Rick Wairimu

Integroitu vikakoodi- ja tiedostonhakusovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ajoneuvotekniikka

Insinöörityö

5.3.2025

Tiivistelmä

Tekijä:	Rick Wairimu
Otsikko:	Integroitu vikakoodi- ja tiedostonhakusovellus
Sivumäärä:	20 sivua
Aika:	5.3.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Ajoneuvotekniikka
Ammatillinen pääaine:	Ajoneuvosuunnittelu
Ohjaajat:	Jälkimarkkinointipäällikkö Pepe Juvamo Lehtori Sanna Heikkinen

---

Tässä oppinäytetyössä käsitellään vian- ja tiedostohaun sovelluksen kehitys ja mahdollinen käyttöönotto J-Tradingin huolto- ja jälkimarkkinointiosastolla. Aihe syntyi keskustelussa jälkimarkkinointipäällikön kanssa kesällä 2024, kun työn kirjoittaja keskusteli hänen kanssaan korjaamotoiminnan kehityksestä sovelluksen avulla.

Kuten monissa monimerkkikorjaamoissa, J-Tradingilla on ollut pitkään käytössä integroitu varaosa- ja huoltosovellus, mutta tiedostojen haussa ovat käytössä vielä perinteiset menetelmät. Projektin tavoitteena oli kehittää pohja integroidulle sovellukselle. Sovelluksen avulla tiedostojen ja vikakoodien haku olisi huomattavasti selkeämpää ja nopeampaa.

Työssä selvitettiin monimerkkikorjaamon jälkimarkkinatoiminnan keskeiset haasteet huoltoneuvonnan näkökulmasta ja miten niitä on mahdollista ratkoa integroidulla tiedostohakusovelluksella. Työssä dokumentoitiin sovelluksen suunnittelua, haluttujen toimintojen ohjelmointiratkaisuja sekä käytettyjä työmenetelmiä.

Projektin lopputuloksena syntyi kattava selvitys korjaamotoiminnan sovelluksen kehittämisestä sekä toiminnallinen demoversio, jossa on esitetty keskeiset toiminnot ja käyttölogiikka. Työssä on myös pohdittu huoltosovelluksien kehitystä tulevaisuudessa, erityisesti tekoälyn roolia diagnostiikassa.

Avainsanat: Sovelluskehitys, Jälkimarkkinointi, Vikakoodi, Python

---

Abstract

Author:	Rick Wairimu
Title:	Integrated Fault Code and File Retrieval System
Number of Pages:	20 pages
Date:	5 March 2025
Degree:	Bachelor of Engineering
Degree Programme:	Automotive Engineering
Professional Major:	Automotive Design
Supervisors:	Pepe Juvamo, Head of Aftersales
	Sanna Heikkinen, Senior Lecturer

The objective of this thesis is to develop an integrated fault code and file retrieval application for use at J-Trading's aftersales and maintenance department. The idea was born out of a conversation in summer 2024 with the head of aftersales at J-Trading, the conclusion of which was to develop an integrated application that would ease the burden on supervisors. As a result the department would become more efficient.

As with many multi-brand workshops, J-Trading already has an integrated spare part and maintenance app, but file retrieval is still done manually. The idea now was to develop an integrated app in which fault codes and maintenance instructions would be easy to search and retrieve.

First, the challenges of working in a multi-brand workshop from the perspective of a workshop manager are examined. Then the thesis introduces the benefits an integrated application provides in quickening and simplifying the information retrieval process.

This thesis documents the development process of the application, the coding solutions used in integrating the desired functions, and the working methods used during the development. Python is the programming language used for the application.

The result of this work is a comprehensive inquisition and blueprint for the development of an integrated file retrieval application for use in a workshop. This thesis also discusses the potential development of such applications, and particularly the potential use of AI in preventative maintenance. A working demo was also produced to present the key features such an app would contain.

Keywords: application development, aftersales, fault Codes, python

Sisällys

Lyhenteet

OOP: Oliokeskeinen ohjelmointi (Object-Oriented Programming, OOP) on ohjelmointitapa organisoida ja jäsentää objekteja, jonka avulla kehittäjät voivat luoda uudelleenkäytettäviä, modulaarisia ja skaalautuvia ohjelmistoja.

SQL: Tietokannan hallintajärjestelmä. Ohjelmisto tiedon tehokkaan hakemisen, säilyttämisen ja päivittämisen toteuttamiseksi

1 Johdanto

Tässä opinnäytetyössä tutkittiin sovelluksen kehittämistä J-Tradingin jälkimarkkinointiosastolle. Työssä käsitellään käytössä olevien sovelluksien toimintoja, työnjohtajien tehtävät ja tarpeet, sovelluksen kehittämistä ja ohjelmointia, sekä jälkimarkkinointiosaston tarve sovellukselle.

J-Trading on vuonna 1982 perustettu kiinteistö- ja ympäristöhoidon asiantuntija, joka edustaa alalla johtavia tuotemerkkejä katuharjakoneista piha- ja liikuntakalusteisiin. (Tuoteluettelo 2025.)

Yrityksessä jälkimarkkinointi on jaettu myynti- ja huolto-osastoihin ja niihin kuuluvat jälkimarkkinointipäällikkö, varaosamyyjät, työnjohtajat ja asentajat. Osastot tekevät tiivistä yhteistyötä asiakastyytyväisyyden varmistamiseksi ja yrityksen pitkät asiakassuhteet antavat kuvan onnistuneesta toimintamallista. Tiedonkulku on avainasemassa toimintamallin ylläpitämiseksi, minkä takia huoltoneuvontaprosessin kehittämistä ja selventämistä integroidun sovelluksen avulla on lähdetty selvittämään.

Työssä tutustutaan J-Tradingin tämänhetkisiin ohjelmiin ja käytäntöihin huoltoneuvonnassa, minkä avulla voidaan määrittää paremmin sovellukseen halutut toiminnot. Siinä perehdytään myös monimerkkikorjaamotoimintaan yleisesti, sekä eri valmistajien käytäntöihin ja siihen, miten sovelluksen avulla niitä voi parantaa.

Sovellus pitää sisällään tietokannan, hakukoneen ja käyttöliittymän. Sovellus tallentaa koneiden vikakoodit ja niihin liittyvät ohjeet yhteen tietokantaan, josta on mahdollista hakea jokaisen koneen vikakoodit ja niihin liitetyt ohjeet hakukoneen avulla. Haun tulokset tulostetaan käyttöliittymään, josta käyttäjä pääsee niihin käsiksi.

Opinnäytetyön lopullinen tavoite oli luoda selkeä pohja ja suunnitelma sovellukselle, jonka avulla sitä on helppo jatkokehittää ja ottaa käyttöön. Tarkoituksena

oli myös kartoittaa sovelluksen tarvetta osastolle ja pohtia korjaamohallinta-sovelluksien kehitystä tulevaisuudessa.

2 Ohjelman laadinta

Ohjelman laadinnassa lähdettiin liikkeelle jälkimarkkinointiosaston tarpeista ja erityisesti huoltoneuvonnan parissa toimivien työnjohtajien tarpeista. Suunnitelma laadittiin siitä, mille alustalle sovellus toteutetaan, mitkä olisivat sovelluksen vaatimukset ja millainen työnkulku ohjelmassa olisi eli millainen sitä olisi käyttää.

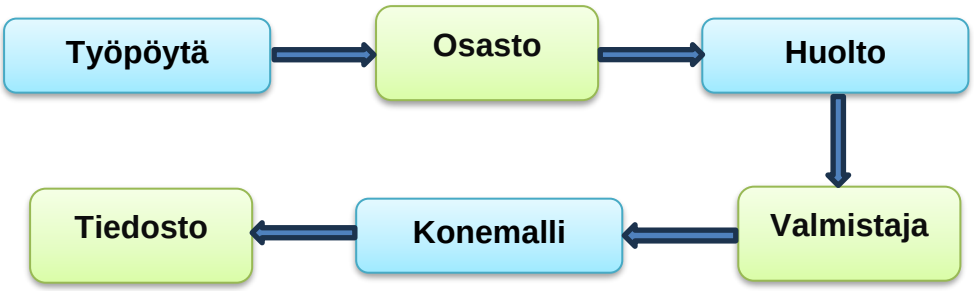
2.1 Käytössä olevat ohjelmat

Ohjelman suunnittelussa otettiin erityisesti huomioon käytössä olevat ohjelmat ja toimintaperiaatteet ja niistä lähdettiin rakentamaan sovelluksen pohja.

J-Tradingilla on jo käytössä verkkopohjainen sovellus varaosamyyntiin, jossa on oma tietokanta, josta käyttäjä voi hakea varaosia valmistajan koodilla tai varaosan nimellä. Samaan ohjelmaan on integroituna huoltovarausjärjestelmä, johon avataan jokaiselle huollolle oma tehtävä, johon myös myydään kaikki varaosat ja laskutetaan kaikki työt.

Huoltoneuvonnassa on käytössä monta eri ratkaisua lähdeaineistojen löytämiseen riippuen valmistajasta. Joillain valmistajalla on omat verkkosovellukset, joista löytää helposti vikakoodit ja niihin liittyvät ohjeet ja käytännöt. Esimerkiksi Belroboticsilla on käytössä Myrobot-niminen verkkosovellus, josta näkee reaaliajassa koneen sijainnin ja tiedot, sekä mahdolliset vikakoodit ja niiden selostukset. (Belrobotics Connected Services Manual 2022.)

J-Tradingilla asia on hoidettu niin, että vikakoodit on saatu valmistajalta joko Excel- tai PDF-tiedostoissa ja ohjeet ovat erikseen PDF-tiedostoissa. Nämä tiedostot on siten säilytetty Windows-kansioihin ja rajattu osaston, valmistajan, konetyypin ja konemallin mukaan.



Kuva 1. Nykyinen tiedonhankintakaavio

Kuten kuvasta 1 voidaan huomata, nykyisessä mallissa on kuusi eri vaihetta ennen kuin päästään haluttuun tiedostoon. Tämä kaavio voidaan tietysti lyhentää, kun kyseessä on yleinen konemalli, esimerkikisi pikalinkin avulla tai kopioimalla kansiota työpöydälle, mutta näin ei voi tehdä jokaiselle valmistajalle erikseen. Uudelle työnjohtajalle prosessin monimutkaisuus korostuu entisestään, koska hän ei ole vältämättä tietoinen siitä, mitkä tiedot löytyvät mistä, ja hän joutuu monta kertaa toistamaan prosessia.

2.2 Huoltoneuvonta monimerkkikorjaamossa

Huoltoneuvonnan tärkein tehtävä on antaa nopeaa, asiantuntevaa ja luotettavaa palvelua asiakkaalle. Työnkuvaukseen kuuluu asiakaspalvelua, teknistä osaamista, yhteistyötä ja myyntiä. Hyvä huoltoneuvoja on asiakaslähtöinen, rehellinen ja asiantunteva. Hän ottaa asiakkaita vastaan puhelimitse, sähköpostitse ja paikan päällä. J-Tradingilla huoltoneuvojat ovat avainasemassa, sillä he toimivat asiakkaan ja yrityksen yhteishenkilöinä ja heidän tehtäviensä lisäksi heidän täytyy edustaa J-Tradingin arvoja ja käytäntöjä ympäristöhoidon asiantuntijana. (Pelisääntökirja 2023.)

Huoltoneuvonta monimerkkikorjaamossa on huomattavasti monimutkaisempaa kuin merkkikorjaamossa, joissa valmistajalla on konkreettisia ja standardoituja

toimintamalleja. Monimerkkikorjaamossa huoltoneuvoja kohtaa monia ohjelmistoja ja käytäntöjä, joihin hänen pitää perehtyä.

Merkkikorjaamossa työnjohtajalla on yleensä sama polku tietojen löytämiseen ja tällöin työkuormaa on paljon vähemmän, sillä hän saa keskittyä ainoastaan ongelman ratkomiseen.

Monimerkkikorjaamossa tehtävä on paljon työläämpää. Jokaisella valmistajalla on omat toimintamenetelmät, mikä tarkoittaa sitä, että tiedot ovat monessa eri muodossa. Kun kyseessä on kiinteistöhuoltokoneiden maahantuoja, asia hankaloituu entisestään, sillä konemalleja on useita kymmeniä ja konetyyppejä myös kymmeniä. Työnjohtajan työnkulku on täten erittäin vaativaa, sillä hänen pitää muistaa jokaisen valmistajan toimintamalleja ja käytäntöjä, eli toisin sanoen mistä tiedot löytyvät ja missä muodossa tiedot ovat ja miten niihin pääsee käsiksi mahdollisimman nopeasti.

Kuva 2. Monimerkkikorjaamon huoltoprosessikaavio

Kuvan 2 prosessikaaviosta voidaan huomata, että työnjohtajalla on hyvin vaativa työnkulku, jota hänen täytyy seurata palvelulaadun ylläpitämiseksi. Huomio kiinnittyy erityisesti siihen, kuinka heti huoltoprosessin alussa hänen pitää nopeasti päästä tietoihin käsiksi asiakkaan palvelemiseksi. Usein asiakaskontaktin aikana pyydetään jo ohjeita tai huoltoneuvoa. Tämä on ehdottomasti yleisin syy asiakaskontaktiin ja kyseessä voi olla kysymys pikkukoneen bensatäyttömäärästä, johon löytyy helposti vastaus, katuharjakoneen taka-akselin asentotunnistimen vikakoodinselvitykseen, johon ei taas löydy niin helposti vastausta, ellei samanlaista tapausta ole tullut aikaisemmin vastaan. Selvää on, että huoltoprosessin monessa vaiheessa työnjohtaja tarvitsisi luotettavan ja nopean keinon

päästä näihin tietoihin käsiksi etenkin, jos kyseessä on vasta aloittanut työnjohtaja.

2.3 Vaatimukset

Vaatimukset määriteltiin kokemuspohjaisesti ja yhdessä jälkimarkkinointipäällikön kanssa. Määritellyssä otettiin huomioon käytössä olevat menetelmät, sovellukset ja tiedostomuodot, sekä ominaisuudet, joita haluttaisiin uuteen sovellukseen. Tärkeimpänä teemana oli käytettävyys. Sovellus haluttiin mahdollisimman käyttäjäystävälliseksi ja loogiseksi. Kartoitusvaiheessa päätettiin myös, että kehitettäisiin työpöytäsovellusta sen tehokkuuden ja turvallisuuden vuoksi.

Käyttäjäystävällinen sovellus on ulkonäöllisesti intuitiivinen ja helppo ymmärtää. Sen käyttölogiikka on sellainen, että käyttäjä ymmärtää sitä ilman mitään ohjeita. Tämä tarkoittaa sitä, että ulkoasu on mahdollisimman yksinkertainen. Toiminnot on järjestetty niin, että niiden tarkoitus ja logiikka on helposti ymmärrettävissä. (Käyttöliittymäsuunnittelu 2023.)

Työpöytäsovellus eroaa selainpohjaisesta sovelluksesta siinä, ettei se vaadi internetyhteyttä toimiakseen. Työpöytäsovellus on myös huomattavasti tehokkaampi, koska se pystyy käsittelemään suurempia tietomääriä ja sen tietokantojen ainoana rajoittavana tekijänä on tietokoneen muisti. Työpöytäsovelluksella on myös vähemmän kuluja, sillä sovelluksella ei ole ylläpitokustannuksia, koska sitä ei ylläpidetä pilvessä. Tieturvariskejä on huomattavasti vähemmän, koska sovellus ei ole yhteydessä internettiin. (Gomes 2024.)

2.4 Tavoite

Sovelluksen tavoitteena on helpottaa työnjohtajien kuormaa huoltoneuvonnassa ja selkeyttää heidän työnkulkuansa. Ohjelma suunnitellaan, jotta huoltoneuvonnan parissa työnjohtajilla olisi selkeämpi polku oikeisiin tiedostoihin. Tämän

ansiosta huoltoprosessi olisi selkeämpää ja työkuorma vähenisi. Sovelluksen avulla uusien työjohtajien perehdyttäminen olisi myös huomattavasti selkeämpää, sillä kaikki huoltoneuvontaan liittyvät ohjeet ja käytännöt löytyisivät samalta alustalta.

3 Ohjelman suunnittelu

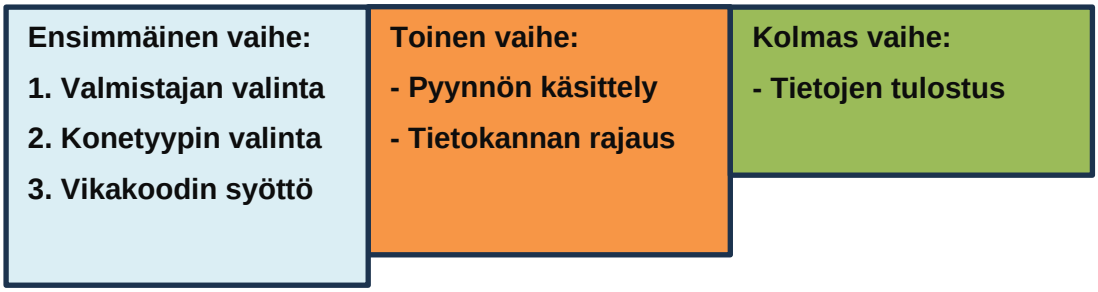
Sovelluksen suunnittelu lähti siitä, että kartoitettiin käyttäjän tarpeet ja määritettiin niistä selkeät toteutettavissa olevat toiminnot. Näiden avulla luotiin sovelluksen looginen rakenne ja työnkulku.

3.1 Sovelluksen työnkulku

Sovelluksen työnkulku on suunniteltu niin, että sovellusta olisi mahdollisimman helppoa käyttää. Ohjelmiston helppokäyttöisyyteen vaikuttavat sekä ohjelmiston työnkulku, käyttöliittymä, että ulkonäkö. Työnkululla tarkoitetaan ohjelmaan suunniteltuja käyttövaiheita, joiden kautta päästään loppupisteeseen eli tuloksen esittämiseen.

Työnkulku rakentuu kolmesta vaiheesta, jotka samalla määräävät sovelluksen käyttöliittymän suunnittelua. Ensimmäisessä vaiheessa käyttäjä valitsee konevalmistajan ja konemallin valikosta. Seuraavaksi hän syöttää haluamansa tiedon, kuten vikakoodin tai huoltokaavion.

Toisessa vaiheessa ohjelma käsittelee pyynnon. Koska haku on rajoitettu ja käsittelyssä olevaa tietomäärää on hyvin pieni, pyynnön käsittelyssä ei mene kauan. Lopuksi tulokset näytetään käyttäjälle. Mikäli kyseessä on vikakoodi, ohjelmaa esittää SQL-tietokantaan liitetyn vikakuvauksen ja mahdolliset korjausohjeet suoraan käyttöliittymään. Jos kyseessä on tiedosto, ohjelma antaa linkin ja painikkeen, jota painamalla tiedosto avautuu tietokoneen oletustiedostonlukijassa.



Kuva 3. Sovelluksen työnkulku.

Tietokannan rakenne määritetään jo tässä vaiheessa ja jokaiselle toiminnolle on määritettävä oma rakenne. Tämä onnistuu varsin helposti SQL:n avulla, sillä se tarjoaa selkeät tavat tietojen erotteluun taulujen avulla. Jokaiselle toiminnallisuudelle tehdään oma taulurakenne, jotka on liitetty viiteavaimilla. Ne mahdollistavat tietojen haun rajoittimilla. Esimerkiksi kaikki konemallit on yhdistetty viiteavaimella tiettyyn valmistajaan, minkä ansiosta niitä voi rajata valmistajan mukaan. Samoin vikakoodit ja tiedostot on yhdistetty viiteavaimiin, jotka yhdistävät niitä konemalleihin.

Näin on saatu looginen ja tehokas tietokanta. Sen ansiosta sovelluksen toimintaa on saatu tehostettua, koska jokaisella haulla ei käydä koko järjestelmän tietokantaa läpi vaan haut on rajoitettu hyvin pieneen alueeseen.

3.2 SQLite-tietokanta ja hakukone

Hakukone on sovelluksen toiminnan ytimessä ja sen tehtävänä on ottaa syötteen käyttäjältä ja hakea tietokannasta oikeat tiedot. Hakukoneen tehokkuus vaikuttaa suoraan sovelluksen käyttökelpoisuuteen ja tehokkuuteen. Hakukoneen on oltava nopea ja tarkka, jotta sovelluksella olisi merkittävä vaikutus huoltoneuvojien toimintaan.

Tietokanta on liitetty Pythoniin sisäänrakennetun SQLite3-kirjaston avulla. Kirjaston avulla SQL-tietokannat voidaan perustaa ja niitä voidaan hyödyntää Pythonissa vaivattomasti. Hakukone on optimoitu viiteavaimilla, jotka

mahdollistavat tietojen ja tulosten rajaamisen, vaikka kyseessä olisi suuria tietomääriä. Tässä yhteydessä hakukoneella tarkoitetaan käyttäjän syötteen käsittelyä ja yhdistämistä tietokantaan oikean tiedon löytämiseksi. SQL:ssä on sisäänrakennetut hakutoiminnot, joiden avulla perustetusta tietokannasta voidaan rajata ja hakea halutut tiedot ilman erillisen hakukoneen suunnittelua. (Berge 2025.)

Tietokannan perustamisessa otettiin huomioon J-Tradingilla käytössä olevat tietokannat ja tiedostomuodot. Tämä helpotti tietokannan ohjelmointia valtavasti, koska uuden tietokannan määrittely alusta lähtien olisi ollut hyvin työlästä. SQL:ssä on taulukkorakenne tarkoittaen, että ensin perustetaan taulukko. Sen jälkeen taulukon sarakkeisiin tulevat otsikot ja samalla tietotyypit, joita sinne laitetaan. Lopuksi tuodaan riveihin sopivat tietotyypit. Taulukoita voi olla niin monta kuin haluaa ja jokaiseen voi määrittää erilliset parametrit. Sallitut tietotyypit ovat teksti, numerot ja desimaalit mutta sinne voi myös säilyttää tiedostoja, kuvia ja linkkejä.

Koska ohjelman hakulogiikka oli jo määritelty, tietokannan perustaminen oli suoraviivaista. Aluksi luotiin erikseen taulukoita valmistajille, konemalleille ja dokumenteille. Valmistajien taulukkoon määritettiin valmistajan tunnistenumero, sekä nimi, koska tarkoituksena oli yhdistää valmistajat ja konemallit tällä tunnusnumerolla.

	id ▲	konemalli_id	nimi	polku
	Fil...	Filter	Filter	Filter
1	1	1	Ohje A1	C:/Users/Rick/Desktop/APP/Thesis/…
2	2	2	Ohje A2	C:/Users/Rick/Desktop/APP/Thesis/…
3	3	3	Ohje B1	C:/Users/Rick/Desktop/APP/Thesis/…

Kuva 4. SQLite-dokumenttitaulukon rakenne.

Konemalleille määritettiin samanlainen rakenne mutta sinne lisättiin myös konemallin tunnus, jonka avulla konemallien tiedot yhdistettäisiin. Lopuksi dokumenttitaulukon rakenteeseen määriteltiin valmistajan tunnus, konemallin tunnus, dokumentin nimi ja dokumenttipolku, kuten on esitetty kuva 4:ssa.

Seuraavaksi määritettiin tietojen tuominen taulukkoihin executemany-toiminnolla, yllä mainitulla järjestyksessä. Tässä vaiheessa tietokanta oli valmis käytettäväksi, mutta tähän kohtaan ei vielä määritelty tietojen hakua tietokannasta vaan se suunniteltiin suoraan käyttöliittymään.

```
c.execute("CREATE TABLE Valmistajat (id INTEGER PRIMARY KEY, nimi
TEXT)")
c.execute("CREATE TABLE Konemallit (id INTEGER PRIMARY KEY, valmis-
taja_id INTEGER, mallin_nimi TEXT)")
c.execute("CREATE TABLE Dokumentit (id INTEGER PRIMARY KEY,
konemalli_id INTEGER, nimi TEXT, polku TEXT)")
c.executemany("INSERT INTO Valmistajat (id, nimi) VALUES (?, ?)", [(3,
'Valmistaja C'), (4, 'Valmistaja D')])
c.executemany("INSERT INTO Konemallit (id, valmistaja_id, mallin_nimi)
VALUES (?, ?, ?)", [(4, 3, 'Malli A1'), (5, 4, 'Malli A2'), (6, 4,
'Malli B1')])
c.executemany("INSERT INTO Dokumentit (konemalli_id, nimi, polku) VA-
LUES (?, ?, ?)", [(1, 'Ohje A1',
'C:/Users/Rick/Desktop/APP/Thesis/sample.pdf'),(2, 'Ohje A2',
'C:/Users/Rick/Desktop/APP/Thesis/dummy.pdf'),(3, 'Ohje B1',
'C:/Users/Rick/Desktop/APP/Thesis/explain.pdf')])
```

Ohjelmakoodi 1. Tietokannan perustaminen ja tietojen tuonti SQLite-tietokantaan.

3.3 Käyttöliittymä ja ulkonäkö

Ohjelman käyttöliittymän suunnittelussa käytetiin Pythoniin sisäänrakennettua Tkinter-kirjastoa. Pythonissa ulkoasun suunnittelu on tunnetusti hyvin rajoitettua, mutta Tkinterin avulla on saatu hieman paremman näköisiä käyttöliittymiä vuosien varrella. Kirjaston käyttö on hyvin intuitiivista, sillä se on pitkään ollut suosituin käyttöliittymän suunnittelutyökalu Pythonissa. Kirjasto valittiin myös sen takia, että siihen löytyy eniten ohjeita ja materiaalia netissä.

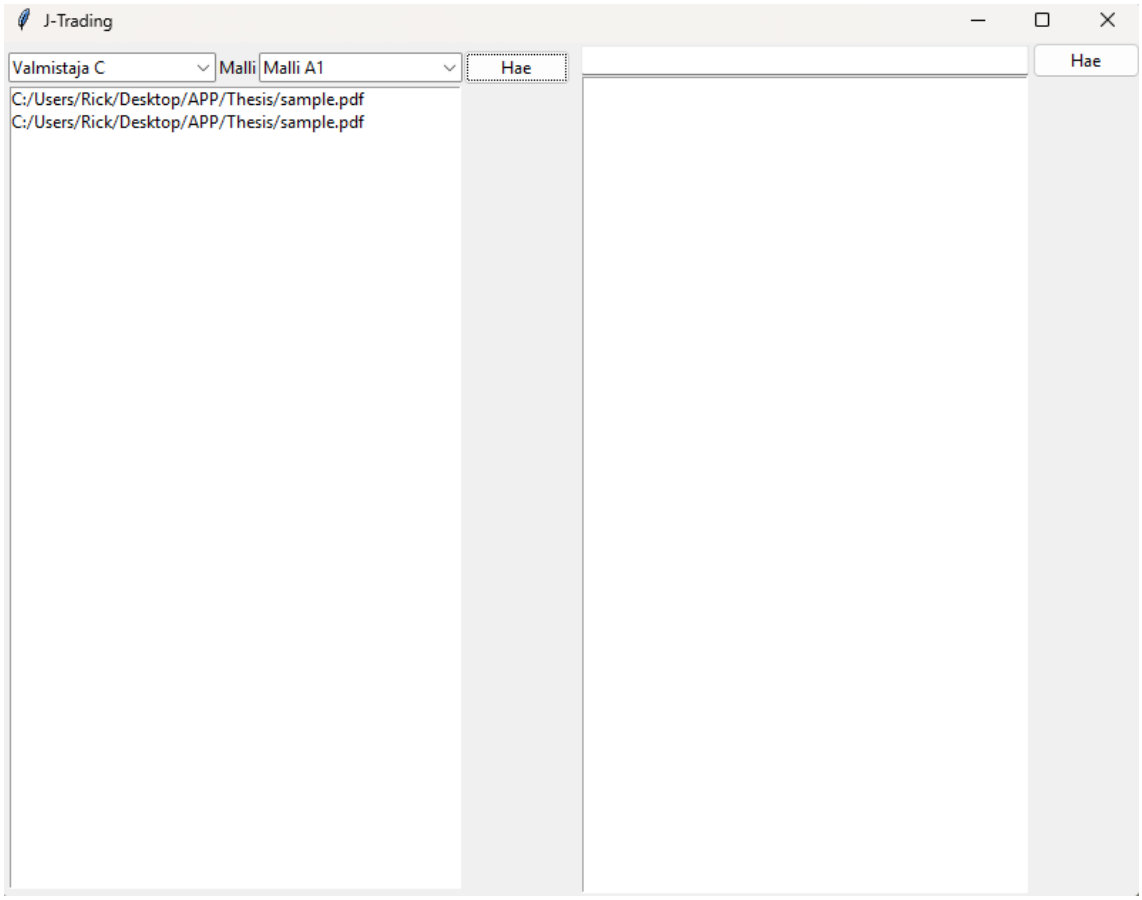
Käyttöliittymä on suunniteltu mahdollisimman yksinkertaiseksi ja selkeäksi seuraten yllä määriteltyä työnkulkua. Käyttöliittymän toiminnallisuus pohjautuu työnkulkuun, jossa ohjelman käyttö etenee loogisesti. Käyttöliittymän suunnittelussa on vältetty turhia valikoita tai kuvioita.

Ohjelmoinnissa lähdettiin liikkeelle siitä, että määritettiin pääikkuna, jonka päälle kaikki rakennettaisiin. Tämän jälkeen määritettiin kaksi kehystä, joiden päälle yllä mainitut toiminnot tulisivat näkyviin. Ajatuksena tässä oli se, että käyttöliittymä oli tulevaisuudessa helppo päivittää, kun kehykset ovat toisistaan riippumattomia.

Vasempaan kehykseen sijoitettiin kaksi alasvetovalikkoa, joilla valitaan valmistaja sekä konemalli. Näiden lisäksi lisättiin hakunapit, joita painamalla ohjelma suorittaa haun. Elementit löytyivät suoraan kirjastosta. Niiden paikoittaminen onnistui määrittämällä ensin rivi, sarake ja kehys, johon ne tulisivat näkyviin.

Sovelluksen ulkoasun vasempaan kehykseen sijoitettiin valmistajan valinta, konemallin valinta ja dokumenttipolun tulostusalue. Valmistajan ja mallin valinta on suunniteltu niin, että siellä on kaksi valintalistaa yhdistettynä tietokantaan, jotka ovat toisistaan riippuvaisia, eli mallin valintalista näyttää vaihtoehtoja vasta silloin, kun valmistaja on valittuna. Kun molemmat ovat valittuna, hakupainike ottaa syötteen vastaan ja lähettää käsittelypyynnön tietokantaan, jossa se ensin yhdistää ne viiteavaimeen, jonka jälkeen se hakee dokumenttitaulukosta kaikki dokumenttipolut, joissa on sama viiteavain. Tällä saadaan varmistettua, ettei missään vaiheessa ohjelma ole prosessoimassa koko tietokantaa, koska tämä

olisi hyvin resurssi-intensiivinen ja sovelluksen tehokkuus olisi huomattavasti huonompi.



Kuva 5. Käyttöliittymän pääikkuna.

Tuloksen esitys nähdään kuvassa 5, jossa on haettu esimerkki konemallin kaikki tiedostot ja linkkiä painamalla käyttäjän järjestelmä avaa tiedostot oletus-lukusovelluksessa.

```
def imp(self, event=None):
valmistaja = self.valm_entry.get()
self.conn = sqlite3.connect('THESIS4.db')
self.c = self.conn.cursor()
self.c.execute("SELECT id FROM Valmistajat WHERE nimi = ?", (valmis-
taja,))
```

Ohjelmakoodi 2. Esimerkki SQlite-kyselystä.

Ohjelmakoodissa 2 esitetään esimerkki SQlite-kyselystä, jonka avulla on yhdistetty käyttäjän valitsema valmistaja tietokannanvalmistajatunnisteeseen.

Haun jälkeen ohjelma tulostaa vasempaan tulostusalueeseen dokumenttipolut, joita painamalla käyttäjän järjestelmä käynnistää oletuslukusovelluksen, jossa tietoihin pääsee käsiksi eli on kokonaan vältetty kansioiden selaaminen.

Oikealla puolella on taas hakukenttä ja tulostusalue. Hakukenttä on ohjelmoitu ottamaan käyttäjän vikakoodisyötteen. Käyttäjän valitseman koneen perusteellä ohjelma käsittelee ja lukee ennalta määritettyä vikakooditiedostoa. Tämän jälkeen käyttäjän syöte haetaan tiedostosta, ja jos tiedosto sisältää samankaltaista tietoa, se tulostetaan tulostusalueeseen.

Oikeaan kehykseen voi liittää esimerkiksi vikakooditiedoston, jota ohjelma lukisi. Sen avulla käyttäjä pääsisi heti hakemaan haluttua vikakoodia joutumatta itse etsimään niitä yllä mainituista tiedostoista. Tämän toiminnon suunnitellussa on pohdittu sovelluksen käytettävyyttä ja tehokkuutta. Tämä on toinen niistä keskeisistä toiminnoista, joita voi tulevaisuudessa kehittää paremmaksi ja tehokkaammaksi.

3.4 Ohjelmointi

Tässä vaiheessa alettiin hyödyntämään Pythonin OOP-ohjelmointia eli oliokeskeistä ohjelmointia. Tämä ohjelmointitapa perustuu modulaarisuuteen, eli pyritään mahdollisimman vähiin toistoihin koodissa. Siinä mallinnetaan ne ominaisuudet, joita tullaan käyttämään usein kapseloivissa luokissa. Toisin sanoen sen sijaan, että määrittäisin tiettyä toimintoa aina uudestaan, kun siihen on tarvetta, OOP:ssä mallinnetaan ja segmentoidaan kaikki toiminnallisuudet funktioihin ja säilytetään ne luokissa. Niitä voidaan hyödyntää myöhemmin kutsumalla määritettyä luokkaa. Tämän avulla koodista tulee myös selkeämpi. Kaikki toiminnot ovat siististi luokkien sisällä ja muuttamalla luokan parametrit saadaan päivitettyä helposti kaikki kohdat, joissa kyseistä luokkaa on hyödynnetty. (Oliopohjainen-ohjelmointi-oop 2025.)

Ohjelma on ensin rakennettu paloista, eli kaikki toiminnot on kehitetty toisistaan erillään ennen niiden yhdistämistä. Suurimpana syynä tähän oli koodin ja erityisesti toimintojen sisäistäminen ennen integroimista. Kun toiminnot yhdistettäisiin ja virhetilanteet syttyisivät, olisi tiedossa heti, missä kohtaa on mennyt sekaisin.

Ohjelman integroinnissa kaikki on tehty OOP:n mukaan. Kaikki funktiot on siirretty luokkiin, joissa niitä on ollut mahdollista käyttää uudelleen. Isoin muutos oli kuitenkin se, että OOP:n takia ohjelmointi lähti liikkeelle käyttöliittymästä ja sen päälle rakennettiin kaikki toiminnot. Toisin sanoen ohjelma on puhtaasti käyttöliittymän instanssina tarkoittaen, että kaikki toiminnot on rakennettu tKinterin rajoitukset huomioiden ja mukautuen niihin.

Ohjelman logiikka lähti tKinterin pääikkunan määrittelystä. Ensin kutsuttiin kaikki kirjastot, joita tarvittaisiin sovellusten ominaisuuksien pyörittämisessä. Sen jälkeen on määritetty sovelluksen pääluokka App(), johon sisältyvät sovelluksen kehykset ja juurikonfiguraatio. Toisin sanoen tässä luokassa määriteltiin, millainen olisi sovelluksen taustarakenne, jonka päälle kaikki hakukentät ja napit rakennettaisiin.

```
class App(tk.Tk): # Application is a tkinter instance
    def __init__(self): # sice the class is an instance of tk / self
refers to tk
        super().__init__()

        self.title("J-Trading")
        self.geometry("800x600")

        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=3)
        self.rowconfigure(0, weight=1)

        frame = db_search(self)#container for widgets
        frame.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)
        frame2 = db_print(self)#container for widgets
        frame2.grid(row=0, column=1, sticky="nsew", padx=2, pady=2)
```

Ohjelmakoodi 3. App-luokan määrittäminen

Seuraavaksi luotiin ohjelmakoodi 3:n mukaan luokka, joka sisälsi kaikki vasemman kehyksen toiminnot mukaan lukien kaikki tietokannan lukufunktiot. Tässä OOP:n edut nousivat hyvin selkeästi esille. Sen sijaan, että kaikki funktiot olisi määritetty erikseen käyttöliittymästä ja sitten kutsuttu, niitä määritettiin ja integroitiin yhteen luokkaan.

Tämä mahdollisti sen, että vaikka joku muuttuja olisi paikallisesti funktion sisällä, se oli käytettävissä koko ohjelmassa, vaikka se olisi funktion tai luokan ulkopuolella. Ilman tätä joustavuutta oikean kehykseen riippuvuus vasemman kehyksen syötteeseen olisi ollut hyvin hankalaa toteuttaa ilman funktioiden uudelleen määrittelyä, mikä olisi johtanut toistuvaan koodiin.

```
class db_search(ttk.Frame): # Class with tk Frame as it's instance
    def __init__(self, parent):
        super().__init__(parent)
        import sqlite3
        conn = sqlite3.connect("THESIS4.db")
        c = conn.cursor()

        self.columnconfigure(0, weight=1) # allows self to expand
        self.rowconfigure(1, weight=1)


        self.text_box = tk.Listbox(self, width=50)
        self.text_box.grid(row=1, column=0, columnspan=3,
sticky="nsew")
        self.text_box.bind('<Double-1>', self.avaa_tiedosto)

        self.valm_entry = ttk.Combobox(self, state="readonly")
        self.valm_entry.grid(row=0, column=0, sticky="ew", col-
umnspan=1)
        self.valm_entry['values'] = [row[0] for row in c.execute("SE-
LECT nimi FROM Valmistajat").fetchall()]
        self.valm_entry.bind("<<ComboboxSelected>>", self.imp)
```

Ohjelmakoodi 4. Luokka, jossa rakennettu tkinterin päälle toiminnot.

Ohjelmakoodi 4:ssä on esitetty vasemman kehyksen asettelua ja siinä näkyy myös, miten funktiot on liitetty tKinterin toimintoihin. Integroinnissa tuli vastaan hyvin paljon rajoitteita etenkin sen takia, että ohjelma on rakennettu tKinter-käyttöliittymäkirjaston päälle.

Vaikka kirjasto on iän ansiosta hyvin vakaa ja siihen löytyy paljon apumateriaalia, sen kanssa oli hyvin haastavaa toimia etenkin oikeanpuoleisen kehyksen integroinnissa vasemman puolen kanssa. Tämän takia toiminnot ovat jääneet hyvin vähäisiksi.

Modulaarisuuden takia ohjelma on helppo päivittää. Siihen mahtuu vielä monta toimintoa, joita tämän opinnäytetyön aikana ei olisi mahdollista toteuttaa.

3.5 Toiminnallisuus

Toiminnallisuus on ollut tämän työn tärkeimpiä teemoja ja se on jokaisessa suunnittelun vaiheessa huomioitu kattavasti. Ohjelmoinnissa on huomioitu

kaikki mahdolliset virhetilanteet, joita sovelluksen käytön aikana voi syntyä. Sen avulla sovelluksesta on tehty mahdollisimman käyttökelpoinen.

Käyttöliittymässä on varmistettu toiminnallisuutta siten, että kaikki toiminnot ovat toisistaan riippuvaisia. Se tarkoittaa, että ohjelma ei lähde väärästä painalluksesta pyörittämään mitään taustalla.

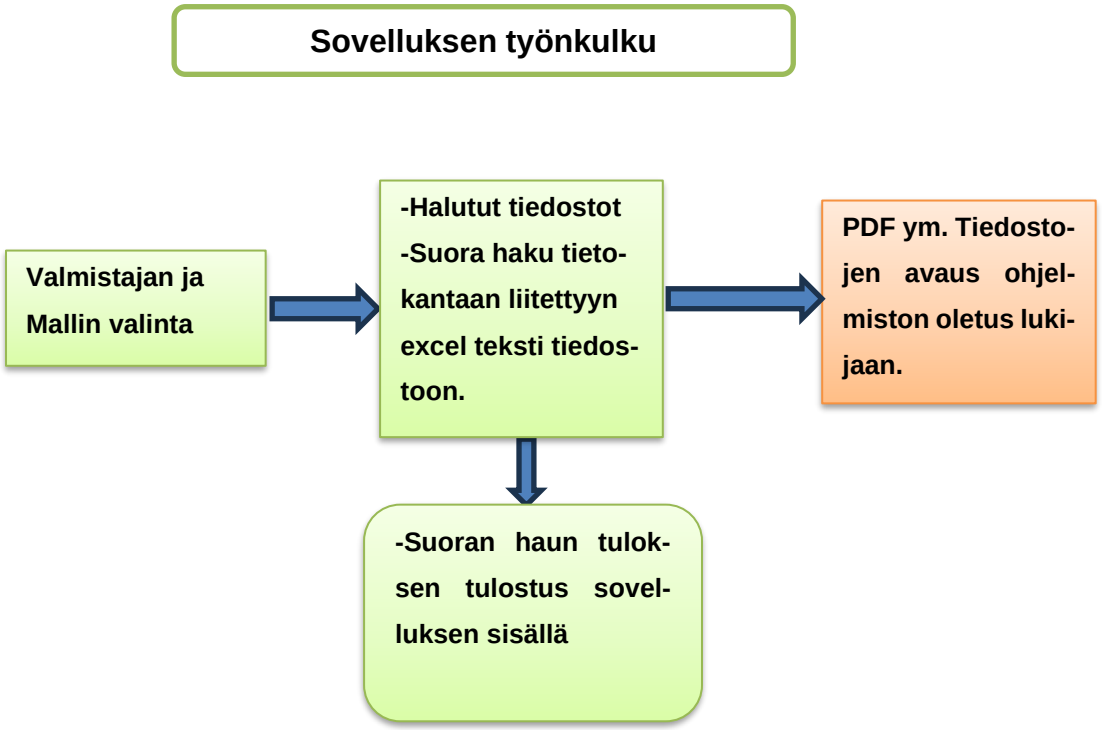
Toiminnallisuutta on varmistettu myös kattavalla tietokantasuunnittelulla, joka on tehty myös tarvittaessa muokattavaksi. Suurin toiminnallisuutta edistävä tekijä on kuitenkin ohjelman modulaarisuus, joka on saatu aikaan OOP:llä. Luokkien ja olioiden avulla ohjelman muokkaaminen ja päivittäminen onnistuu ongelmitta ja se on koettu suunnittelussa monta kertaa hyödylliseksi.

4 Käyttöönotto ja toiminnan kehitys

Ohjelman yksinkertaisuuden vuoksi käyttöönotto olisi hyvin helppoa. Ohjelman käyttöön ei tarvitsisi ollenkaan perehdytystä. Se on myös ohjelmoitu niin, että virhetilanteita ei pääse käytön aikana syntymään.

4.1 Käyttöönotto huoltoneuvonnassa

Huoltoneuvonnassa sovelluksella olisi monipuolisesti käyttöä ja sen voisi uskoa helpottavan merkittävästi huoltoneuvonnan työnkulkua. Sovellus nykymuodossa lyhentää tiedonhankintaan kuuluvaa aikaa huomattavasti, sillä se korvaa jo useita askeleita nykyisessä tiedonhankintaprosessissa sen ominaisuuksilla.



Kuva 6. Sovelluksen työnkulku.

Kuvassa 6 on esitetty sovelluksen työnkulku, jossa havaitaan, että tiedostonhakupolussa on vain 3 vaihetta. Verrattuna kuvassa 1 esitettyyn tiedonhankintakaavioon sovelluksen avulla tiedonhankinta on huomattavasti yksinkertaisempaa ja nopeampaa.

Merkittävä rooli tässä on ohjelman suunnitelulla, sillä se on mukautettu ja suunniteltu valmiina olevien tietokantojen ympärille. Vaikka kyseessä on uusi ohjelma, sen käyttöönotto olisi melko vaivatonta, sillä tietokantojen perustamiseen ei kuluisi liikaa resursseja.

4.2 Toiminnan kehitys

Sovellus tarjoaa tässä vaiheessa mielenkiintoisen perspektiivin siitä, miten työnjohtajien toimintaa voidaan kehittää jatkossa tehokkaammaksi. On yllättävää,

kuinka näin yksinkertaisella työkalulla voi tehostaa tiedonhakua näin merkittävästi.

Etenkin uusille työnjohtajille ja varaosamyyjille tällaisen työkalun saatavuus työuran alussa olisi iso kuormankantaja. Varsinkin, kun ensimmäisinä kuukausina uudessa työtehtävässä on paljon asioita muistettavana.

Ohjelma soveltuisi erinomaisesti perehdytystyökaluksi. Jos sovellusta vielä jalostettaisiin, siitä tulisi loistava tiedonhankintatyökalu jälkimarkkinointikäyttöön.

4.3 Sovelluksen jatkokehitys

Sovelluksessa on vielä monia puolia ja ominaisuuksia, joita voi lähteä tulevaisuudessa kehittämään. Yksi näistä ominaisuuksista, josta jo kartoitusvaiheessa keskusteltiin toimeksiantajan kanssa, on koneoppimallin integroiminen tällaiseen työkaluun jälkimarkkinoinnin käyttöön.

Monilla autovalmistajilla on jo käytössä tekoälyavusteisia ja ennakoivia huoltojärjestelmiä ja niiden ennustetaan lisääntyvään tulevaisuudessa. (Heikkinen 2024.)

Tämän sovelluksen jatkokehitystä varten on mietitty koneoppimallin integroiminen, sillä tavalla, että sen koulutusmateriaalina olisivat valmistajan ohjeet ja sellaiset materiaalit, joihin on olemassa selkeästi kyllä tai ei -vastauksia. Koska kyseinen mallin koulutusmateriaali ei pohjautuisi mielipiteisiin, tällaisesta mallista saisi hyvin luotettavan.

Tämän kaltaisiin työkaluihin on kuitenkin vielä pitkä matka, mutta koneoppimallit ovat jo ilmaiseksi saatavilla esimerkiksi GitHubilla. Joten jos osaamista on, tällaisen mallin kehittäminen olisi mahdollista.

5 Yhteenveto

Tässä opinnäytetyössä suunniteltiin ja luotiin toimivaa sovellusta, jolla osoitettiin tällaisen työkalun tarve huoltoneuvonnan ja jälkimarkkinoinnin parissa toimivien keskuudessa. Sovelluksen tarkoitus oli luoda selkeää pohja integroidun tiedonhankinnan sovelluksen kehittämiseen, minkä avulla huoltoneuvojien ja varaosamyyjien työnkulkua saataisiin selkeämmäksi ja tehokkaammaksi.

Opinnäytetyössä on käsitelty huoltoneuvonnan parissa toimivien huoltoneuvojien vaatimuksia ja haasteita. Työssä on tutustuttu myös huoltoprosesseihin ja miten niitä voi selkeyttää aputyökalun avulla.

Opinnäytetyön lähtökohtana olivat omat kokemukset uutena työnjohtajana ja miten kokemusten avulla saisi kehitettyä työkalua, joka tarjoaisi käytännön ratkaisuja työnelämään. Työn tuloksena syntyi tehokas tiedonhankintatyökalun pohja, joka jatkokehittämällä olisi hyvin hyödyllinen huoltoneuvonnan parissa lisätyökaluna etenkin uusien huoltoneuvojien käsissä. Vaikka kyseessä on vielä hyvin yksinkertainen sovellus, jatkokehittämällä tästä saisi hyvin kattava työkalu jopa pidempään alalla toimineille ammattilaisille.

Lähteet

Belrobotics Connected Services Manual. 2022. Verkkoaineisto. Belrobotics. <https://www.belrobotics.com/en/>. Luettu 5.4.2025.

Gomes, Jose. 2024. Web Apps vs desktop Apps: Understanding the differences. Verkkoaineisto. Koombea. < https://www.koombea.com/blog/web-apps-vs-desktop-apps/>. Luettu 7.4.2025

Heikkinen, Hannes. 2024. Tekoälyn rooli autokorjaamolla. Opinnäytetyö. Tampereen Ammattikorkeakoulu. Theseus-tietokanta. Luettu 23.4.2025.

Käyttöliittymäsuunnittelu. 2023. Verkkoaineisto. Haltu. <https://www.haltu.fi/blogi/kayttoliittymasuunnittelu>. Päivitetty 14.3.2023. Luettu 5.4.2025.

Oliokeskeinen-ohjelmointi-oop. Verkkoainesto. Thecodest. <https://thecodest.co/fi/sanakirja/oliopohjainen-ohjelmointi-oop/>. Luettu 7.4.2025.

Pelisääntökirja. 2023. Yrityksen sisäinen materiaali. J-Trading Oy.

Tuoteluettelo 2025. 2025. Verkkoaineisto. J-Trading Oy. < https://issuu.com/j-trading/docs/j-trading_tuoteluettelo_2025>. Luettu 30.4.2025.