ja se on suunniteltu erityisesti web-kehitykseen. [22.]

4.4 Tekoälyn käyttö opinnäytetyössä

Tässä opinnäytetyössä hyödynnettiin tekoälyä rajatusti tekstin suunnittelun ja tekstin tuottamisen tukena. Käytetty työkalu oli OpenAI:n kehittämä ChatGPT-4o [23], joka toimi apuvälineenä kirjallisessa työssä. Tekoälyä ei käytetty varsinaisen ohjelmakoodin tai käyttöönottotyökalun kehityksessä, eikä sillä tuotettu lainkaan koodia tai teknisiä toteutuksia. Kaikki tekninen suunnittelu, ohjelmointi sekä testaus toteutettiin itsenäisesti ilman tekoälyn apua.

5 Projektin suunnittelu ja toteutus

Tässä osiossa esitellään opinnäytetyön varsinainen kehitysprosessi alusta loppuun – ideasta ensimmäisiin suunnitelmiin ja tutkimuksesta toteutukseen. Projektin edetessä korostui erityisesti vaiheittainen eteneminen, jossa käyttäjäpalautteen ja teknisen testaamisen avulla ohjattiin kehitystä oikeaan suuntaan. Suunnittelussa painotettiin selkeyttä, iteratiivisuutta sekä valmiutta tehdä muutoksia matkan varrella, mikä on olennaista tällaisessa käytännönläheisessä kehitysprojektissa. Osiossa kuvataan konkreettisesti, millaisia ratkaisuja toteutuksessa tehtiin, miksi tiettyjä reittejä valittiin ja mitä niistä opittiin.

5.1 Suunnittelu- ja tutkimusvaihe

Projektin ensimmäinen vaihe oli tutkimusvaihe, missä tutkittiin saatavilla olevaa dataa sekä tehtiin dokumentaatiota siitä, millaisia johtopäätöksiä saatavilla olevan datan pohjalta voitaisiin tehdä. Tavoitteena oli muodostaa mahdollisimman kattava kokonaiskuva siitä, millaisia käyttöönottovaiheita järjestelmään voitaisiin sisällyttää ja mitä asioita kussakin vaiheessa olisi järkevää tarkastaa. Analyysin avulla pyrittiin ymmärtämään, miten käyttöönottoprosessi voisi käytännössä rakentua. Tutkimuksen tuloksena määritimme käyttöönotolle neljä selkeää vaihetta ja niissä tarkastettavat asiat. Nämä neljä käyttöönottovaihetta olivat: veneen perustiedot, Q-näytön asennus, Lukkoboksin asennus sekä veneen julkaisutiedot. Jokainen käyttöönottovaihe sisälsi useita käyttöönottotarkastuksia, joiden avulla voitiin varmistaa, että kyseisen vaiheen edellyttämät asiat oli hoidettu asianmukaisesti ennen seuraavaan vaiheeseen siirtymistä. Esimerkiksi veneen perustietojen käyttöönottovaiheessa yksi tarkistettava asia oli se, että veneelle oli määritetty asianmukainen nimi.

Projektin toinen vaihe oli suunnitteluvaihe, jonka keskeisenä lähtökohtana toimi vaatimusmäärittely. Vaatimusmäärittelyssä kartoitettiin projektin tavoitteet, käytettävät teknologiat sekä toteutuksen laajuus. Lisäksi määriteltiin keskeiset vaatimukset sekä asetettiin selkeä kuva halutusta lopputuloksesta sekä käyttötarkoituksesta. Suunnitteluvaiheen tarkoituksena oli varmistaa, että projekti etenee johdonmukaisesti ja että toteutusvaiheessa vältetään tarpeettomia muutoksia tai epäselvyyksiä. Suunnittelua ohjasi erityisesti käyttäjälähtöinen ajattelumalli: kuka työkalua käyttää, mihin tarkoitukseen ja millaisessa ympäristössä.

Projektin toteutuksessa päätettiin hyödyntää samoja teknologioita ja arkkitehtuuria, joita muu web-sovellus käytti. Tämä ei ainoastaan helpottanut kehitystyötä, vaan oli myös luonnollinen ratkaisu.

Suunnitteluvaiheessa yhdeksi keskeisimmistä vaatimuksista nousi työkalun modulaarisuus ja joustavuus. Tavoitteena oli rakentaa järjestelmä, joka mahdollistaisi tulevaisuudessa yksittäisten veneiden omien käyttöönottovaiheiden määrittelyn sekä uusien vaiheiden lisäämisen tai olemassa olevien poistamisen. Vaikka tätä veneiden yksilöllistä konfigurointia ei tässä vaiheessa vielä toteutettu, koko käyttöönottotyökalu suunniteltiin siten, että kyseinen ominaisuus olisi vaivattomasti lisättävissä tulevaisuudessa. Lisäksi työkalun tuli tarjota käyttäjälleen selkeitä sekä hyödyllisiä palautteita käyttöönoton eri vaiheista sekä niiden onnistumisesta. Mikäli esimerkiksi osa käyttöönotosta oli suoritettu onnistuneesti ja osa ei, ohjelman piti palauttaa, mikä osa ei ole onnistunut ja miten se tulisi korjata. Tässä opinnäytetyössä näitä palautteita kutsutaan ohjelappusiksi.

Suunnitteluvaiheen aikana laadittiin myös alustavat kaaviot sekä kuvaukset järjestelmän toiminnasta, jotta projektin rakenne ja keskeiset toiminnallisuudet hahmottuvat selkeästi jo varhaisessa vaiheessa ennen toteutusvaiheen alkamista. Lisäksi palvelinpuolen ohjelmoinnista laadittiin palvelinpuolen arkkitehtuurikaavio, joka kuvaa palvelinpuolen ohjelmakoodin rakennetta sekä eri osien vuorovaikutusta toisiinsa. Erityistä huomiota kiinnitettiin palvelinpuolen skaalautuvuuteen sekä suorituskykyyn.

Onnistuneen projektin kulmakivi on huolellisesti laadittu aikataulu. Tässä projektissa luotiin kymmenen viikon suunnitelma, jossa kuvattiin projektin eri vaiheet, aikataulut sekä tavoitteet. Projektin keston suunnittelussa otettiin huomioon kehitystyön lisäksi opinnäytetyön kirjoittamisen vaikutus projektin kestoon. Projektin kulku suunniteltiin iteratiiviseksi, mikä tarkoitti sitä, että kehitysvaiheita ei tehty täysin peräkkäin, vaan suunnittelun, toteutuksen ja arvioinnin välillä oli useita syklejä. Tämä mahdollisti jatkuvan parantamisen ja tarpeen mukaan tehtävät muutokset projektin edetessä.

Projektin aikataulu jakautui seuraavasti:

Viikot 1–2: Tutkimusvaihe

Projekti aloitettiin tausta- sekä tutkimustyöllä. Käytettäviä työkaluja ja teknologioita kartoitettiin ja niihin tutustuttiin, sekä eri ratkaisuvaihtoehtoja tutkittiin huolellisesti. Keskeiseksi tutkimustyöksi muodostui saatavilla olevan datan tutkiminen sekä se, millaisia johtopäätöksiä sen pohjalta pystyttiin tekemään. Se ohjasi lopulta merkittävästi käyttöönottotyökalun vaatimusmäärittelyä.

Viikot 2–3: Suunnitteluvaihe 1

Kun tarvittava tutkimustyö oli tehty, siirryttiin järjestelmän suunnitteluun. Tässä vaiheessa määriteltiin projektin kokonaisarkkitehtuuri, valittiin käytettävät teknologiat sekä laadittiin alustavat suunnitelmat käyttöliittymästä. Samalla luotiin vaatimusmäärittelyt sekä käytiin läpi, mitä käyttöönottovaiheita järjestelmään tarvitaan ja millaiset kriteerit kunkin vaiheen onnistuneelle suorittamiselle asetetaan. Jokaiselle vaiheelle laadittiin selkeät reunaehdot siitä, mitä tietoja tai toimintoja tulee olla olemassa tai suoritettuna, jotta vaihe voidaan merkitä suoritetuksi.

Viikot 3–4: Ohjelmointi 1

Ensimmäinen ohjelmointivaihe keskittyi palvelinpuolen toteutukseen ja järjestelmän perustoiminnallisuuksien rakentamiseen. Kehitystyössä hyödynnettiin testivetoista kehitystä (TDD). Lisäksi käyttöliittymästä laadittiin ensimmäinen versio, jonka tarkoituksena oli havainnollistaa käyttöliittymän pääkomponentteja sekä toiminnallisuutta.

Viikko 5: Suunnitteluvaihe 2

Ensimmäisen ohjelmointivaiheen jälkeen pysähdyttiin tarkastelemaan toteutettua kokonaisuutta ja arvioimaan sen toimivuutta suhteessa alkuperäisiin suunnitelmiin. Tässä vaiheessa keskityttiin erityisesti niihin haasteisiin, jotka olivat nousseet esiin ensimmäisen ohjelmointiviikon aikana. Tarvittavat muutokset dokumentoitiin ja suunnitelmia päivitettiin sen mukaisesti, jotta kehitystyö voitiin viedä hallitusti eteenpäin seuraavaan vaiheeseen.

Viikko 6: Ohjelmointi 2

Toisen suunnitteluvaiheen jälkeen kehitystyö jatkui laajempien toiminnallisuuksien ja käyttöliittymän jatkototeutuksen parissa. Viimeisen ohjelmointivaiheen tavoitteena oli viimeistellä projektin perustoiminnallisuudet ja saattaa ohjelmointi lähes valmiiksi niin palvelinpuolella kuin käyttöliittymässäkin. Tässä vaiheessa toiminnallisuuksia laajennettiin kattamaan monipuolisempia käyttötapauksia ja käyttöliittymää kehitettiin entistä selkeämmäksi ja informatiivisemmaksi. Lisäksi ohjeita kirjoitettiin.

Viikot 7–8: Arviointi ja testaus

Tässä vaiheessa suoritettiin perusteellinen testaus, jossa käytiin läpi sekä yksittäisten komponenttien toimivuus että koko järjestelmän suorituskyky ja käytettävyys. Testauksen tavoitteena oli varmistaa, että kaikki osat toimivat suunnitellusti ja saumattomasti yhdessä, sekä että järjestelmä täyttää vaaditut laatu- ja käytettävyyskriteerit. Testaus painottui palvelinpuolen toteutukseen. Testauksesta kerrotaan tarkemmin luvussa 6.

Viikko 9: Julkaisuvaihe

Kaikki lopulliset muutokset ja korjaukset tehtiin ja projekti valmisteltiin julkaisua varten. Tässä vaiheessa kehitystyö painottui testauksessa havaittuihin epäkohtiin sekä optimointitarpeisiin. Projektin julkaisusta sekä viemisestä tuotantoympäristöön kerrotaan tarkemmin luvussa 5.5.

Viikko 10: Deadline sekä käyttäjätestaus

Viimeinen viikko oli varattu projektin viimeistelyyn sekä varmistamiseen, että kaikki asetetut tavoitteet oli saavutettu. Lisäksi käyttäjätestaus suoritettiin sen jälkeen, kun projekti oli saatu vietyä onnistuneesti tuotantoympäristöön. Käyttäjätestauksesta kerrotaan tarkemmin luvussa 6.3.

5.2 Käyttöliittymän suunnittelu

Käyttöliittymän suunnittelusta vastasi käyttöliittymäsuunnittelija, joka hyödynsi Figmaa suunnittelutyökaluna. Figman avulla luotiin aluksi wireframe-malli, joka toimi suunnittelun pohjana. Tämä malli kuvasi sovelluksen rakenteen, eri näkymien järjestyksen sekä keskeiset käyttöliittymäelementit. Wireframe-malli, eli käyttöliittymän luonnos, on yksinkertainen hahmotelma sovelluksen näkymistä ilman värejä tai tyylejä. Wireframe-mallin avulla voitiin keskittyä käyttöliittymän toiminnallisuuteen ja käyttäjän kulkemiin reitteihin sovelluksessa ilman, että ulkoasun yksityiskohdat veivät huomiota.

Perusrakenteen jälkeen suunnittelija loi korkearesoluutioiset visuaaliset mallit, jossa määriteltiin käyttöliittymän visuaaliset elementit kuten värit, typografia, ikonit ja muut graafiset yksityiskohdat. Suunnitteluprosessissa hyödynnettiin komponenttipohjaista lähestymistapaa, jossa käyttöliittymäelementit kuten napit, lomakkeet ja valikot suunniteltiin uudelleenkäytettäviksi komponenteiksi sekä suunnittelussa käytettiin jo muualla käytettyjä komponentteja. Lopullinen käyttöliittymä rakennettiin vastaamaan näitä visuaalisia malleja.

5.3 Toteutusvaihe

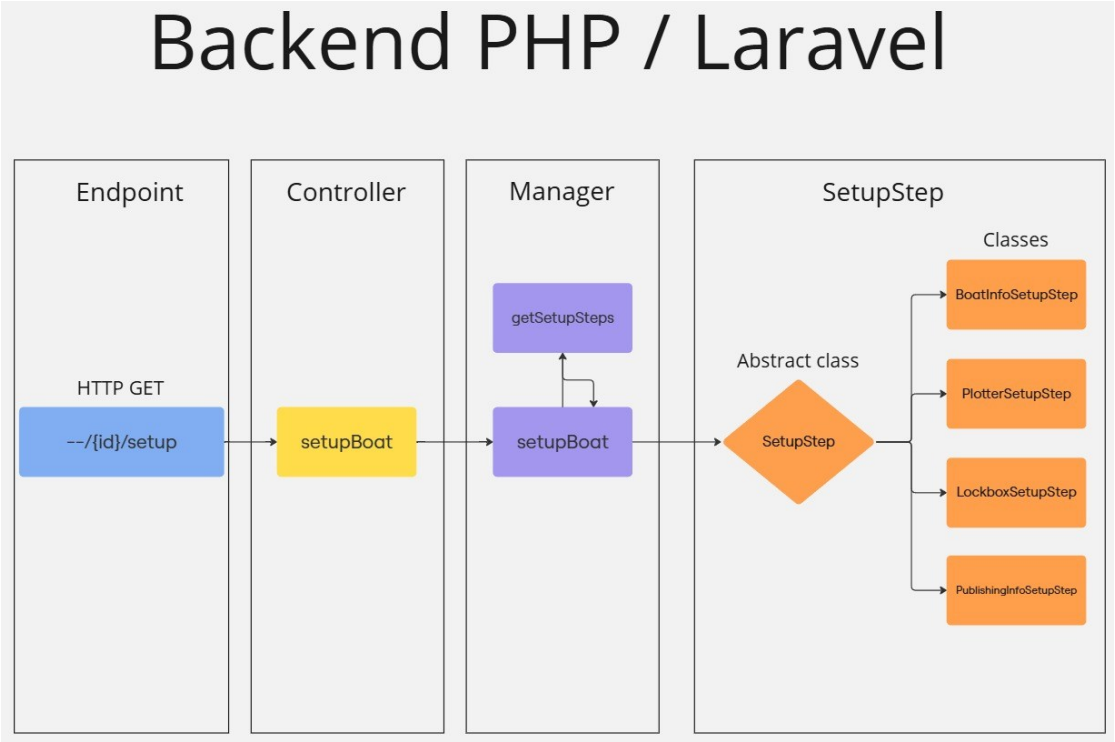
Projektin toteutusvaihe alkoi palvelinpuolen koodauksella. Palvelinpuolen koodauksessa noudatettiin testivetoista kehitystä (Test Driven Development, TDD), jossa testit kirjoitettiin osittain jo ennen varsinaista ohjelmakoodia. Tämä oli mahdollista, sillä suunnittelu- ja tutkimusvaihe oli perusteellisesti tehty. Tämä lähestymistapa varmisti sen, että jokainen uusi toiminnallisuus täytti sille asetetut vaatimukset heti alusta alkaen ja että ohjelmakoodi pysyi mahdollisimman virheettömänä läpi kehitysprosessin. Lisäksi TDD auttoi ylläpidettävyyden ja modulaarisuuden varmistamisessa, sillä testit ohjasivat koodin kehitystä.

Ohjelmistorajapinta toteutettiin REST-arkkitehtuurin mukaisesti seuraten muun sovelluksen käytäntöjä. REST-periaate keskittyy resurssien käsittelyyn, jossa jokainen resurssi, kuten tuote tai käyttäjä, esitetään yksilöllisellä URL-osoitteella. Tässä tapauksessa luotiin yksittäinen REST-rajapinta, joka suorittaa käyttöönottovaiheet tietylle veneelle. Rajapinta on määritelty polkuun --/{id}/setup jossa {id} viittaa kyseisen veneen yksilölliseen id-numeroon, joka on käyttöönotettavana. Tätä tunnistetta käyttäen järjestelmä tietää, mille veneelle käyttöönotto suoritetaan.

Palvelinpuolen toteutuksessa käytettiin controller/service-arkkitehtuuria, jossa koodi on jaettu kolmeen eri pääkomponenttiin: controller, service ja käyttöönottovaiheet. Tämä rakenne valittiin, koska se mahdollistaa selkeän erottelun sovelluksen eri tasoilla ja tekee koodista modulaarisen ja ylläpidettävän.

Controllerin tehtävänä on vastaanottaa ulkoiset HTTP-pyynnöt ja ohjata ne eteenpäin oikeaan käsittelykerrokseen. Tässä toteutuksessa controller ei sisällä laisinkaan liiketoimintalogiikkaa, vaan se delegoi tehtävät service-kerrokseen. Kun controller vastaanottaa HTTP-pyynnön, se validoi pyynnön ja delegoi sen service-kerrokselle, joka suorittaa tarvittavat toimenpiteet. Tällainen rakenne pitää controllerin yksinkertaisena ja nopeana.

Service-kerros käsittelee varsinaisen liiketoimintalogiikan. Se toimii sovelluksen ytimenä, jossa toteutetaan keskeiset toiminnot kuten tietojen hakeminen, laskentatehtävät ja tiedonsiirto muihin kerroksiin. Service-kerros on tärkeä, sillä sen avulla saadaan eriytettyä liiketoimintalogiikka muista järjestelmän osista, kuten tietokantakerroksesta tai äsken kerrotusta controller-kerroksesta. Tässä toteutuksessa service-kerros vastaa käyttöönottovaiheiden käsittelystä ja ajamisesta oikeassa järjestyksessä. Tässä projektissa service-kerrosta kutsuttiin nimellä Manager. Kuvassa 7 esitetään palvelinpuolen toteutuksen vuokaavio, joka havainnollistaa palvelinpuolen ohjelmakoodin rakenteen ja sen eri osien vuorovaikutuksen.



Kuva 7. Palvelinpuolen toteutuksen vuokaavio

Käyttöönottovaiheiden toteutuksessa hyödynnettiin abstraktia luokkaa, joka määrittelee yleisen rakenteen kaikille käyttöönottovaiheille. Jokainen yksittäinen käyttöönottovaihe on oma luokkansa, joka perii tämän abstraktin luokan ja toteuttaa sen määrittelemät metodit ja toiminnot. Tämä lähestymistapa takasi sen, että kaikki itsenäiset käyttöönottovaiheet noudattavat samaa rakennetta, mikä parantaa koodin johdonmukaisuutta ja selkeyttä. Abstrakti luokka toimii eräänlaisena "sopimuksena", joka määrittelee, mitkä metodit ja toiminnot ovat pakollisia kaikille käyttöönottovaiheille. Tällöin uusi vaihe voidaan lisätä helposti järjestelmään, kunhan se seuraa tätä ennalta määriteltyä rakennetta.

Toteutuksen yhtenä keskeisenä vaiheena oli rajapintavastauksen rakenteen suunnittelu ja toteutus. Rajapintavastausten suunnittelussa oli tärkeää miettiä, miten tieto esitetään selkeästi ja loogisesti niin, että käyttöliittymä pystyy käsittelemään sen tehokkaasti. Toisaalta vastauksen pitäisi olla myös ihmiselle intuitiivinen ja selkeä, jotta sitä voi tulkita tietämättä taustalla tehdystä ohjelmoinnista mitään.

Rajapintavastauksen rakenteeksi valikoitui seuraavanlainen malli:

```
{
    "step": "LockboxSetupStep",
    "result": {
        "status": "ok",
        "message": "",
        "steps": [
            {
                "status": "ok",
                "message": "Lockbox found and linked succesfully!",
                "steps": null
            },
            {
                "status": "ok",
                "message": "Battery level ok!",
                "steps": null
            }
        ]
    }
},
```


Kuva 8. Rajapintavastaus lukkoboksin käyttöönotosta

Rajapintavastauksessa Step tarkoittaa käyttöönottovaihetta ja Result käyttöönottovaiheen tulosta. Tulos sisältää tiedon siitä, onko käyttöönottovaihe valmis (status) sekä listauksen julkisista tarkastusvaiheista (steps). Steps on jono tarkastuskohtia, joita on käyty läpi käyttöönottovaiheen aikana, jotka määrittelevät, onko käyttöönottovaihe valmis vai kesken. Lisäksi vastaukseen on mahdollista liittää vapaavalintainen viesti (message). Kuvan 9 rajapintavastauksessa veneen perustiedot ovat kunnossa, ja käyttöönottovaiheen valmius on valmis (ok).

```
{
    "step": "BoatInfoSetupStep",
    "result": {
        "status": "ok",
        "message": "",
        "steps": [
            {
                "status": "ok",
                "message": "Boat name OK!",
                "steps": null
            },
            {
                "status": "ok",
                "message": "Boat model OK!",
                "steps": null
            },
            {
                "status": "ok",
                "message": "Boat location OK!",
                "steps": null
            }
        ]
    }
},
```


Kuva 9. Rajapintavastaus veneen perustietojen käyttöönotosta

Toteutuksessa käyttöönottovaiheelle päätettiin luoda kolme selkeää sekä erillistä tilaa, jotka kuvaavat käyttöönottovaiheen valmiutta. Näitä tiloja ovat error, info ja ok.

Error-tila ilmaisee, että käyttöönottovaihe on kesken tai epäonnistunut jollain kriittisellä tavalla - eli kyseessä on tilanne, jossa asennus ei voi jatkua ennen kuin jokin virhe korjataan. Tähän voivat kuulua esimerkiksi puuttuvat tiedot tai virheelliset asennukset.

Info-tila puolestaan tarkoittaa, että järjestelmä on havainnut jotakin huomionarvoista, mutta se ei ole varsinaisesti este käyttöönoton onnistumiselle. Tällaisia voivat olla esimerkiksi erilaiset varoitukset tai epätyypilliset lukemat, kuten se, että polttoainetankki on tyhjä.

Ok-tila tarkoittaa, että kaikki käyttöönottovaiheen edellytykset täyttyvät, eikä järjestelmä ole havainnut mitään poikkeavaa - toisin sanoen käyttöönottovaihe on valmis.

Kolmiportaisen tilajärjestelmän tavoitteena oli symboloida liikennevaloja, jotka ovat useimmille käyttäjille jo entuudestaan tuttuja ja intuitiivisesti ymmärrettäviä: punainen (error), keltainen (info) ja vihreä (ok). Koodissa tila päätettiin toteuttaa enum-periaatteella, jossa määriteltiin kolme ennalta rajattua ja nimettyä tilaa, joita käyttää muussa ohjelmakoodissa. Kuvassa 10 havainnollistetaan toteutusta.

```
Tuomas Mellin
enum Status: string implements JsonSerializable
{
    case OK = 'ok';
    case INFO = 'info';
    case ERROR = 'error';

    Tuomas Mellin
    public function jsonSerialize(): string
    {
        return $this->value;
    }
}
```

Kuva 10. Status-luokitus

Käyttöönottotyökalun kehityksessä haluttiin varmistaa, että kaikki vaiheet palauttavat tuloksensa yhtenäisellä ja helposti tulkittavalla tavalla. Tätä varten päätettiin luoda yhtenäinen malli, jota voitiin käyttää funktioiden palautusarvona koko sovelluksen laajuudessa. Mallin tarkoituksena oli tuoda selkeyttä niin funktioiden palautuksiin kuin testauslogiikkaankin.

Tämä lähestymistapa helpotti sekä kehitystyötä että testausta, sillä kaikki käyttöönottovaiheet palauttivat tuloksensa samassa muodossa. Testauksessa oli myös helppo hyödyntää yhtenäistä palautusmallia. Kuvassa 11 havainnollistetaan, kuinka palautusrakenne on toteutettu erillisenä mallintavana PHP-luokkana nimeltä StepResult.

```
Tuomas Mellin
class StepResult implements JsonSerializable
{
    /** @param array{StepResult}|null $steps */
    Tuomas Mellin
    public function __construct(
        public readonly Status $status,
        public readonly ?string $message = null,
        public readonly ?array $steps = null
    ) {
    }

    Tuomas Mellin
    public function jsonSerialize(): array
    {
        return [
            'status' => $this->status,
            'message' => $this->message,
            'steps' => $this->steps
        ];
    }
}
```

Kuva 11. Käyttöönottovaiheen palautusrakennetta mallintava PHP-luokka

Yksittäisen käyttöönottovaiheen ohjelmalogiikka rakennettiin siten, että yksi julkinen funktio vastasi koko prosessin koordinoinnista ja suorittamisesta. Tämä pääfunktio toimi kuin kapellimestari, joka ohjaa orkesteria – se kutsui luokan yksityisiä funktioita, joista jokainen vastasi omasta tarkastuksestaan. Näin eri tarkastusvaiheet muodostivat yhdessä toimivan kokonaisuuden, jossa jokaisella osalla oli selkeä rooli prosessissa. Jokainen yksityinen funktio oli vastuussa tietyn osa-alueen, kuten moottoridatan, SIM-kortin tilan tai jonkin muun oleellisen asian tai arvon tarkastamisesta. Tavoitteena oli selkeyttää kokonaisuutta ja varmistaa, että jokainen tarkastettava osa-alue oli erikseen hallittavissa. Tämä lähestymistapa mahdollisti yksittäisten tarkastusten muokkaamisen tai laajentamisen ilman, että koko käyttöönottoprosessia tarvitsi kirjoittaa uudelleen.

```php
/** @param array<object> $deviceData */
1 usage  Tuomaaaas
private function checkForEngine(array $deviceData): StepResult
{
    $status = isset($deviceData['engines'][0]['hours'])
        ? Status::OK
        : Status::ERROR;

    $message = $status === Status::OK ? 'Engine detected' : 'Engine not detected!';

    return new StepResult($status, $message);
}

/** @param array<object> $deviceData */
1 usage  Tuomaaaas
private function checkForSIM(array $deviceData): StepResult
{
    $status = !empty($deviceData['imsi'])
        ? Status::OK
        : Status::ERROR;

    $message = $status === Status::OK ? 'SIM-card detected' : 'SIM-card not detected!';

    return new StepResult($status, $message);
}
```

Kuva 12. Q-näytön käyttöönottovaiheita.

Kuvassa 12 esitellään kaksi Q-näytön käyttöönottovaihetta. Ensimmäisessä vaiheessa tarkastellaan, tunnistetaanko veneen perämoottoria NMEA-verkosta, ja toisessa vaiheessa tarkastellaan, tunnistetaanko SIM-korttia Q-näytöstä. Molemmat funktiot saavat parametrina Q-näytön rajapintavastauksen ja molemmat

funktiot palauttavat ennalta määritellyn rakenteen mukaisen palautteen. Funktioiden parametrit ovat lisäksi annotoitu PHPStanilla, mikä mahdollistaa koodin staattisen analyysin.

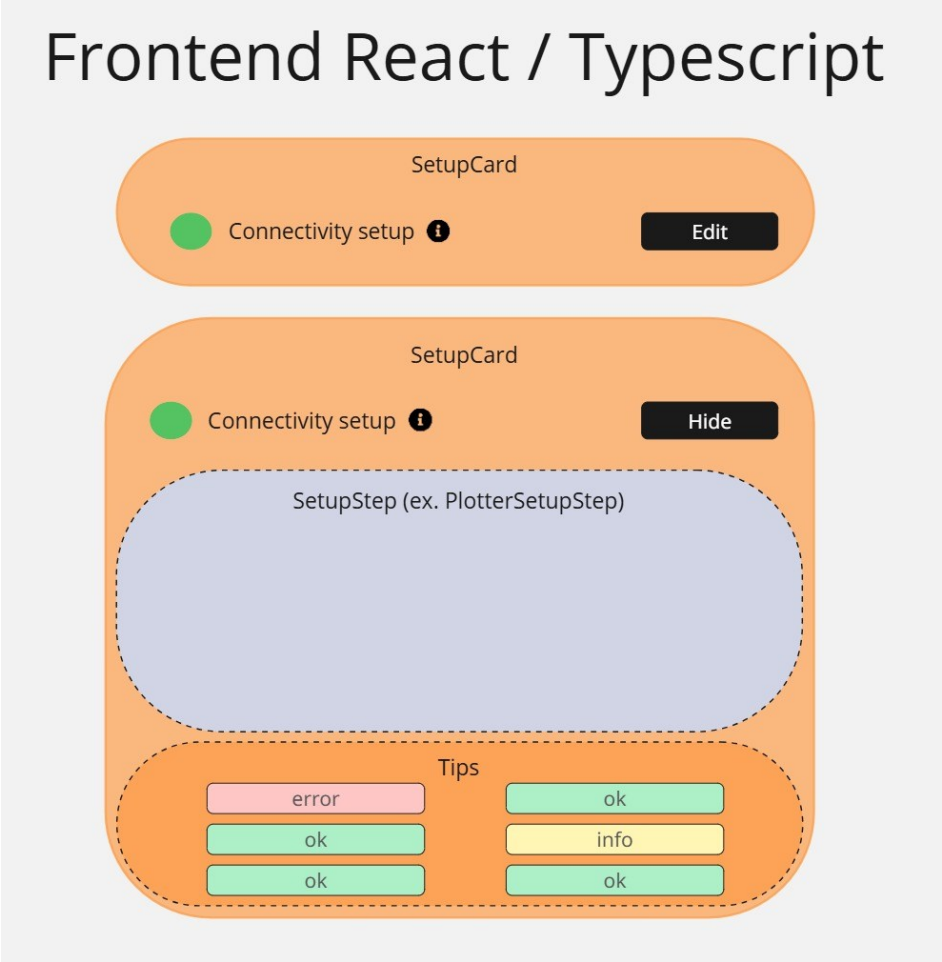
5.4 Käyttöliittymä ja ulkoasu

Projektin käyttöliittymä rakennettiin Reactilla hyödyntäen komponenttipohjaista arkkitehtuuria. Tietojen hakeminen palvelimelta toteutettiin Axios-kirjastolla, joka vastasi HTTP-pyynnöistä ja mahdollisti tiedon hakemisen rajapinnasta. Axiosin päällä toimi React Query, joka hallinnoi datan hakua, käsitteli lataus- ja virhetiloja sekä optimoi välimuistin käyttöä.

Käyttöliittymä jaettiin pieniin, itsenäisiin komponentteihin, joita voitiin yhdistellä joustavasti eri näkymissä. Tämä vähensi toistuvaa koodia ja mahdollisti uudelleenkäytettävyyden – esimerkiksi samoja painikkeita, näkymiä ja tyylittelyjä voitiin käyttää eri konteksteissa ilman, että niitä täytyi koodata joka näkymässä uudelleen. Tällöin käyttöliittymät pysyivät samannäköisinä, kun samoja komponentteja käytettiin kaikkialla. Lisäksi se nopeutti kehitystyötä merkittävästi.

Reactin perusperiaate on se, että komponentit voivat hallita omaa tilaansa (state) ja vastaanottaa ulkoisia tietoja (props). Kun komponentin tila tai siihen saatu tieto muuttuu, React renderöi (piirtää uudelleen) vain sen osan käyttöliittymästä, joka on muuttunut. Tämä tekee Reactista tehokkaan, sillä vain tarvittavat osat päivittyvät, mikä parantaa sovelluksen suorituskykyä.

Käyttöönottoprosessin eri käyttöönottovaiheet toteutettiin itsenäisinä komponentteina, mutta niiden pohjana käytettiin yhteistä SetupCard-komponenttia. Tämä komponentti toimi rakenteellisena peruspalikkana, johon kunkin käyttöönottovaiheen oma logiikka ja sisältö integroitiin. Tämän rakenteen ansiosta käyttöliittymä pysyi modulaarisena, skaalautuvana ja helposti ylläpidettävänä. Kuva 13 havainnollistaa yksittäisen käyttöönottovaiheen rakennetta käyttöliittymässä.



Kuva 13. Havainnekuva käyttöönottovaiheen rakenteesta käyttöliittymässä

SetupCard oli käyttöliittymän keskeinen elementti, joka huolehti käyttöönottovaiheen esittämisestä visuaalisesti. Sen vastuulla oli taulukon 2 mukaiset ominaisuudet.

Taulukko 2. Kuvaus SetupCard-komponentin ominaisuuksista

Ominaisuus	**Käyttötarkoitus**
Otsikko	Jokainen vaihe sai oman nimensä.
Tilailmaisin	Komponentti sai tilan ("ok", "info" tai "error"), jonka perusteella se vaihtoi väriä.
Info-ikoni	Käyttäjä pystyi navigoimaan ohjeisiin info-ikonista.
Ohjevinkit	Jokainen vaihe sai omat ohjevinkit, jotka olivat käyttöönoton tarkastusvaiheet, jotka ohjasivat käyttäjää, mitkä asiat oli suoritettu onnistuneesti ja missä kohdin tarvittiin vielä toimenpiteitä.
Refresh painike	"Refresh"-painike oli valinnainen parametri, joka lisättiin vain, jos se oli tarpeellinen kyseiselle käyttöönottovaiheelle.
Edit painike	Painike avasi käyttöönottokortin kaikki tiedot näkyville ja piilotti ne uudelleen.

Koska SetupCard ei sisältänyt kiinteää logiikkaa vaan pelkästään rakenteen ja ulkoasun, sitä voitiin käyttää monessa eri kontekstissa ja kaikissa eri käyttöönottovaiheissa. Sen saamat ominaisuudet (propsit) määrittelivät, mitä sisältöä ja toimintoja komponenttiin tuotiin, mikä mahdollisti sen muokkaamisen eri käyttöönottovaiheisiin sopivaksi ilman, että peruspohjaa tarvitsi kirjoittaa uudestaan. Esimerkiksi ohjevinkit toimitettiin SetupCard-komponentille listana, jolloin SetupCard piirsi ne omien tyylimäärittelyjensä mukaan.

Yksi opinnäytetyön toteutusvaiheen osa-alueista oli käyttöönottotyökalun ohjeiden luominen Notion-portaaliin. Notion-portaalissa ohjeet oli selkeästi jaoteltu käyttöönottovaiheiden mukaan, mikä helpotti käyttäjän siirtymistä oikeiden ohjeiden pariin tarpeen mukaan. Jokaisen käyttöönottovaiheen info-ikonia klikkaamalla käyttäjä pääsi suoraan kyseisen vaiheen ohjeisiin ilman tarvetta etsiä niitä erikseen. Ohjeet tarjosivat selkeät ja tarkat ohjeet virheiden korjaamiseen, ja ne sisälsivät kaikki käyttöliittymän virheviestit. Tavoitteena oli, että käyttäjä voisi helposti tunnistaa virheviestin ohjeista ja saada niistä selkeän polun ongelman ratkaisemiseksi. Näin käyttäjä sai tarvitsemaansa tietoa ja pystyi hoitamaan ongelman itsenäisesti ilman ulkopuolista apua. Ohjeet rakennettiin Notion-portaaliin, jotta kaikilla työntekijöillä oli mahdollisuus muokata ohjeita tarpeen mukaan ilman, että niitä tarvitsi kovakoodata ohjelmakoodin sekaan. Tämä helpotti ohjeiden ylläpidettävyyttä merkittävästi.

5.5 Tuotantoon vieminen

Projektin viimeisessä vaiheessa valmis koodi vietiin tuotantoympäristöön. Ennen siirtoa varmistettiin, että kaikki tarvittavat testit oli suoritettu ja koodi oli valmis tuotantoon. Koodin siirron jälkeen palvelua seurattiin tarkasti NewRelic-nimisen työkalun avulla, joka antoi kattavan yleiskuvan palvelimen tilasta ja prosessien toiminnasta tuotantoympäristössä. Seurannan pääasiallisena tarkoituksena oli varmistaa, ettei palvelussa ilmenisi virheilmoituksia ja että sen suorituskyky pysyisi vakaana. Erityisesti huomiota kiinnitettiin siihen, ettei prosessoreissa ollut piikkejä, järjestelmän vasteajat pysyivät kohtuullisina eikä virheilmoituksia ilmennyt. Tavoitteena oli taata, että palvelu toimisi sujuvasti ilman häiriöitä ja että mahdolliset ongelmat, jotka eivät olleet ilmenneet testausvaiheessa, voitaisiin havaita ja ratkaista nopeasti.

6 Testaus ja laadunvarmistus

Testaus on oleellinen osa modernia ohjelmistokehitysprosessia ja sen laadunvarmistusta. Huolellisesti toteutetulla testauksella varmistetaan, että ohjelmisto toimii odotetulla tavalla ja tuottaa odotettuja lopputuloksia. Kattava testaus varmistaa, että mahdolliset virheet havaitaan ja korjataan jo ennen tuotantoon siirtymistä.

Tässä projektissa testaus toteutettiin useilla eri tasoilla:

- **Yksikkötestit (Unit Test):** Yksikkötestauksen tavoitteena oli varmistaa, että yksittäiset komponentit ja funktiot toimivat oikein erillään muusta ohjelmasta. Yksikkötestejä kirjoitettiin palvelinpuolen käyttöönottovaiheiden ohjelmakoodiin.

- **Ominaisuustestit (Feature Test):** Ominaisuustesteillä testattiin laajempia kokonaisuuksia, joissa eri komponentit ja järjestelmän osat toimivat yhdessä.

- **Käyttäjätestaus (User Testing):** Käyttäjätestauksessa keskityttiin käyttäjien yksilölliseen käyttökokemukseen sekä siihen, kuinka hyvin käyttöönottotyökalu vastasi käyttäjien tarpeisiin ja odotuksiin. Lisäksi pyrittiin selvittämään, kuinka hyvin käyttöliittymä havainnollisti käyttäjälle erilaisia tilanteita sekä antoi selkeitä ohjeita vikojen selvittämiseen.

6.1 Yksikkötestaus

Yksikkötestauksessa käytettiin PHPUnit-testikirjastoa palvelinpuolen testien toteutukseen. Yksikkötestauksen tavoitteena oli varmistaa, että yksittäiset funktiot, luokat ja komponentit toimivat oikein erillään muusta ohjelmasta. Tässä projektissa yksikkötestejä kirjoitettiin yksittäisten käyttöönottovaiheiden testaamiseen. Tavoitteena oli varmistaa, että ne tuottavat haluttuja tuloksia määritellyillä syötteillä. Yksikkötestit ovat automatisoituja testejä, joita ajetaan jatkuvasti kehitystyön aikana. Tässä projektissa yksikkötestejä kirjoitettiin osittain jo ennen varsinaisen ohjelmakoodin toteuttamista testivetoisen kehitysmallin (TDD) periaatteiden mukaisesti.

```php
public function testLockboxAssociatedButTooLowBatteryLevel(): void
{
    $user = User::factory()->create();
    $lock = SmartLock::factory()->for($this->organization)->create();
    SmartLockBatteryLevel::factory()
        ->for($user)
        ->for($lock)
        ->withBatteryLevel( batteryLevel: 20)
        ->createdAt( createdAt: '2000-01-01T00:00:00Z')
        ->create();

    $this->boat->smartLock = $lock;

    $response = $this->setupStep->process($this->boat);

    $expected = [
        new StepResult( status: Status::OK,  message: 'Lockbox found and linked successfully!',  steps: null),
        new StepResult( status: Status::INFO,  message: 'Battery level is too low!',  steps: null)
    ];

    $this->assertEquals( expected: Status::INFO, $response->status);
    $this->assertEquals($expected, $response->steps);
}
```

Kuva 14. Yksikkötesti lukkoboksin käyttöönotosta, kun paristotaso on liian alhainen

Kuvassa 14 esitellään yksikkötesti lukkoboksin käyttöönotosta. Testissä määritellään ensin alkutilanne: luodaan käyttäjä, lukkoboksi ja erillinen lukkoboksin paristotason mittaustulos. Seuraavaksi lukko assosioidaan veneeseen. Lopuksi vene, johon lukko ja sen paristotason mittaustulos kuuluvat, lähetetään setupStep-luokalle, ja saadun palautteen odotetaan vastaavan ennakoitua tulosta.

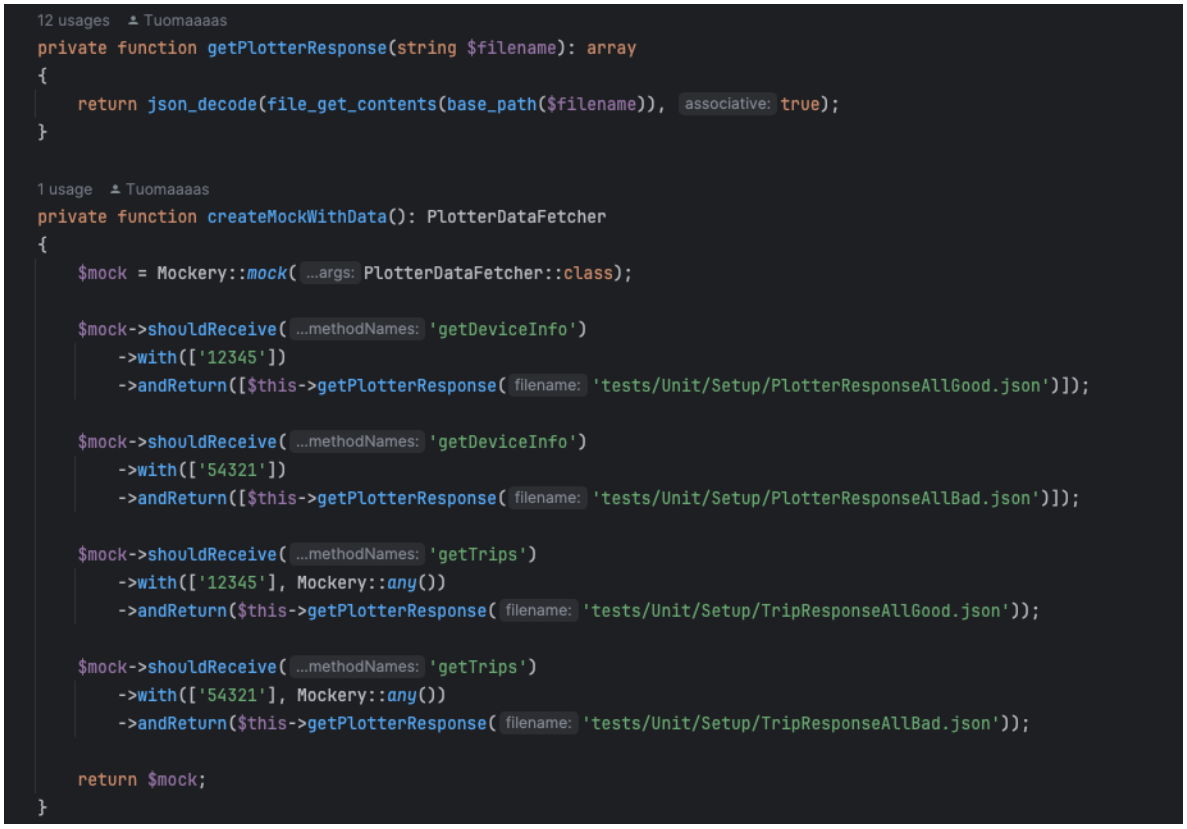
Mikäli palaute ei vastaa ennakoitua tulosta, yksikkötesti ei mene läpi. Mikäli palaute vastaa ennakoitua tulosta, yksikkötesti menee läpi. Kuvan 14 esimerkkitilanteessa lukkoboksin paristotaso on liian alhainen, ja testin tarkoituksena on varmistaa, että tämä huomioidaan oikein palautteessa.

```php
public function testLockboxAssociatedButNoBatteryLevel(): void
{
    $lock = SmartLock::factory()->for($this->organization)->create();

    $this->boat->smartLock = $lock;

    $response = $this->setupStep->process($this->boat);

    $expected = [
        new StepResult( status: Status::OK,  message: 'Lockbox found and linked successfully!',  steps: null),
        new StepResult( status: Status::ERROR,  message: 'Battery level not found!',  steps: null)
    ];

    $this->assertEquals( expected: Status::ERROR, $response->status);
    $this->assertEquals($expected, $response->steps);
}
```

Kuva 15. Yksikkötesti lukkoboksin käyttöönotosta, kun paristotasoa ei ole tiedossa.

Kuvassa 15 esitellään toinen lukkoboksin käyttöönottoa käsittelevä yksikkötesti, jossa testataan tilannetta, jossa paristotason mittaustieto puuttuu kokonaan. Testin alussa määritellään alkutilanne luomalla lukkoboksi, joka tämän jälkeen assosioidaan veneeseen. Tällä kertaa paristotason mittaustulosta ei kuitenkaan luoda lainkaan. Varsinainen testaus tapahtuu jälleen, kun vene toimitetaan setupStep-luokalle, jonka palautteen odotetaan vastaavan ennalta määriteltyä odotusarvoa. Tässä tapauksessa testauksessa on tarkoitus varmistaa, että palautteessa huomioidaan se, että paristotason mittaustulos puuttuu kokonaan.

Yksikkötestauksen yksi merkittävistä haasteista on testattavan luokan riippuvuudet muihin komponentteihin kuten tietokantoihin, ulkopuolisiin rajapintoihin tai muihin ohjelmistomoduuleihin. Tämän ongelman ratkaisemiseksi käytettiin hyödyksi mockaamista eli muiden komponenttien riippuvuuksien korvaamista simuloiduilla palautteilla. Tässä tapauksessa simuloitiin ulkopuolisen rajapinnan vastausta Q-näytön käyttöönottovaiheessa:



```
12 usages  Tuomaaaas
private function getPlotterResponse(string $filename): array
{
    return json_decode(file_get_contents(base_path($filename)), associative: true);
}

1 usage  Tuomaaaas
private function createMockWithData(): PlotterDataFetcher
{
    $mock = Mockery::mock( ...args: PlotterDataFetcher::class);

    $mock->shouldReceive( ...methodNames: 'getDeviceInfo')
        ->with(['12345'])
        ->andReturn([$this->getPlotterResponse( filename: 'tests/Unit/Setup/PlotterResponseAllGood.json')]);

    $mock->shouldReceive( ...methodNames: 'getDeviceInfo')
        ->with(['54321'])
        ->andReturn([$this->getPlotterResponse( filename: 'tests/Unit/Setup/PlotterResponseAllBad.json')]);

    $mock->shouldReceive( ...methodNames: 'getTrips')
        ->with(['12345'], Mockery::any())
        ->andReturn($this->getPlotterResponse( filename: 'tests/Unit/Setup/TripResponseAllGood.json'));

    $mock->shouldReceive( ...methodNames: 'getTrips')
        ->with(['54321'], Mockery::any())
        ->andReturn($this->getPlotterResponse( filename: 'tests/Unit/Setup/TripResponseAllBad.json'));

    return $mock;
}
```

Kuva 16. Mock-objektin luominen yksikkötestauskoodissa

Kuvassa 16 luodaan Mock-objekti, jossa mallinnetaan simuloidun luokan funktioiden palautusarvoja. Simuloituun palautteeseen rakennettiin logiikka, jossa palautteen sisältö riippuu funktiolle annetusta parametrista. Tässä tapauksessa, kun parametrina annetaan '12345', palautus kertoo, että kaikki on kunnossa. Jos taas parametrina on '54321', palautus simuloi tilannetta, jossa kaikki on pielessä. Näin saatiin testattua helposti ja todenmukaisesti luokkaa, jolla oli riippuvuuksia muihin komponentteihin.

```
Tuomaaaas
public function testPlotterSetupAllGood(): void
{
    $this->boat->plotter_serial_id = '12345';
    $response = $this->setupStepMocked->process($this->boat);

    $expected = [
        new StepResult( status: Status::OK,  message: 'Engine detected',  steps: null),
        new StepResult( status: Status::OK,  message: 'SIM-card detected',  steps: null),
        new StepResult( status: Status::OK,  message: 'Trips found',  steps: null),
        new StepResult( status: Status::OK,  message: 'Fuel sensor detected! Fuel tank 100% full',  steps: null),
        new StepResult( status: Status::OK,  message: 'Plotter linked and Organization correct',  steps: null)
    ];

    $this->assertEquals( expected: Status::OK, $response->status);
    $this->assertEquals($expected, $response->steps);
}
```

Kuva 17. Q-näytön käyttöönottovaiheen yksikkötesti, jossa kaiken oletetaan olevan kunnossa

Kuvissa 17 ja 18 havainnollistetaan Q-näytön käyttöönottoon liittyvää yksikkötestiä. Syötteenä hyödynnetään simuloitua Q-näytön rajapintavastausta. Kuvassa 17 kaiken oletetaan olevan kunnossa ja käyttöönoton valmis, kun taas kuvassa 18 kaiken oletetaan olevan pielessä ja käyttöönoton kesken. Syötettä kontrolloidaan veneen karttaplotterin sarjanumerolla, joka lopulta reflektoi simuloidun funktion parametriarvoon.

```
Tuomaaaas
public function testPlotterSetupAllBad(): void
{
    $this->boat->plotter_serial_id = '54321';
    $response = $this->setupStepMocked->process($this->boat);

    $expected = [
        new StepResult( status: Status::ERROR,  message: 'Engine not detected!',  steps: null),
        new StepResult( status: Status::ERROR,  message: 'SIM-card not detected!',  steps: null),
        new StepResult( status: Status::ERROR,  message: 'No trips found!',  steps: null),
        new StepResult( status: Status::INFO,  message: 'Fuel sensor not detected or fuel tank empty!',  steps: null),
        new StepResult( status: Status::ERROR,  message: 'Plotter not linked or Organization incorrect!',  steps: null)
    ];

    $this->assertEquals( expected: Status::ERROR, $response->status);
    $this->assertEquals($expected, $response->steps);
}
```

Kuva 18. Q-näytön käyttöönottovaiheen yksikkötesti, jossa kaiken oletetaan olevan pielessä.

Vastaavat yksikkötestit luotiin jokaiselle käyttöönottovaiheelle. Tavoitteena oli varmistaa, että jokainen vaihe toimii oletetulla tavalla ja palauttaa tilanteeseen sopivan vastauksen.

6.2 Ominaisuustestaus

Ominaisuustestauksessa keskityttiin palvelinpuolen kokonaisuuden toiminnallisuuden testaamiseen. Testissä tehtiin kutsu palvelimen `--/{id}/setup` -rajapintaan ja tarkistettiin, että se palauttaa oikeanlaisen vastauksen oikeilla arvoilla. Tässä testissä varmistettiin, että HTTP-pyynnön status on 200 (OK), palautuvia käyttöönottovaiheita on neljä, ja että ne ovat juuri oikeat vaiheet oikeassa järjestyksessä. Kuvassa 19 esitellään yksi ominaisuustesti.

```
Tuomas Mellin *
public function testCityBoatSetup(): void
{
    $this
        ->withHeaders(['Skipperi-Organization' => $this->organization->id])
        ->actingAs($this->admin)
        ->get( uri: "                    /{$this->boat->id}/setup")
        ->assertStatus( status: 200)
        ->assertJsonCount( count: 4)
        ->assertJson([
            ['step' => 'BoatInfoSetupStep'],
            ['step' => 'PlotterSetupStep'],
            ['step' => 'LockboxSetupStep'],
            ['step' => 'BoatPublishingSetupStep']]);
}
```

Kuva 19. Ominaisuustesti

Ominaisuustestin tarkoituksena ei ollut arvioida, ovatko käyttöönottovaiheet valmiita, vaan varmistaa, että oikea määrä vaiheita palautuu oikealle veneelle oikeassa järjestyksessä. Tällä hetkellä jokaiselle veneelle palautuvat samat käyttöönottovaiheet samassa järjestyksessä, mutta tulevaisuudessa näin ei välttämättä ole, jos eri veneisiin määritellään omat yksilölliset käyttöönottovaiheensa. Siksi testauksen suunnittelussa ei keskitytty pelkästään nykyhetkeen vaan myös siihen, että ratkaisu toimii joustavasti tulevaisuudessa.

6.3 Käyttäjätestaus

Käyttäjätestauksessa ensimmäinen tuotantokelpoinen toteutus vietiin testattavaksi aidossa käyttöympäristössä. Kyseessä ei siis ollut vielä valmis tuote vaan kehitystyön vaihe, jossa keskeisenä tavoitteena oli kerätä aitoa ja monipuolista palautetta. Tavoitteena oli siis arvioida ohjelmiston toimivuutta ja käytettävyyttä realistisissa olosuhteissa.

Käyttäjätestaukseen osallistui kaksi henkilöä, jotka edustivat keskenään hyvin erilaisia käyttäjäprofiileja. Toinen testaajista oli yrityksen pitkäaikainen työntekijä, joka oli käyttänyt Skipperi Consolea aiemmin ja tunsi sen toimintaperiaatteet. Toinen testaaja puolestaan ei ollut yrityksen työntekijä eikä hänellä ollut aiempaa kokemusta Skipperi Consolen käytöstä. Tällainen käyttäjäasetelma valittiin tarkoituksella, jotta testauksessa voitaisiin havainnoida sekä kokeneen että kokemattoman käyttäjän näkökulmia ja käyttöhaasteita. Testauksen toteutuksessa käytettiin Liitteen 1 ja Liitteen 2 mukaista paperipohjaista ohjeistusta, jonka avulla käyttäjille annettiin tarvittavat tiedot tehtävän suorittamiseen ilman lisäapua tai ohjausta. Tällä tavalla pyrittiin saamaan mahdollisimman autenttinen kuva heidän omasta selaamisestaan ja ongelmanratkaisustaan. Lisäksi testaukset järjestettiin eri ajankohtina, mikä esti käyttäjiä vaikuttamasta toistensa kokemuksiin.

Testausasetelmana hyödynnettiin venettä, joka oli valmiina käyttökunnossa satamassa. Testaus suoritettiin käytännössä kenttäolosuhteissa, joissa käyttäjän tehtävänä oli luoda uusi vene järjestelmään ja suorittaa sen käyttöönotto alusta alkaen. Käyttäjä sai ohjeet järjestelmän käyttöönottoprosessista, ja hänen tuli suorittaa kaikki tarvittavat vaiheet itsenäisesti.

Testauksen tulokset vastasivat suurelta osin odotuksia, mutta esiin nousi myös joitain yllättäviä havaintoja. Kokeneella käyttäjällä oli vahva intuitio käyttää uutta käyttöliittymää edellisen käyttöliittymämallin mukaisesti. Uuteen käyttöliittymään tottuminen vaati selkeästi aikaa sekä työskentelyä aivoilta. Lisäksi kokenut käyttäjä yritti klikata käyttöliittymässä esiintyviä ohjevinkkejä, kuten ”Trips found” tai ”Engine detected”. Käyttäjä oletti niiden avaavan lisätietoja tai syventävän tietämystä aiheesta. Tämä siis viittaa siihen, että käyttöliittymä loi mielikuvan interaktiivisuudesta, jota ei kuitenkaan ollut toteutettu. Idea oli kuitenkin hyvä ja erittäin toteutuskelpoinen.

Toinen, kokemattomampi käyttäjä omaksui uuden käyttöliittymän huomattavasti kokenutta käyttäjää paremmin ja osasi navigoida käyttöliittymässä intuitiivisesti. Hän ymmärsi lisäksi yllättävän hyvin, mitä kukin käyttöönottovaihe tarkoittaa ja mikä sen funktio on kokonaisuuden kannalta. Kuitenkin pieni epäselvyys heräsi erityisesti käyttöliittymän painikkeiden toimivuudessa. Esimerkiksi, kun koehenkilö painoi ”Refresh”-nappia, järjestelmä ei tarjonnut mitään visuaalista palautetta, kuten ilmoitusta tai muutosta näytöllä, joka vahvistaisi, että painallus oli rekisteröity ja toiminto suoritettu. Tämä sai hänet painamaan nappia uudelleen ja uudelleen odottaen, milloin käyttöliittymässä tapahtuu jokin muutos.

Käyttäjätestauksessa huomattiin myös merkittäviä teknisiä haasteita, jotka liittyivät käyttöönottotyökalun riippuvuuteen kolmansien osapuolien palveluista ja rajapinnoista. Esimerkiksi kun testiveneen Q-näytöstä oli poistettu NMEA-kaapeli, kolmannen osapuolen rajapinta ilmoitti edelleen tunnistavansa moottorin. Vaikka rajapinnan mukaan moottori oli tunnistettu, konetunnit ilmoitettiin olevan arvoltaan 0. Konetunnit olivat todellisuudessa lähemmäs 500. Tämä ilmiö osoitti, että olimme tässä tapauksessa vahvasti riippuvaisia kolmansien osapuolien toimittamasta tiedosta. Q-näytön asennusta analysoitaessa jouduimmekin tukeutumaan yksinomaan kolmannen osapuolen rajapinnan tarjoamaan tietoon, mikä heikensi suoraan tehtyjen johtopäätösten luotettavuutta.

Vastaava tilanne ilmeni myös SIM-kortin kohdalla: rajapinta ilmoitti kortin olevan asennettuna, vaikka se todellisuudessa oli irrotettuna Q-näytöstä. Nämä tapaukset korostivat, miten kriittistä on ottaa huomioon ulkoisten palveluiden ja rajapintojen mahdolliset häiriöt. Lopputuloksessa oli hyväksyttävä se, että vaikka kohdeyrityksen oma järjestelmä toimisi moitteettomasti, kolmannen osapuolen palvelut voivat vaikuttaa ratkaisevasti koko käyttöönottotyökalun toimivuuteen. Tämä haaste pakotti pohtimaan entistä tarkemmin, miten ohjelmistokehityksessä voidaan ennakoida ja hallita ulkopuolisiin järjestelmiin liittyviä epävarmuustekijöitä.

Yhteenvetona käyttäjätestaus osoitti olevansa erittäin arvokas osa ohjelmistotuotantoa. Kyseessä ei ole pelkästään viimeistelyprosessi, vaan iteratiivisen ohjelmistokehityksen yksi vaihe, jossa tuote viedään testattavaksi todellisissa käyttötilanteissa. Käyttäjätestauksessa tavoitteena on oppia käyttötapauksista, jotta tuotetta voidaan kehittää havaintojen pohjalta. Harvoin tuotteen lopullista käyttötapaa tai tarkoitusta pystytäänkään täysin suunnittelemaan jo alkuvaiheessa. Tarkemmat käyttäjätestauksen tulokset sekä vaiheet löytyvät Liitteestä 3 sekä Liitteestä 4.

7 Pohdinta ja yhteenveto

Tämän insinöörityön tavoitteena oli kehittää web-pohjainen käyttöönottotyökalu, joka keskittyi IoT-laitteiden asennukseen ja käyttöönottoon Skipperi Fleet -palvelussa. Työkalu suunniteltiin tukemaan erityisesti niitä käyttäjiä, joilla ei ole syvällistä teknistä osaamista, mutta jotka kuitenkin vastaavat laitteiden asennuksesta ja käyttöönotosta kenttäolosuhteissa. Tavoitteena oli vähentää virheitä, nopeuttaa käyttöönottoa ja varmistaa, että kaikki kriittiset vaiheet suoritetaan oikein.

Kokonaisuudessaan projekti onnistui hyvin asetettuihin tavoitteisiin nähden. Kehitetty käyttöliittymä oli selkeä, vaiheittain ohjaava ja rakenteeltaan intuitiivinen. Käyttäjätestauksen tulokset osoittivat, että työkalulla oli selkeä vaikutus käyttöönoton onnistumiseen ja jouhevuuteen. Lisäksi työkalun oheen rakennetut ohjeistukset auttoivat käyttäjiä ratkaisemaan virhetilanteita itsenäisesti ilman ulkopuolista tukea. Toteutus yhdistettiin olemassa olevaan Skipperi Console -hallintatyökaluun.

Tässä työssä toteutettiin kattava tutkimus- ja suunnitteluprosessi. Usein kehitystyön aloittamiseen liittyy suuri innostus, mikä johtaa siihen, että suunnitteluvaiheelta pyritään siirtymään toteutukseen mahdollisimman nopeasti. Tässä projektissa tuli kuitenkin huomattua, että suunnittelu muodosti lähes puolet koko työmäärästä, erityisesti yhdistettynä testivetoiseen kehitysmalliin. Toteutusvaiheessa ei juurikaan tarvinnut enää pohtia vaatimusmäärittelyjä tai ohjelmalogiikkaa, sillä ne oli kirjattu selkeästi auki jo suunnittelun aikana.

Työn aikana heräsi useasti ajatus – kuinka kauan paljon organisaatioissa käytetään aikaa asioiden tarkastamiseen ja seurantaan? Monilla aloilla joudutaan jatkuvasti tarkkailemaan, onko jokin asia kunnossa – olipa kyse tuotannosta, logistiikasta tai asiakaspalvelusta. Tämän projektin pohjalta syntyi ajatus, että käyttöliittymien ei tarvitsisi olla vain tiedon esittämistä varten, vaan ne voisivat aktiivisesti kertoa käyttäjille tilanteista: mitä on tehty, mitä on kesken ja mihin tulisi seuraavaksi kiinnittää huomiota. Tällainen toimintakeskeinen lähestymistapa voi vähentää turhaa tarkistelua ja vapauttaa aikaa varsinaiseen tekemiseen. Lisäksi se pienentäisi inhimillisten virheiden määrää.

Käyttöönotto ei ole ongelma vain IoT-laitteissa. Monissa järjestelmissä on edelleen haasteita siinä, että käyttäjille ei ole annettu riittävästi tukea tai näkymää siihen, missä vaiheessa hän on, mitä on vielä tehtävänä ja mitä mahdollisesti edellisissä vaiheissa on mennyt vikaan. Tämän työn perusteella voi arvioida, että käyttöönottotyökaluille saattaisi olla kysyntää myös muissa konteksteissa, mikäli ne rakennetaan aidosti käyttäjälähtöisesti, helposti ylläpidettäviksi ja visuaalisesti selkeiksi.

Tämän työn onnistumisen kannalta huomattavimpia vahvuuksia olivat selkeä käyttäjäohjeistus, modulaarinen arkkitehtuuri ja kokonaissuunnittelun iteratiivinen lähestymistapa. Käyttäjätestauksessa saatiin arvokasta palautetta, joka osoitti, että työ tuotti sekä odotettuja tuloksia että yllättäviä huomioita. Näiden perusteella on hyvä jatkaa käyttöönottotyökalun kehitystyötä.

Opinnäytetyön toteutus toi esiin useita kehityskohteita, joiden huomioiminen parantaisi käyttöönottotyökalun toimivuutta ja käytettävyyttä entisestään. Käyttöliittymän osalta selkein puute liittyi visuaalisen palautteen vähäisyyteen. Käyttäjätestauksessa havaittiin, että toiminnon rekisteröitymisestä ei aina saanut selkeää vahvistusta, mikä aiheutti epävarmuutta. Tämän perusteella käyttöliittymään tulisi lisätä visuaalisia elementtejä, kuten ilmoituksia tai animaatioita, jotka osoittavat toiminnon onnistuneen.

Opinnäytetyön aikana opittiin erityisesti suunnitteluprosessin, iteratiivisen kehitystyön sekä käyttäjätestauksen merkityksestä käytännön ratkaisun rakentamisessa. Lisäksi opinnäyteyössä opittiin modernista web-kehityksestä sekä IoT-laitteiden integroimisesta osaksi suurempia kokonaisuuksia.

Työn tuloksena saatiin toimiva ja tuotantokäyttöön viety ratkaisu, joka selkeyttää ja nopeuttaa IoT-laitteiden käyttöönottoa. Lisäksi työn tuloksena saatiin kattava dokumentaatio niin tehdystä työstä kuin taustalla tehdystä tutkimustyöstä. Kaikkia työn alkuperäisiä tavoitteita voidaan pitää pääosin saavutettuina.

Lähteet

1 Skipperi. Tietoa meistä. Verkkoaineisto.
https://www.skipperi.fi/tietoa-meista.
Luettu 15.2.2025

2 Skipperi. Lehdistötiedotteet. Verkkoaineisto.
https://www.skipperi.fi/press.
Luettu 15.2.2025

3 Skipperi. Pressitiedote 21.09.2021. Verkkoaineisto.
https://www.skipperi.fi/lokikirja/pressitiedote-21-9-21-geofencing.
Luettu 15.2.2025

4 Skipperi. Veneilykohteet kartalla. Verkkoaineisto.
https://www.skipperi.fi/lokikirja/veneilykohteet-kartalla.
Luettu 15.2.2025

5 Esconet Blog. NMEA 2000 -verkko ja sen kytkennät. Verkkoaineisto.
https://www.esconet.fi/blog/elektroniikka-13/nmea-2000-verkko-ja-sen-kyt-kennat-31.
Luettu 4.3.2025

6 National Marine Electronics Association. NMEA 2000. Verkkoaineisto.
https://www.nmea.org/nmea-2000.html.
Luettu 4.3.2025

7 Kipparilehti. Näin toimii veneen tietoverkko – NMEA 2000. Verkkoaineisto.
https://kipparilehti.fi/nain-toimii-veneen-tietoverkko-nmea-2000/.
Luettu 4.3.2025

8 Golden Channels. Yamaha Command Link. Verkkoaineisto.
https://goldenchannels.com/yamaha-command-link-nmea-and-plus/.
Luettu 4.3.2025

9 ContinuousWave. Yamaha Engine NMEA 2000 Connection. Verkkoa-ineisto.
https://continuouswave.com/whaler/reference/Yamaha/Yamaha_En-gine_NMEA_2000_Connection.pdf.
Luettu 4.3.2025

10 The Q Experience. Q Display 2 Series. Verkkoaineisto. https://theqexpe-rience.com/q-display-2-series-fi/.
Luettu 5.3.2025

11 The Q Experience. Q Experience -mediakirjasto. Verkkoaineisto.
https://theqexperience.com/tervetuloa-q-experience-mediakirjastoon/.
Luettu 5.3.2025

12 The Q Experience. Q2 Support. Verkkoaineisto.
https://theqexperience.com/q2-support/.
Luettu 5.3.2025

13 Igloohome. Keybox 3. Verkkoaineisto.
https://www.igloohome.co/products/keybox-3.
Luettu 6.3.2025

14 Kotiautomaatiokauppa. Igloohome Smart Keybox EU. Verkkoaineisto.
https://www.kotiautomaatiokauppa.fi/fi/tuote/igloohome-smart-keybox-eu.
Luettu 6.3.2025

15 Igloohome. How it works: AlgoPIN. Verkkoaineisto.
https://www.igloohome.co/how-it-works.
Luettu 6.3.2025

16 Igloohome. Masking PINs. Verkkoaineisto.
https://support.igloohome.co/support/solutions/articles/35000160156-masking.
Luettu 6.3.2025

17 Microsoft. TypeScript. Verkkoaineisto.
https://www.typescriptlang.org/.
Luettu 7.3.2025

18 Meta. React. Verkkoaineisto.
https://react.dev/.
Luettu 7.3.2025

19 Meta. React Native. Verkkoaineisto.
https://reactnative.dev/.
Luettu 7.3.2025

20 Expo. Expo Developer Tools. Verkkoaineisto.
https://expo.dev/.
Luettu 7.3.2025

21 Wikipedia. PHP. Verkkoaineisto.
https://fi.wikipedia.org/wiki/PHP.
Luettu 7.3.2025

22 Laravel. Laravel PHP Framework. Verkkoaineisto.
https://laravel.com/.
Luettu 7.3.2025

23 OpenAI. ChatGPT. Verkkoaineisto.
https://chatgpt.com/.
Luettu 11.4.2025

Käyttäjätestauksen ohjeet

Käyttäjätestaus

Uusi lippulaivamme on saapunut! 🚤✨

Olet Mikko Mallikas, Skipperin pääkaupunkiseudun operatiivisen osaston päällikkö. 🎩

Tänään sinulla on tärkeä tehtävä: uusi vene on saapunut suoraan tehtaalta, ja siitä tulee laivastosi tuorein lippulaiva!

Henkilökuntasi on jo laskenut veneen vesille ja se odottaa nyt sinua omalla laituripaikallaan. Sinun tehtäväsi on lisätä vene osaksi Skipperin laivastoa ja varmistaa, että se on käyttövalmis klubin jäsenille.

Tehtäväsi:

✅ **Luo vene Skipperi Consoleen** ja varmista, että kaikki tiedot ovat oikein.

✅ **Tarkista vene huolellisesti** – tehtaan henkilökunta on ollut huolimaton, ja veneessä saattaa olla virheitä.

✅ **Havaitse ja korjaa mahdolliset virheet**, jotta vene on täysin käyttövalmis.

✅ **Julkaise vene varattavaksi asiakkaille**, mutta varmista, että ensimmäinen varaus voidaan tehdä vasta 14.4.2025.

Onnea matkaan! 🚤

Käyttäjätestauksen esimerkkiveneen tiedot

Tieto	Kuvaus
Veneen nimi	Jake
Veneen malli	Yamarin Cross 57BR
Veneen kotisatama	Soukan venekerho, Espoo
Veneen julkaisu varattavaksi	Välittömästi
Veneen ensimmäinen käyttöpäivä	14.4.2025

Käyttäjätestauksen testitapaukset

Testitapaus 1: Uuden veneen lisääminen	
Vaihe 1.1	Navigoi Skipperi Consolen kohtaan Boats ja paina + New Boat.
Testitapaus 2: Uuden sataman lisääminen	
Vaihe 2.1	Paina käyttöönottotyökalussa painiketta + New Location
Testitapaus 3: Veneen perustiedot	
Vaihe 3.1	Vaihda veneen nimi esimerkkiveneen tietojen mukaiseksi
Vaihe 3.2	Vaihda veneen malli esimerkkiveneen tietojen mukaiseksi
Vaihe 3.3	Vaihda veneen satama esimerkkiveneen tietojen mukaiseksi
Testitapaus 4: Q-näytön käyttöönotto	
Vaihe 4.1	Syötä Q-näytön sarjanumero järjestelmään
Vaihe 4.2	Kytke irronnut NMEA-kaapeli takaisin Q-näyttöön
Testitapaus 5: Lukkoboksin käyttöönotto	
Vaihe 5.1	Liitä oikea lukkoboksi veneeseen valitsemalla se alasvetovalikosta
Vaihe 5.2	Mittaa lukkoboksin paristotaso ja vaihda tarvittaessa paristot
Testitapaus 6: Veneen julkaisutiedot	
Vaihe 6.1	Julkaise vene asiakkaille tällä päivämäärällä
Vaihe 6.2	Sulje veneen kalenteri ajankohdalta tämä päivä – 14.4.2025
Testitapaus 7: Käyttöönoton viimeistely	
Vaihe 7.1	Merkitse käyttöönotto suoritetuksi

Käyttäjätestauksen tulokset

Vaihe	Onko käyttöliittymä onnistunut havainnollistamaan vian käyttäjälle?		Onko käyttäjä ymmärtänyt vian sekä osannut ryhtyä tarvittaviin toimenpiteisiin?		Saiko käyttäjä tehtävän suoritettua?	
Käyttäjä	1	2	1	2	1	2
Vaihe 1.1	K	K	K	K	K	K
Vaihe 2.1	K	K	K	K	K	K
Vaihe 3.1	K	K	K	K	K	K
Vaihe 3.2	K	K	K	K	K	K
Vaihe 3.3	E	K	K	K	K	K
Vaihe 4.1	K	K	K	K	K	K
Vaihe 4.2	K	K	K	K	K	K
Vaihe 5.1	K	K	K	K	K	K
Vaihe 5.2	K	K	K	K	K	K
Vaihe 6.1	K	K	K	K	K	K
Vaihe 6.2	K	K	K	K	K	K
Vaihe 7.1	K	K	K	K	K	K