TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Tartu kolledž

VÄIKELENNUKITE HOOLDUSTARKVARA ARENDAMINE

DEVELOPMENT OF SMALL AIRCRAFT MAINTENANCE SOFTWARE

BAKALAUREUSETÖÖ

Üliõpilane: Kaarel Vomm

Üliõpilaskood 178441EDTR

Juhendaja: Taavi Kase

Tartu 2023

(Tiitellehe pöördel)

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud.
Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

05.06.2023

Autor: Kaarel Vomm
/ allkirjastatud digitaalselt /

Töö vastab bakalaureusetöö esitatud nõuetele

05.06.2023

Juhendaja: Taavi Kase
/ allkirjastatud digitaalselt /

Kaitsmisele lubatud

05.06.2023

Kaitsmiskomisjoni esimees: Aime Ruus
/ allkirjastatud digitaalselt /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina Kaarel Vomm (sünnikuupäev: 05.07.1995)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

"Väikelennukite hooldustarkvara arendamine", mille juhendaja on Taavi Kase

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

---

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil.*

/ allkirjastatud digitaalselt /

05.06.2023

TalTech Tartu kolledž

LÕPUTÖÖ ÜLESANNE

Üliõpilane: Kaarel Vomm 178441EDTR

Õppekava, peaeriala: EDTR17/17 - Telemaatika ja arukad süsteemid

Juhendaja(d): Insener, Taavi Kase, 6204808

Lõputöö teema:

Väikelennukite hooldustarkvara arendamine

Development of Small Aircraft Maintenance Software

Lõputöö põhieesmärgid:

1. Arendada välja täis-pinu veebirakendus, mis võimaldab väikelennuki hooldusprotsesside haldamist.

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Väikeste lennukite hooldusprotsesside analüüsimine ja olemasolevate lahenduste uurimine	15.02.2023
2.	Rakenduse arhitektuuri kavandamine ja tehnoloogiate valimine	01.03.2023
3.	Kasutajaliidese arendamine Vue.js 3 ja PrimeVue abil	15.03.2023
4.	Serverrakenduse arendamine Spring Boot ja Java abil	30.03.2023
5.	Andmebaasi integreerimine ja andmemudelite loomine	15.04.2023
6.	Rakenduse juurutamine ja kasutajale avalikuks tegemine	10.05.023
7.	Tulemuste analüüsimine, väljakutsete ja õppetundide esitamine ning lõputöö kirjutamine	22.05.2023

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** 05.06.2023

Üliõpilane: Kaarel Vomm / allkirjastatud digitaalselt / 05.06.2023

Juhendaja: Taavi Kase / allkirjastatud digitaalselt / 05.06.2023

Programmijuht: Aime Ruus / allkirjastatud digitaalselt / 05.06.2023

Kinnise kaitsmise ja/või lõputöö avalikustamise piirangu tingimused formuleeritakse pöördel

SISUKORD

EESSÕNA 7
Lühendite ja tähiste loetelu 8
SISSEJUHATUS 9
1 VÄIKELENNUKITE HOOLDUSE VÄLJAKUTSED 10
1.1 Rutiinsed ja mitterutiinsed hooldusprotsessid 10
1.2 Euroopa Liidu nõuded 11
1.3 Traditsiooniline paberipõhine dokumenteerimine 12
1.4 Tänaste tarkvaralahenduste väljakutsed 12
1.5 Lõputöö eesmärgi püstitamine 13
2 RAKENDUSE FUNKTSIONAALSUS 14
2.1 Peamised funktsionaalsused 14
2.1.1 Lennuki hoolduste graafiku vaade 14
2.1.2 Teostatud hooldusprotsesside vaade ja haldamine 15
2.1.3 Lennukite vaade ja haldamine 16
2.1.4 Lendude vaade ja logimine 17
2.2 Toetavad funktsionaalsused 18
2.2.1 Kasutajate haldamine ja rollid 18
2.2.2 Lennuki mallid 19
2.2.3 Hooldusprotsessi mallid 19
3 RAKENDUSE ARHITEKTUUR 21
3.1 MVC (Mudel-Vaade-Kontroller) 21
3.2 Mikroteenuste arhitektuur 21
4 KASUTAJALIIDESE LOOMINE 23
4.1 Kasutajaliidese kavandamine 23
4.2 Vue.js 3 ja PrimeVue kasutamine 23
4.2.1 Vue.js 3 23
4.2.2 PrimeVue 23
4.2.3 Rakenduse komponentide loomine 24
4.3 Tulemused ja õppetunnid 25
5 SERVERRAKENDUSE LOOMINE 26
5.1 Mikroteenused 26
5.1.1 Lennukite teenus 27
5.1.2 Hooldusprotsesside teenus 27
5.1.3 Kasutajate teenus 27

5.2 API disain 27
5.2.1 API lõpp-punktid 27
5.3 API Gateway teenus 28
5.4 Mikroteenuste omavaheline suhtlemine 29
5.5 Turvalisus ja autentimine 30
5.6 Tulemused ja õppetunnid 31
6 ANDMEMUDELI LOOMINE 32
6.1 Andmemudelite loomine 32
6.2 Andmebaasi valik 32
6.3 Andmemudelite integreerimine PostgreSQL andmebaasiga 33
6.3.1 Flyway Java teek 33
6.3.2 Loodud SQL 34
6.4 Andmete indekseerimine 34
6.5 Tulemused ja õppetunnid 35
7 RAKENDUSE JUURUTAMINE KESKKONDA 36
7.1 Heroku tutvustus ja valiku põhjendus 36
7.2 Arenduskeskkondade loomine 36
7.3 Konfiguratsioon 37
7.4 Live keskkonda juurutamine 38
8 TULEVASTE ARENDUSTE PLAANID 39
8.1 Teavituste saatmine 39
8.2 Parema kasutajakogemuse loomine 39
8.3 Tehnilised parendused ja jõudluse suurendamine 40
KOKKUVÕTE 41
SUMMARY 42
KASUTATUD KIRJANDUSE LOETELU 43

EESSÕNA

Käesoleva lõputöö eesmärk on arendada välja täis-pinu veebirakendus, mis võimaldab väikelennuki hooldusprotsesside haldamist.

Autor kasutab selle saavutamiseks modernseid tarkvara arenduse praktikaid ja tehnoloogiaid. Lõputöö idee ja teema sündis autori isiklikust huvist lennunduse vastu ja lähedastest tuttavatest lennundussektorist.

Lõputöö on eesti keeles ja sisaldab teksti 44 leheküljel, 8 peatükki ja 15 joonist. Märksõnad: väikelennuk, hooldusprotsess, lennundus, tarkvara, mikroteenused.

Lühendite ja tähiste loetelu

API - Programmiliides (inglise keeles *Application Programming Interface*) [1]
Git – Versioonihaldustarkvara [2]
GitHub – Veebimajautusteenus, kastutades Git tarkvara [3]
GRPC – Suhtlusprotokoll hajusate süsteemide vahelisteks päringuteks (inglise keeles *Google Remote Procedure Call*) [4]
HTTP - Hüperteksti edastusprotokoll (inglise keeles *Hypertext Transfer Protocol)* [5]
JWT – Veebimärgi standard (inglise keeles *JSON web token*) [6]
PaaS – Pilvandmetöötluse pakkumine platvormina rentimisena (inglise keeles *Platform as Service*) [7]
REST – Tarkvaraarhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid (inglise keeles *Representational state transfer*) [8]
SQL – Struktuurpäringukeel (inglise keeles *Structured Query Language*) [9]
Token – Veebimärk, võimaldab valideerida veebpäringu saatjat [6]
URL – Internetiaadress ehk üldine infoallika asukohamääraja (inglise keeles *uniform resource locator*) [10]

SISSEJUHATUS

Tänapäeva lennundussektoris on hoolduse efektiivne majandamine muutunud üha olulisemaks, sest konkurents on tihe, aga uute tehnoloogiate kasutusele võtmisega on võimalik saavutada tihedas konkurentsis eelis. Suured ettevõtted mõistavad seda ja arendavad seetõttu keerulisi ja võimsaid süsteeme, mis vastavad just nende vajadustele. Kuid väiksemate lennukite hooldus vajab süsteemi, mis võimaldaks optimeerida hooldusprotsesse ja säilitada ajalugu, samal ajal olles kättesaadava hinnaga ja piisavalt lihtne kasutada ka väikelennuki omanikule.

Käesoleva bakalaureusetöö eesmärk on arendada täisfunktsionaalne tarkvara, mis aitab hõlbustada väikeste lennukite hooldusprotsesside haldamist. Töö uurib väikelennukite hoolduse tarkvara arendamist, kasutades kaasaegseid arenduspraktikaid, keskendudes seejuures erinevatele aspektidele, nagu valitud tehnoloogiad, metoodikad ja protsessid. Tarkvara eesmärk on pakkuda lahendust väikeste lennukite hoolduse haldamisele ja optimeerimisele, analüüsides selles kontekstis tarkvara arendusega seotud probleeme ja protsesse.

Lõputöö käigus arendatakse täisfunktsionaalne täis-pinu rakendus, mis on juurutatud Heroku pilveteenusesse. Kasutajaliides (frontend) on ehitatud Vue.js 3 ja PrimeVuega, mis on kaasaegsed ja kergesti kasutatavad tehnoloogiad. Serverrakendus (backend) on ehitatud Spring Boot ja Javaga, mis on väga laialdaselt levinud. Andmebaasiks on valitud PostgreSQL, mis tagab stabiilsuse ja töökindluse.

Käesolev lõputöö koosneb kaheksast osast. Esimene peatükk tutvustab väljakutseid väikelennukite hooldamise protsessides. Teises peatükis antakse ülevaade rakenduse funktsionaalsusest. Kolmandas peatükis kirjeldatakse rakenduse üldist arhitektuuri. Neljandas peatükis kirjeldatakse kasutajaliidese loomist. Viiendas peatükis kirjeldatakse serverrakenduse osa arendamise protsessi, andes ülevaate Spring raamistikuga loodud API-st ja andmebaasi mudelist. Kuuendas peatükis käsitletakse andmemudeli loomist. Seitsmes peatükk räägib rakenduse Live keskkonda juurutamisest. Kaheksandas kirjeldatakse tuleviku arenduste plaane.

1 VÄIKELENNUKITE HOOLDUSE VÄLJAKUTSED

Traditsiooniliselt on väikelennukite hooldusprotsessid toimunud paberipõhise dokumentatsiooni abil, kus olulised hooldusandmed ja tegevused salvestatakse füüsilistele lehtedele. Kuid digiajastul, kus tarkvara on laialdaselt kasutusel erinevates valdkondades, tekib küsimus, kas ka väikelennukite hooldamisel võiks kasutada kaasaegset digitaalset lahendust, mis oleks samal ajal lihtsasti kasutatav ja taskukohane.

1.1 Rutiinsed ja mitterutiinsed hooldusprotsessid

Lennuki hooldusprotsessid jagatakse lihtsuse mõttes kaheks, rutiinsed ja mitterutiinsed protsessid. [11]

Rutiinsed hooldusprotsessid hõlmavad tsiviillennunduses üldiselt hoolduskontrolle või hooldusplokke, mis koosnevad hooldusülesannete pakettidest, mis tuleb lennukil teatud aja või kasutuse järel läbi viia. [11] Hoolduspaketid koostatakse hooldusülesannete mugavaks jagamiseks väiksemateks osadeks, et minimeerida lennuki hoolduses oleku aega, hoida hoolduskoormust tasakaalus ning maksimeerida hooldusrajatiste kasutamist. [11] Näiteks on rutiinne protsess mootori õlivahetus.

Mitterutiinsed hooldusprotsessid ei ole graafikuga ette määratud ja tulenevad lennuki füüsilisest hetkeseisust. Näiteks mootoririke.

Uurimuses "Digitalization in aircraft maintenance processes" uuriti lennunduse hooldusega seotud spetsialistidelt digitaliseeritavate protsesside kasutatavate ülesannete tüüpi ning nende automatiseerimise ja digitaliseerimise võimalusi. Uuringus küsiti ülesande olemuse kohta – kas vastajate arvates on kasu automatiseerida hooldusprotsessi (rutiinseid või mitterutiinseid). Vastajate hulgas väitis 95%, et rutiinsed ülesanded saab automatiseerida, samas kui ainult 32% oli veendumusel, et mitterutiinseid ülesanded saab automatiseerida. [12]

Sellest tulenevalt on rutiinsed hooldusprotsessid paremini sobivad digitaliseerimiseks ja automatiseerimiseks. Mitterutiinsed ülesanded vajavad inimeste poolset koordineerimist, mida ei saa graafikuga ette ennustada.

1.2 Euroopa Liidu nõuded

Lennuhoolduse rakenduse loomisel ja arendamisel on oluline tagada vastavus Euroopa Liidu sätestatud regulatsioonidele ning täita olulisi nõudeid, mis on seotud väikelennukite hooldusinfo salvestamisega.

Eestis on lennuhoolduse haldamisel oluline järgida Euroopa Liidu määrust nr 1321/2014 „Easy Access Rules for Continuing Airworthiness", mis kehtestab nõuded järjepideva õhukõlblikkuse ja hoolduse info salvestamise kohta. [13] Järgnevana on välja toodud mõned olulisemad punktid.

- **Hooldusinfo salvestamine**: Õhusõiduki jätkuva õhusobivuse haldusorganisatsioon peab salvestama kõik tehtud töö detailid 30 päeva jooksul pärast hooldusprotsessi läbiviimist.

- **Andmete säilitamine**: Organisatsioon peab säilitama koopia kõigist väljastatud või pikendatud õhusobivuse ülevaatustunnistustest ja soovitustest koos kõigi toetusdokumentidega ning õhusobivuse ülevaatustunnistuse koopiast.

- **Andmete säilitamise aeg**: Organisatsioon peab säilitama kõik hooldusprotsessidega seotud andmed kuni kaks aastat pärast lennuki püsivat teenindusest kõrvaldamist.

- **Andmete turvaline säilitamine**: Andmed tuleb säilitada viisil, mis tagab kaitse kahjustuste, muutmise ja varguse eest.

- **Varukoopiate säilitamine**: Kõiki varundussüsteeme tuleb säilitada erinevas asukohas tööandmeid sisaldavast asukohast eraldatuna ja tingimustes, mis tagavad nende heas seisukorras püsimise.

- **Andmete üleandmine**: Kui õhusõiduki jätkuva õhusobivuse haldusorganisatsioon annab lennuohutuse jätkuva haldamise üle teisele organisatsioonile või isikule, tuleb kõik säilitatud andmed üle anda sellele organisatsioonile või isikule. Andmete säilitamiseks kehtestatud ajavahemikud kehtivad uuele organisatsioonile või isikule.

- **Organisatsiooni tegevuse lõpetamine**: Kui õhusõiduki haldusorganisatsioon lõpetab tegevuse, tuleb kõik säilitatud andmed üle anda lennuki omanikule.

1.3 Traditsiooniline paberipõhine dokumenteerimine

Siiani on laialdaselt levinud, et hooldustööde käigus täidetakse erinevaid pabereid, kuhu märgitakse hoolduse käigus tehtud tööd, tehnilised parameetrid ja muu oluline informatsioon. See dokumentatsioon säilitatakse ja arhiveeritakse vastavalt regulatsioonidele ning hooldusajaloo jälgimine toimub paberil dokumentide alusel.

Sellisel lahendusel on selged miinused digitaalsete lahendustega võrreldes:

- **Halb töökindlus** – paberil olev info võib minna lihtsasti kaduma minna.
- **Halb kasutusmugavus** – paber peab füüsiliselt kaasas olema, et infot töödelda.

Uurimuses "Digitalization in aircraft maintenance processes" uuriti ka hooldusprotsesside tüübi kohta, kus võrreldi dokumenteerimisega seotud hooldusprotsesside digitaliseerimise vajadust võrreldes füüsilise hooldusprotsessi digitaliseerimise vajadusega. 95% vastanutest leidis, et dokumenteerimisega seotud ülesanded on vajalikud digitaliseerida , samas vaid 44% vastanutest leidis, et füüsiliste ülesannete digitaliseerimisest oleks kasu. [12]

Saab öelda, et on suur vajadus lennuki hooldusprotsesside automatiseerimisele just dokumenteerimise osas, võrreldes füüsilise protsessiga ise. Antud lõputöö keskendubki dokumenteerimise digitaliseerimisele.

1.4 Tänaste tarkvaralahenduste väljakutsed

Paljud tänased olemasolevad tarkvaralahendused lennukite hooldamiseks on kõrge funktsionaalsusega, paindlikud ja võimekad süsteemid. Kuid neil on miinused kliendi ees, kes on näiteks üksik väikelennuki omanik.

- **Kõrged kulud** - tarkvaralahendused, mis on suunatud lennundussektorile, on tihti keerukad ja sellepärast kallid. Väikeste lennukite omanikel võib olla raske sellistele lahendustele ligipääsu saada ja neid kasutada.
- **Kasutajasõbralikkus** - olemasolevad lahendused võivad olla keerulised ja nõuda põhjalikku koolitust. Suure funktsionaalsusega tarkvara sobib hästi suuremale ettevõttele, kuid üksik väikelennuki omanik tõenäoliselt on rohkem huvitatud kiirusest ja kasutusmugavusest.

1.5 Lõputöö eesmärgi püstitamine

Võrreldes traditsioonilist paberipõhist hooldusdokumentatsiooni ja keerukaid lennundustarkvaralahendusi, pakub lõputöös kirjeldatud lahendus optimaalset keskteed. Rakendus võimaldab väikelennuki omanikel hallata hooldustegevusi, salvestada hooldusandmeid ja jälgida hooldus- ning lennuajalugu mugavalt digitaalsel platvormil, olles samal taskukohane ja lihtsa kasutajaliidesega.

Autor ei leidnud oma uurimise käigus ühtegi ettevõtet, mis vastaks sellistele kriteeriumitele. Seega käesoleva lõputöö eesmärgiks ongi välja töötada taskukohane ja lihtsustatud tarkvaralahendus väikelennukite omanike jaoks.

2 RAKENDUSE FUNKTSIONAALSUS

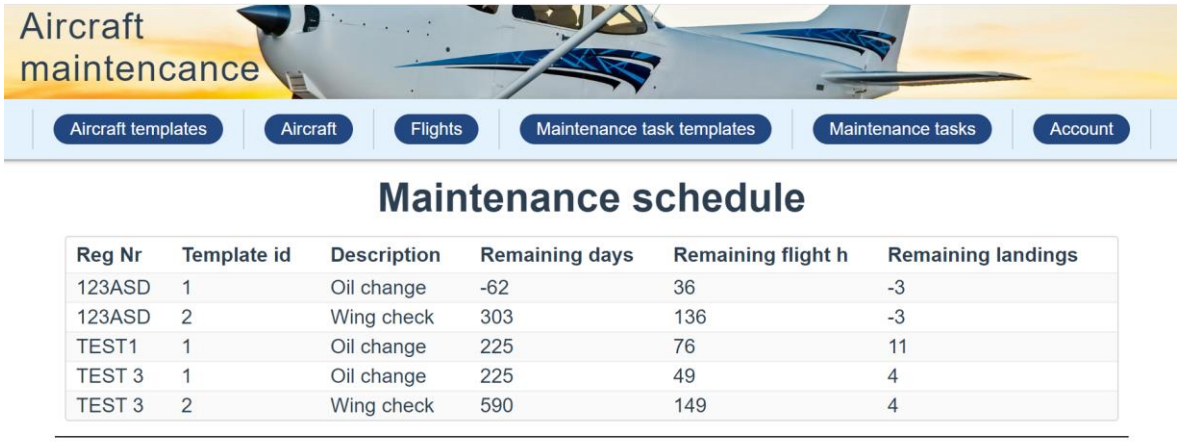
2.1 Peamised funktsionaalsused

2.1.1 Lennuki hoolduste graafiku vaade

Peamine funktsionaalsus kliendi jaoks on näha lennuki hoolduse graafikut (joonis 2.1). Ülejäänud rakenduse funktsionaalsused on toeks sellele, et klient (lennuki omanik) saaks edukalt kasutada seda vaadet.

Oluline on ära mainida, et joonisel 2.1 kuvatud hoolduse infot ei saa niisama lihtsalt küsida vastavast andmebaasi tabelist ja kuvada. See arvutatakse jooksvalt iga kliendi päringu jaoks, sõltudes lennuki tüübist, hiljuti logitud hooldustest, logitud lendudest ja kuupäevast.

- „*Remaining days*" – Allesjäänud päevad konkreetse hooldusprotsessini. Genereeritakse viimase logitud hoolduse, mille on sisestanud näiteks klient, ja päringu tegemise kuupäeva järgi.
- „*Remaining flight h*" – Allesjäänud lubatud lennutundide arv enne hooldusprotsessi. Genereeritakse lennuki tootja poolt määratud hooldusprotsessi, viimase logitud hoolduse, lennuki ajaloo ja kliendi logitud lendude järgi.
- „*Remaining landings*" – Allesjäänud lubatud maandumiste arv enne hooldusprotsessi. Genereeritakse lennuki tootja poolt määratud hooldusprotsessi, viimase logitud hoolduse, lennuki ajaloo ja kliendi logitud lendude järgi.

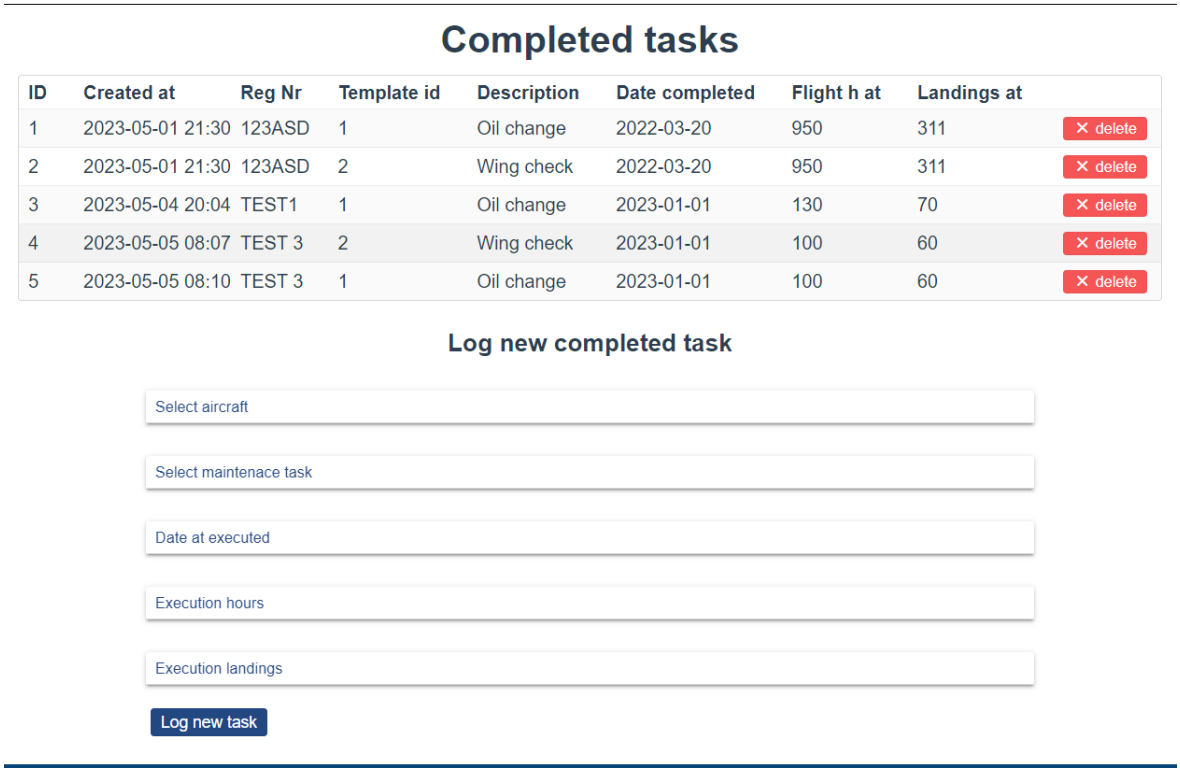


Joonis 2.1 lennuki hoolduse graafik

2.1.2 Teostatud hooldusprotsesside vaade ja haldamine

Järgmine oluline funktsionaalsus kliendi jaoks on hooldusprotsesside vaade (joonis 2.2). Esiteks saab vastava ülevaate klient teostatud hooldustest. Teiseks annab võimaluse kliendil logida sooritatud hooldusprotsesse või neid vajadusel eemaldada. Teostatud hooldusi võivad logida nii lennuki omanik, kui ka näiteks lennuki mehhaanik. Sõltub lihtsalt kasutajate konfiguratsioonist.

Hooldusprotsessi logimisel tuleb valida kõigepealt lennuk ja valida üks sellele määratud hooldusprotsessidest. Seejärel lisada lennuki ajalugu hooldamise hetkel, lennutunnid ja maandumiste arv. Lisaks määrata hoolduse teostamise kuupäev.

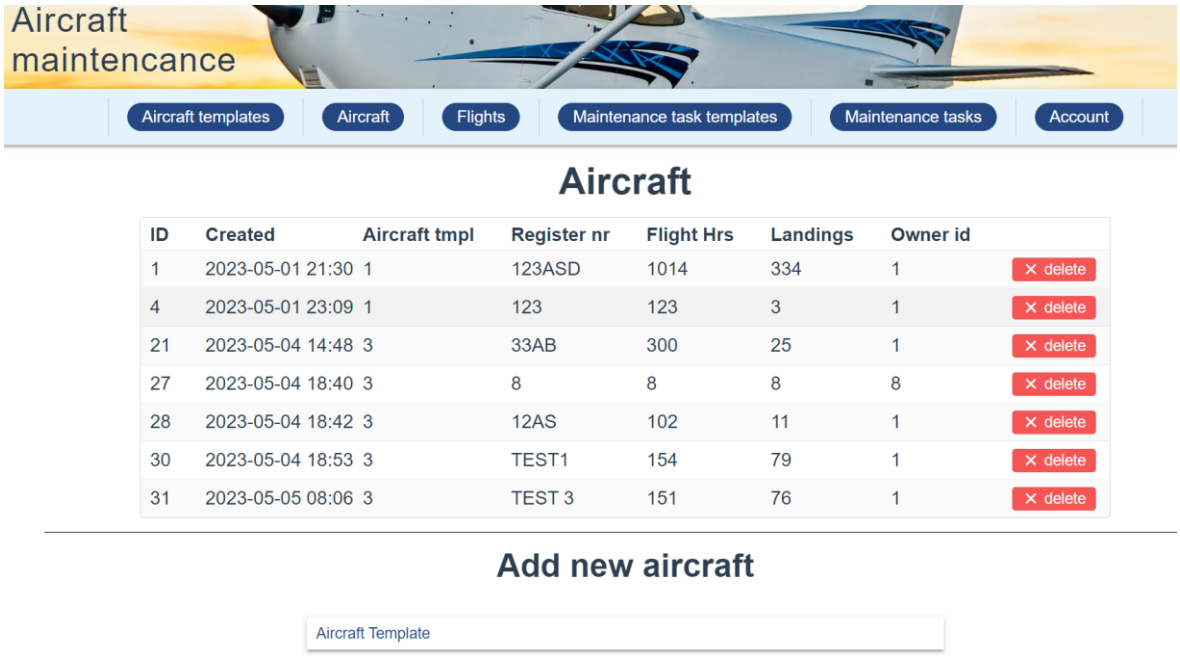


Joonis 2.2 teostatud hooldusprotsesside vaade

2.1.3 Lennukite vaade ja haldamine

Lennukite vaates saab klient näha endaga seotud lennukeid ja nende ajalugu (joonis 2.3).

Saab ka lisada uusi lennukeid või olemasolevaid eemaldada. Uue lisamisel tuleb valida kõigepealt lennuki mall, mis vastab tootjale, mudelile ja aastakäigule. Kui mall puudub, siis uut lennukit ei saa lisada enne, kui hooldusoperaator pole seda lisanud. See tingimus on oluline, et oleks võimalik uut lennukit seostada vastavate hooldusprotsessidega.



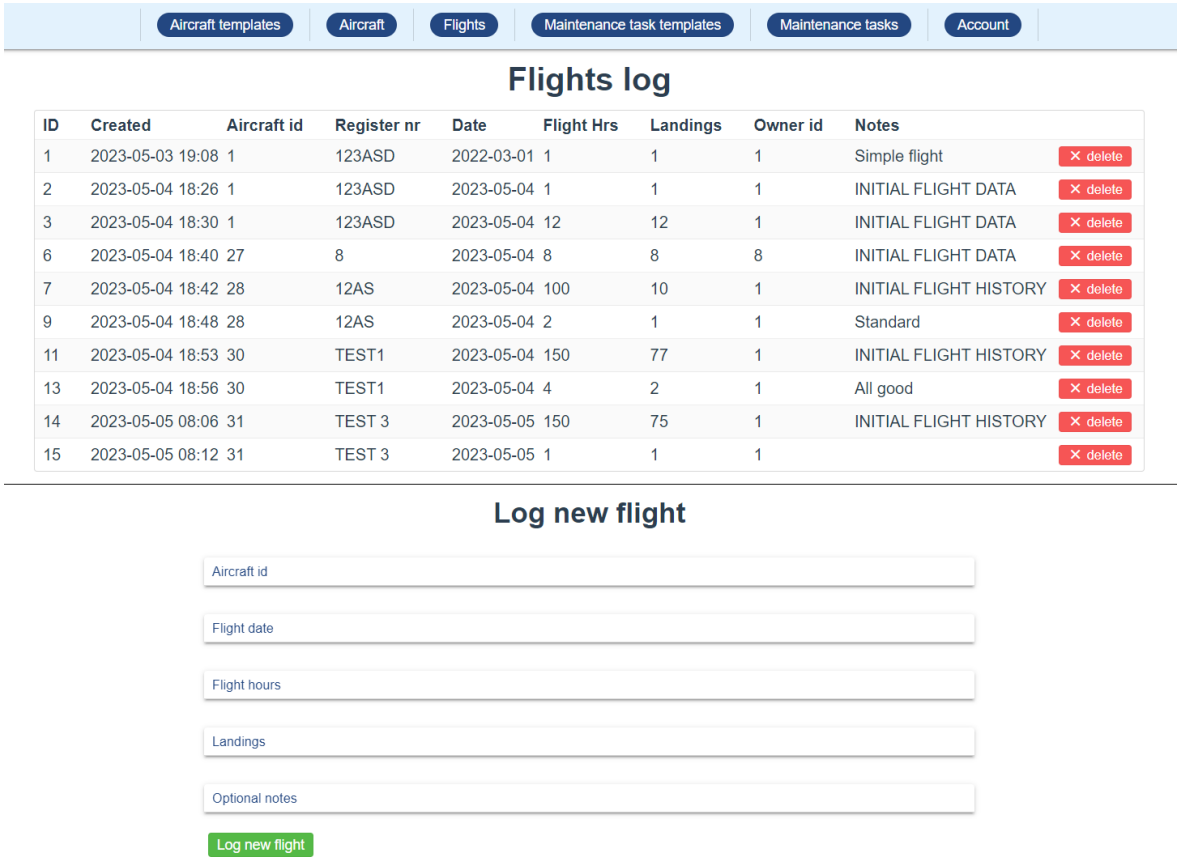
Joonis 2.3 lennukite vaade

2.1.4 Lendude vaade ja logimine

Lendude vaade (joonis 2.4) annab lennuki omanikule võimaluse lisada uusi lende, vaadata nende ajalugu, lisada sooritatud lende ja vajadusel neid ka eemaldada. Lendude ajalugu on oluline ka selleks, et genereerida hoolduste graafikuid.

Iga lennu logi juurde saab määrata, mitu tundi lend kestis, mitu maandumist tehti ja vajadusel lisada märkmeid. Mitu maandumist võib tähendada näiteks, et tehti edasi-tagasi lend.

Lisaks iga kord, kui lisatakse uus lennuk süsteemi, lisatakse üks lennu logi selle konkreetse lennuki kohta. See on lennuki eelnev ajalugu süsteemi lisamise hetkel ja sellel on juures märge „INITIAL_FLIGHT_HISTORY“.



Joonis 2.4 lennatud lendude vaade

2.2 Toetavad funktsionaalsused

2.2.1 Kasutajate haldamine ja rollid

Rakendus toetab kasutajate registreerimist, sisse logimist ja nende info muutmist.

Selleks, et anda erinevatele kasutajatele vastavad õigused, on igal kasutajal oma roll. Järgnevalt toome välja võimalikud rollid, mis võivad kasutajal olla.

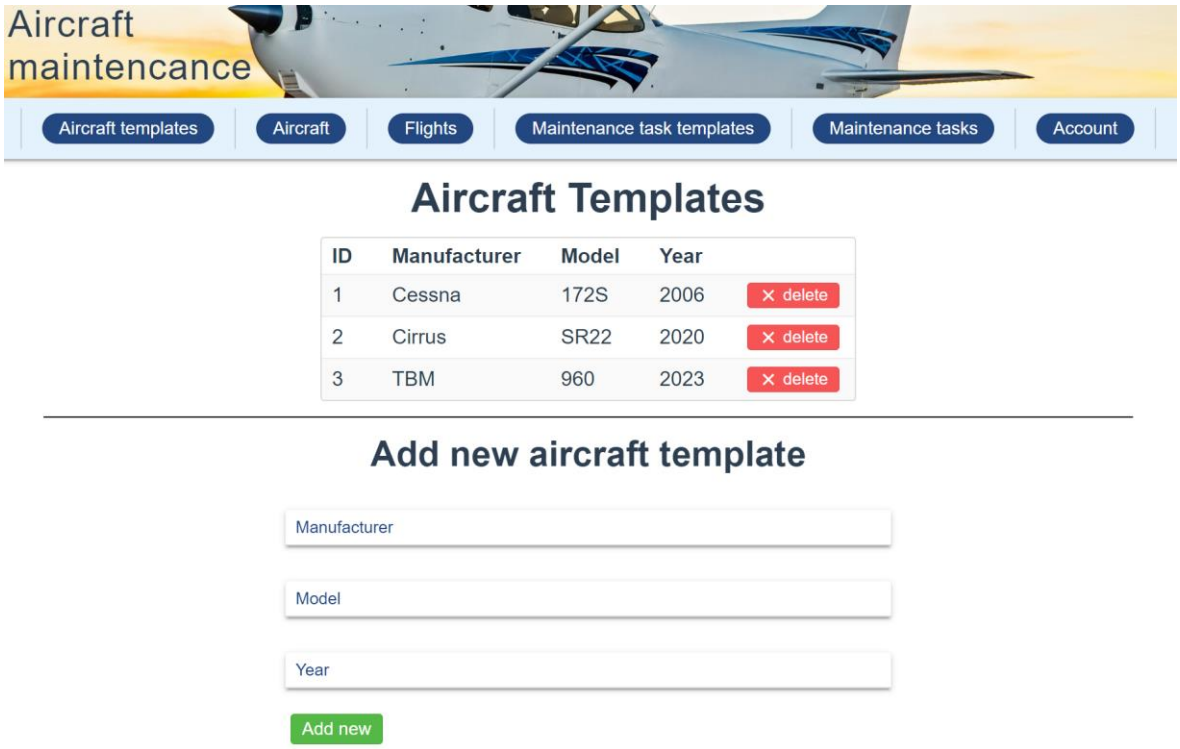
- „*Admin*“ – Näeb kõiki lennukeid, hooldusprotsesse, kasutajaid ja saab seda kõike ka muuta. Saab muuta teisi kasutajaid ja nende rolle. See roll on mõeldud ainult kasutajatele, kes on selle tarkvara omanikud või haldajad.

- „*Maintenance operator*“- Hooldusoperaator, näeb klientide andmeid, saab muuta malle. On sertifitseeritud lennuhooldusega seotud inimene.

- „*Client*“ – Klient, näeb vaid endaga seoses olevaid andmeid, ei ole õigust muuta malle. Lennuki omanik või kasutaja.

2.2.2 Lennuki mallid

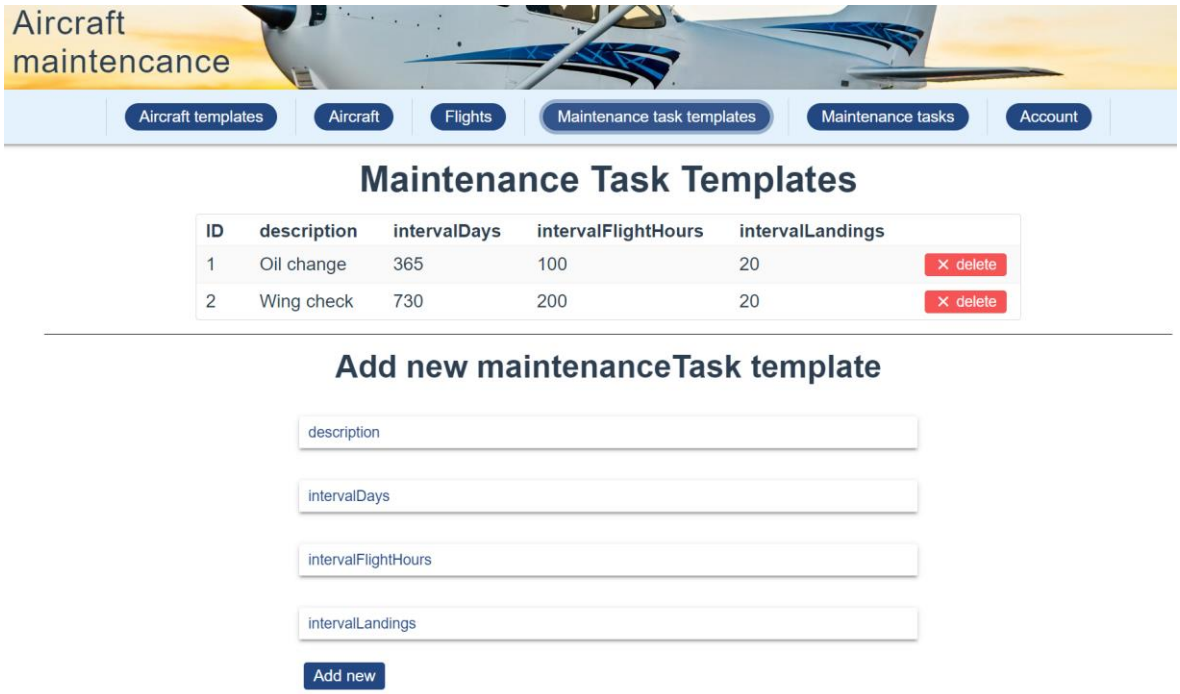
Lennuki mall (joonis 2.5) määrab, millise lennuki mudeli ja aastakäiguga on tegu. Seda on vaja, et lennukit hiljem saaks seostada vaid talle sobilike hoolduste ja omadustega. Lennuki malle saab lisada ainult „*Maintenance operator*“ rolliga kasutaja (või admin). Kliendid saavad siis seejärel lisada oma lennuki, valides sellele sobiv mall. Kui klient avastab, et tema lennukile sobivat malli ei leidu, siis tuleb see luua hooldusoperaatori poolt.



Joonis 2.5 lennuki mallide vaade

2.2.3 Hooldusprotsessi mallid

Hooldusprotsessi mall (joonis 2.6) kirjeldab ühe konkreetse hooldusprotsessi tüüpi. Igale lennuki mallile lisatakse talle vastavad hooldusprotsessi mallid. Hooldusprotsesside malle saab lisada ainult „Maintenance operator“ rolliga kasutaja (või admin). Lennuki omanik ise ei ole piisavalt usaldusväärne, et seda infot muuta, sest hooldusprotsessi kirjeldus peab kindlasti vastama lennuki tootja poolt väljastatud juhendile.



Joonis 2.6 hooldusprotsesside vaade

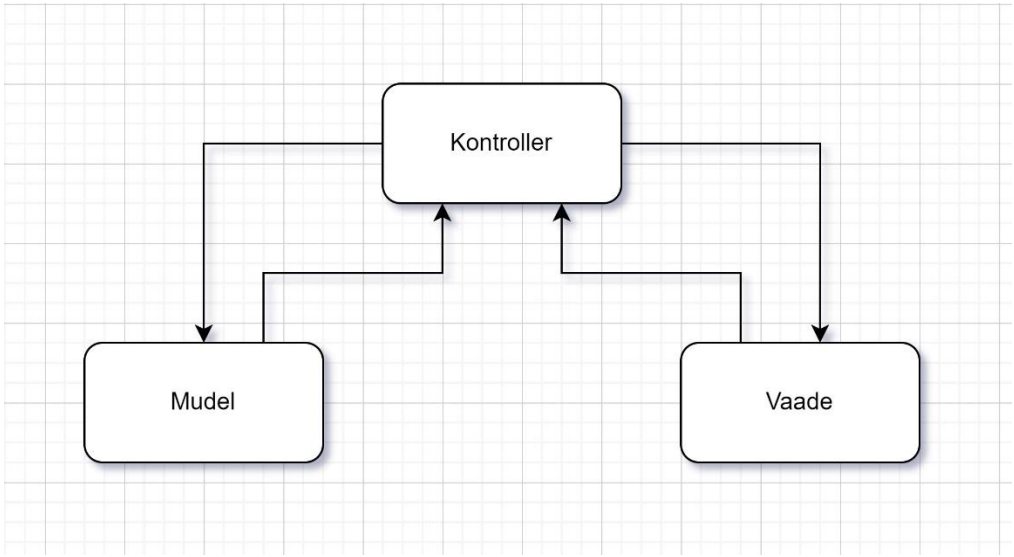
3 RAKENDUSE ARHITEKTUUR

Rakenduse arhitektuur on süsteemi terviklik kujutis, mis hõlmab süsteemi komponente, nende omavahelisi seoseid ja käitumist. Hästi planeeritud arhitektuur on kogu rakenduse kvaliteedi aluseks. Selles lõputöös kasutatakse MVC (Mudel-Vaade-Kontroller) arhitektuurimustrit. Lisaks on teine oluline arhitektuuriline otsus kasutada mikroteenuseid.

3.1 MVC (Mudel-Vaade-Kontroller)

MVC ehk Mudel-Vaade-Kontroller (joonis 3.1) on arhitektuurimuster, mis aitab eristada rakenduse kolme erinevat omavahel seoses osa. [14]

- **Mudel** - talletab info, mis on saadud kontrollerist
- **Vaade** - info esitlus, näiteks tabel või diagramm
- **Kontroller** - võtab sisendina sisse info ning muudab selle käsklusteks



Joonis 3.1 Mudel – Vaade – Kontroller

3.2 Mikroteenuste arhitektuur

Mikroteenuste arhitektuur on kaasaegne lähenemine, mis võimaldab rakenduse funktsionaalsust jagada väiksemateks, iseseisvateks teenusteks. See tähendab, et iga teenus on eraldiseisev ja seda saab arendada, testida, juurutada ja skaleerida sõltumatult. [15]

Mikroteenuste arhitektuur sai valitud monoliitse rakenduse asemel mitmetel põhjustel:

- **Iseseisev skaleerimine** – kui üks teenus vajab rohkem koormust, saab selle ressursse eraldi suurendada, vastavalt vajadusele. [16]
- **Veakindlus** - kui iga mikroteenus on iseseisev ja autonoomne, siis ühe teenuse rike ei mõjuta alati teisi teenuseid. [15]
- **Modulaarsus** – mikroteenuseid saab lihtsamini mõista, arendada, testida ning muuta vastupidavamaks arhitektuuri erosioonile. [15]

Need põhimõtted võivad tunduda väheolulised rakenduse esmase versiooni puhul, kuid tuleviku vaates ja funktsionaalsuse suurenedes muutuvad järjest olulisemaks. Lihtsam on alustada kohe mikroteenustega, kui autoril hakata hiljem monoliitset rakendust mikroteenustele üle viima.

Täpsemalt, kuidas antud töös mikroteenuste arhitektuur implementeeriti, on kirjeldatud põhiosa peatükis 5.

4 KASUTAJALIIDESE LOOMINE

4.1 Kasutajaliidese kavandamine

Kasutajaliidese loomisel on peamiseks põhimõtteks ja fookuseks antud lõputöö kontekstis funktsionaalsus. Aga lisaks jälgitakse ka disaini põhimõtteid, nagu näiteks kasutusmugavus ja esteetlikkus.

Kasutajaliidese kavandamiune hõlmab erinevate kasutajaliidese komponentide, nagu nuppude, menüüde, vormide ja teabekuva elementide kavandamist. Kavandamise etapis on oluline tagada, et rakenduse kasutajaliidese elementide paigutus ja struktuur oleks loogiline ja kasutajasõbralik.

Kasutajaliidese realiseerimse protsessis võib välja tuua kaks olulisemat valikut. JavaScripti raamistikuks valis autor Vue.JS 3 ja sellele omakorda komponentide teegiks valiti PrimeVue.

4.2 Vue.js 3 ja PrimeVue kasutamine

4.2.1 Vue.js 3

Autor kaalus kasutajaliidese arendamiseks kolme erinevat JavaScripti raamistikku: Angular, React ja Vue.js. Lõpuks osutus valituks Vue.js, millest on viimane versioon Vue.js 3.

Valik põhines mitmel põhjusel. Esiteks oli autoril varasem kogemus Vue.js raamistikuga. Teiseks pakuvad kõik kolm raamistikku sarnaseid funktsionaalsusi, seega autori arvates on varasem kogemus piisav põhjus siinkohal valiku tegemiseks.

Vue.js 3 on populaarne JavaScripti raamistik, mida kasutatakse veebirakenduste arendamisel, eriti nende kasutajaliidese loomiseks. Vue.js 3 on uusim versioon, mis pakub täiustatud funktsioone, nagu näiteks *„composition API"*, mis muudab komponendid loetavamaks, kui varasemates versioonides *„options API"*. [17]

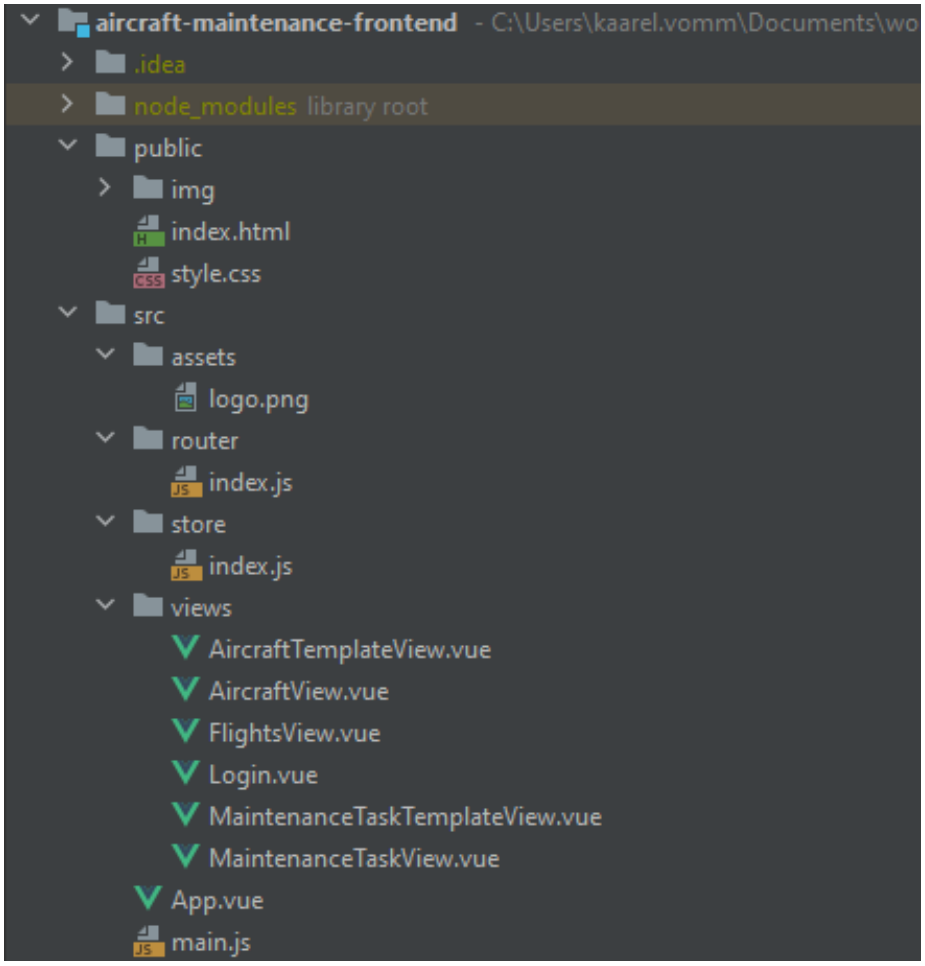
4.2.2 PrimeVue

Kasutajaliidese komponentide teegiks valiti PrimeVue, mis on mõeldud Vue.js raamistikule. PrimeVue pakub laia valikut valmis komponente, nagu nupud, vormielemendid, tabelid ja palju muud. [18]

Varem kasutas autor Vuetify teeki, mis sobis hästi Vue.js 2 versiooniga. Kuid selles projektis oli valikuks uus Vue.js 3 versioon ja selgus, et Vuetify ei tööta Vue.js 3-ga nii sujuvalt, kuna mõned komponendid olid poolikud või puudusid. Vuetify kodulehel on pooleli olevate komponenti kohta väljatoodud nimekiri, ja selles sisaldusid mitmed autoril arenduseks kriitilised komponendid, näiteks andmetabel. [19] Seega oli vaja leida uus teek, milleks sai PrimeVue.

4.2.3 Rakenduse komponentide loomine

Vue.js 3 ja PrimeVue abil luuakse rakenduse kasutajaliidese komponendid. Need komponendid aitavad kaasa modulaarsele ja skaleeritavale arhitektuurile, mis omakorda parandab koodi kvaliteeti ja lihtsustab rakenduse arendamist. Komponentide loomisel järgitakse häid programmeerimistavasid, nagu koodi taaskasutamine, komponentide hierarhia ja vastutusalade eraldamine. Järgneval joonisel (joonis 4.1) on näidatud Vue.js-iga loodud komponente ja projekti ülesehitust.



Joonis 4.1 Vue.js projekti struktuur

4.3 Tulemused ja õppetunnid

Valminud kasutajaliides koosneb kuuest erinevast vaatest, millest igaüks omab eraldi funktsionaalsust. Kasutajaliidese vaated on illustreeritud teises peatükis asuvatel joonistel 2.1 kuni 2.6.

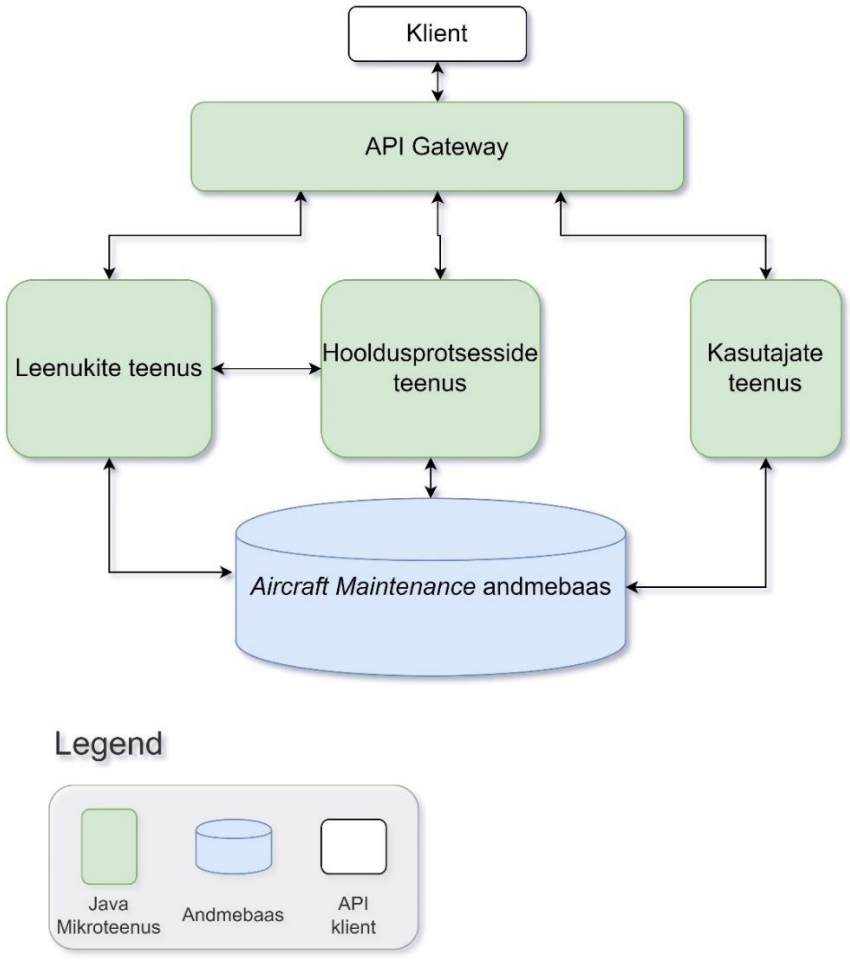
Kokkuvõttes on Vue.js 3 ja PrimeVue kasutamine võimaldanud luua stabiilse, optimeeritud ja kergesti hooldatava kasutajaliidese. See kogemus on tõestanud, et valitud raamistike paindlikkus ja lai valik kasutajaliidese komponente suudavad toetada kvaliteetset veebirakenduste arendust. Tulevikus võib neid tööriistu kindlasti kaaluda teiste projektide kontekstis.

5 SERVERRAKENDUSE LOOMINE

Rakendusloogika on rakenduse koodi osa, mis määrab, kuidas rakendus käitub ja reageerib erinevatele sündmustele ja kasutaja interaktsioonidele. Rakendusloogika kujundamine ja loomine on oluline osa süsteemiarendusest, kuna see määrab suuresti rakenduse funktsionaalsuse.

5.1 Mikroteenused

Selles projektis on rakendusloogika jaotatud erinevate teenuste vahel, mis vastutavad konkreetsete funktsionaalsuste eest. Teenuste kasutamine võimaldab loogika jaotada väiksemateks, hallatavateks osadeks, mis on kergemini testitavad ja taaskasutatavad. Lennukite hooldustööde rakenduse kontekstis on loodud erinevad teenused, mis vastutavad erinevate funktsioonide eest. Kõik teenused on loodud Spring Booti raamistiku abil. (Joonis 5.1)



Joonis 5.1 Mikroteenuste arhitektuur

5.1.1 Lennukite teenus

Lennukite teenus haldab lennukeid ja nende mudelite andmeid. Teenus võimaldab kasutajatel lisada, muuta ja kustutada lennukeid ja lennukite mudeleid. Lisaks võimaldab hallata lennukite hoolduse ajakava ja ajalugu.

5.1.2 Hooldusprotsesside teenus

Hooldusprotsesside teenus haldab hooldusprotsesse ja nende malle. Teenus võimaldab kasutajatel lisada, muuta ja kustutada hooldusprotsesse ja nende malle.

5.1.3 Kasutajate teenus

Kasutajateenusel on kaks suuremat ülesannet:

- Kasutajate haldamine – Vastutab kasutajakontode loomise, haldamise ja autentimise eest. See teenus võimaldab kasutajatel luua uusi kontosid, muuta oma konto andmeid, logida sisse ja välja ning hallata oma õiguseid rakenduses.
- Autentimine – Kasutajate teenus loob ja kontrollib JWT tokeneid.

5.2 API disain

API disain on oluline samm tagamaks, et API on loogiline, ühtne ja kasutajasõbralik. API lõpp-punktid kujundati nii, et need vastavad konkreetsetele toimingutele süsteemis. Lähtuti REST arhitektuuri põhimõtetest, kus iga ressurss on unikaalselt tuvastatav URL-i kaudu ja toimingud ressursiga määratakse HTTP meetodite abil.

5.2.1 API lõpp-punktid

Lennu haldamine

- Lennu loomine (ehk logimine): POST /flight/create
- Kõigi lendude saamine: GET /flight/all
- Lennuki lendude saamine: GET /flight/{aircraftId}

Lennuki haldamine

- Lennuki loomine: POST /aircraft/create
- Kõigi lennukite saamine: GET /aircraft/all
- Spetsiifilise lennuki saamine id järgi: GET /aircraft/{id}
- Lennuki kustutamine id järgi: DEL /aircraft/delete/{id}

Lennuki malli haldamine

- Lennuki malli loomine: POST /aircraft-template/create
- Kõigi lennuki mallide saamine: GET /aircraft-template/all
- Spetsiifilise lennuki malli saamine id järgi: GET /aircraft-template/id
- Lennuki malli kustutamine id järgi: DEL /aircraft-template/delete/{id}

Hooldusprotsessi haldamine

- Hooldusprotsessi loomine: POST /task/create
- Kõigi ülesannete saamine: GET /task/all
- Spetsiifilise ülesande saamine id järgi: GET /task/id
- Hooldusprotsessi kustutamine id järgi: DEL /task/delete/{id}

Hooldusprotsessi malli haldamine

- Hooldusprotsessi malli loomine: POST /task-template/create
- Kõigi ülesande mallide saamine: GET /task-template/all
- Spetsiifilise ülesande malli saamine id järgi: GET /task-template/id
- Hooldusprotsessi malli kustutamine id järgi: DEL /task-template/delete/{id}

Graafiku haldamine

- Kasutaja hooldusgraafiku saamine: GET /schedule/all
- Kasutaja lennuki hooldusgraafiku saamine: GET /schedule/{aircraftId}

Kasutaja haldamine

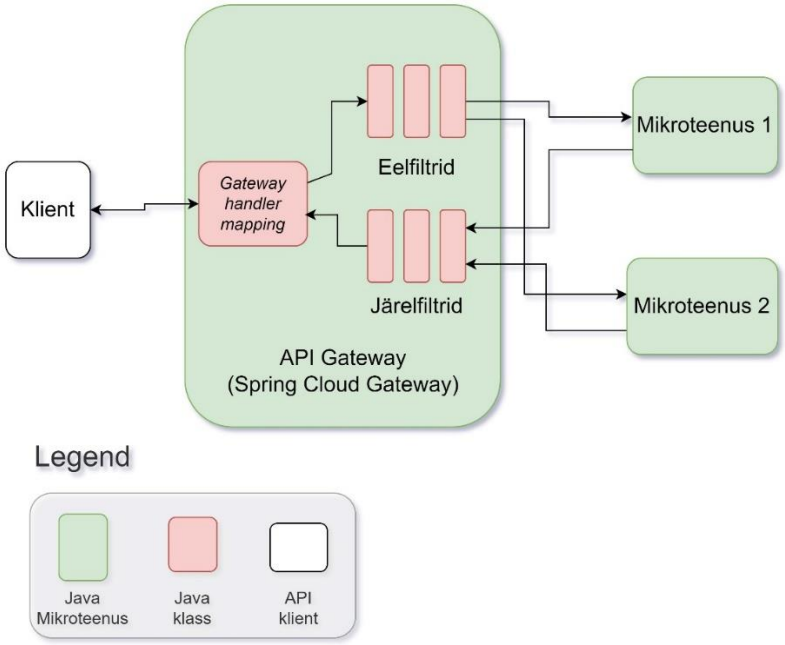
- Kasutaja registreerimine: POST /user/register
- Kasutaja sisselogimine: POST /user/login
- Kasutaja kustutamine: DELETE /user/login
- Kasutaja muutmine: PUT /user/edit
- Kasutaja JWT tokeni uuendamine: POST /user/auth/refresh

5.3 API Gateway teenus

API Gateway on ainuke avalikult kättesaadav teenus API kasutajale. See tähendab, et kõik päringud API baasaadressil lähevad esialgu läbi selle teenuse. API Gateways kontrollitakse päringut ja suunatakse seejärel edasi vastava mikroteenuse suunas. API Gateway on ehitatud Spring Cloud Gateway Java teeki kasutades. [20]

Selle teenuse kasutamisel on projektis mitmeid eeliseid:

- Kasutajaliides saab kasutada vaid ühte baasaadressi, aga teha päringuid erinevatele mikroteenusetele.
- Turvalisus – Kõik päringud lähevad läbi globaalse eelfiltri API Gateways (joonis 5.2), kus iga kord kontrollitakse JWT tokenit.



Joonis 5.2 Spring Cloud Gateway

5.4 Mikroteenuste omavaheline suhtlemine

Mikroteenuste omavaheliseks suhtlemiseks võib kasutada erinevaid meetodeid. Vaatleme neist lühidalt kahte lähenemist, mida autor arenduse käigus kaalus. Järgnevalt on välja toodud Rest päringute eelised ja puudused GRPC ees ning vastupidi.

REST päringud

REST võimaldab kasutada standardseid HTTP protokolli meetodeid, nagu näiteks GET või POST. [8]

- **Eelised** - lihtsam mõista ja rakendada. Kasutusel ka juba kasutajaliidese ja serverrakenduse vahelisel suhtlemisel. [21]

- **Puudused** – Halvem jõudlus, kui GRPC-l, sest kasutatakse HTTP/1 protokolli ja andmed saadetakse JSON kujul. [21]

GRPC

GRPC on Google loodud raamistik, mis kasutab HTTP/2 protokolli, erinevalt REST päringutest, mille standard on HTTP/1. [4]

- **Eelised** – Parem jõudlus, kui REST päringutel tänu HTTP/2 protokollile ja väiksematele andmemahtudele, sest sõnumi keha saadetakse binaarkoodis. [21]
- **Puudused** – Keerukam mõista ja arendada kui REST. [21]

Lõputöö arendusel sai valitud mikroteenuste suhtlemiseks REST päringud, sest arendusprotsess on oluliselt lihtsam. Ühtlasi puudub esmase versiooni arenduse raames vajadus suuremale jõudlusele.

5.5 Turvalisus ja autentimine

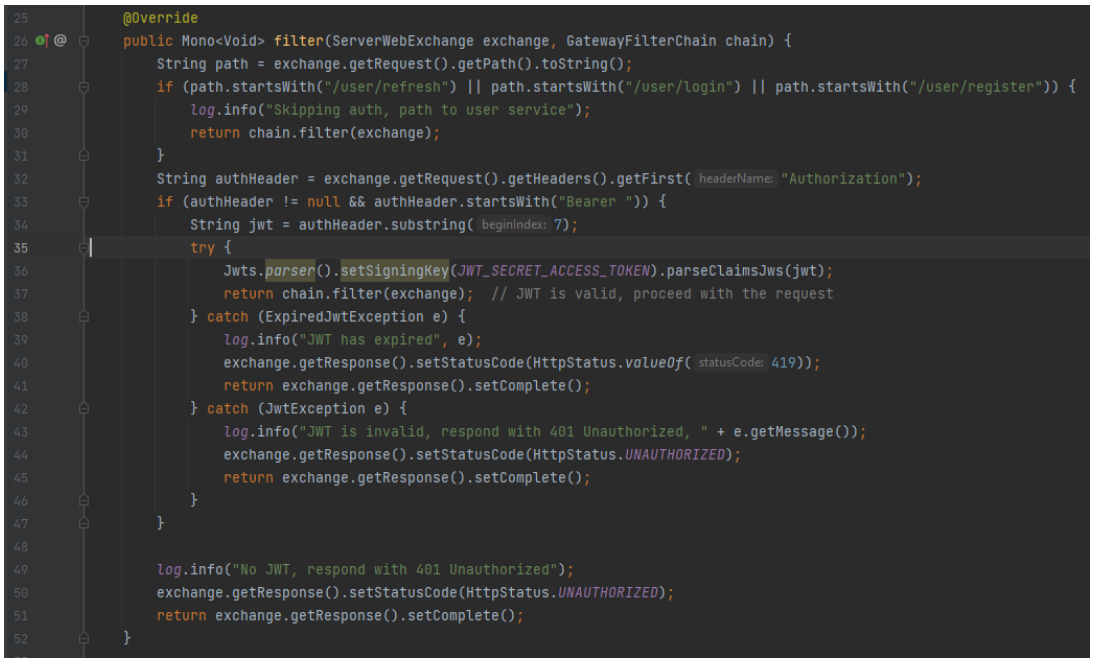
Turvalisuse ja autentimise tagamiseks kasutatakse JSON Web Token (JWT) tehnoloogiat. JWT on internetistandard andmete loomiseks koos digitaalallkirjaga [6].

Token luuakse Kasutajate Haldamise teenuses (joonis 5.3).

```
28 @    public String generateAccessToken(UserDTO user) {
29          // Expiration time for access token
30          Date expiryDate = new Date(new Date().getTime() + JWT_ACCESS_EXPIRATION_MILLIS);
31
32          return Jwts.builder()
33                  .setSubject(String.valueOf(user.getId())) // Set subject as the id
34                  .setIssuedAt(new Date()) // The time token was issued
35                  .setExpiration(expiryDate) // The time after which the token is no longer valid
36                  .signWith(SignatureAlgorithm.HS512, JWT_SECRET_ACCESS_TOKEN) // Sign the token
37                  .compact();
38      }
```

Joonis 5.3 Tokeni genereerimine Kasutajate teenuses

Token kontrollitakse Gateway teenuses. Spring Cloud Gateway Java teek pakub võimaluse luua koodis globaalne filter, mille sisu saab vastavalt soovile Java koodis kujundada (joonis 5.4).



```
@Override
public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
    String path = exchange.getRequest().getPath().toString();
    if (path.startsWith("/user/refresh") || path.startsWith("/user/login") || path.startsWith("/user/register")) {
        log.info("Skipping auth, path to user service");
        return chain.filter(exchange);
    }
    String authHeader = exchange.getRequest().getHeaders().getFirst( headerName: "Authorization");
    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        String jwt = authHeader.substring( beginIndex: 7);
        try {
            Jwts.parser().setSigningKey(JWT_SECRET_ACCESS_TOKEN).parseClaimsJws(jwt);
            return chain.filter(exchange);  // JWT is valid, proceed with the request
        } catch (ExpiredJwtException e) {
            log.info("JWT has expired", e);
            exchange.getResponse().setStatusCode(HttpStatus.valueOf( statusCode: 419));
            return exchange.getResponse().setComplete();
        } catch (JwtException e) {
            log.info("JWT is invalid, respond with 401 Unauthorized, " + e.getMessage());
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }
    }

    log.info("No JWT, respond with 401 Unauthorized");
    exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
    return exchange.getResponse().setComplete();
}
```

Joonis 5.4 Filter tokeni kontrollimiseks Gateway teenuses

5.6 Tulemused ja õppetunnid

Tulemusena valmis RESTful API, mis koosneb lõpp-punktidest, mis hõlmavad lennukite, lennukite mallide, ülesannete, ülesannete mallide, lendude ning kasutajate ja nende hooldusgraafikute haldamist. Need lõpp-punktid pakuvad laia valikut funktsioone, alates andmete loomisest, lugemisest, uuendamisest kuni kustutamiseni, võimaldades sellega terviklikku andmetöötlust.

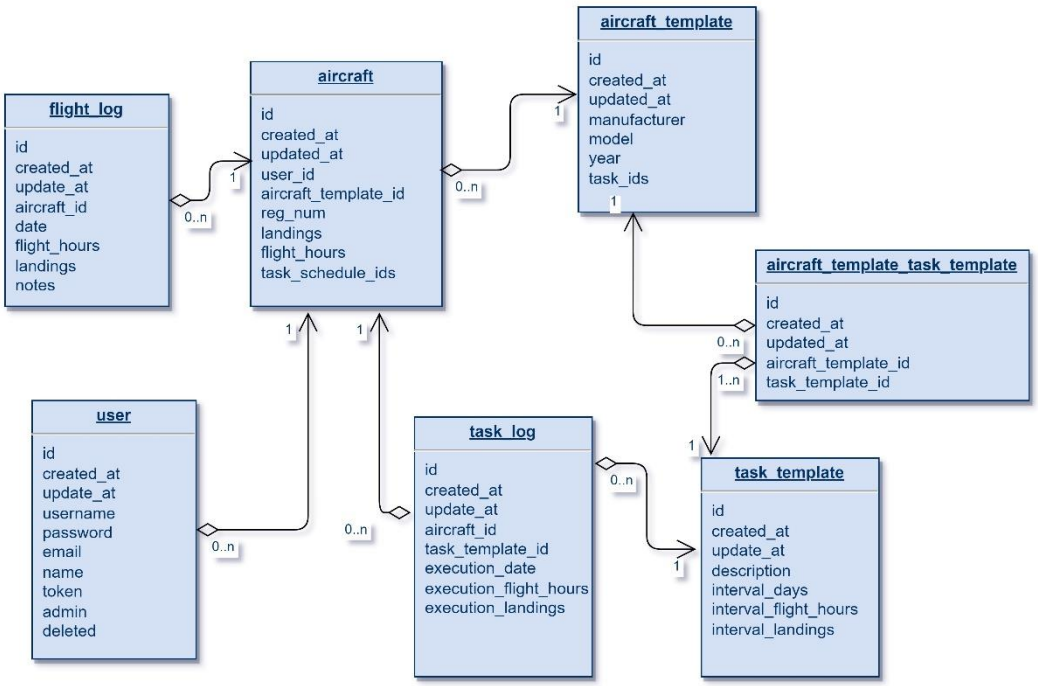
API arendamise käigus õpiti, kui oluline on korrektselt ja loogiliselt kujundada API lõpp-punkte. Lõpp-punktid tuleb kujundada nii, et need oleksid intuitiivsed ja kasutajasõbralikud, mis aitab kaasa paremale kasutajakogemusele.

6 ANDMEMUDELI LOOMINE

6.1 Andmemudelite loomine

Andmemudeli loomiseks on valitud tööriistaks draw.io veebisait. See on tasuta ja avatud lähtekoodiga veebipõhine diagrammitööriist, mis toetab andmebaasi skeeme. [22] Valiku põhjuseks lihtne kasutatavus ja tööriista võimalus töötada veebis, ilma tarkvara alla laadimata.

Töö käigus loodi andmemudel, mis sisaldab seitset tabelit. (joonis 6.1)



Joonis 6.1 Andmemudel

6.2 Andmebaasi valik

PostgreSQL on valitud andmebaasina, kuna see on avatud lähtekoodiga andmebaasihaldussüsteem, mis pakub laialdast tuge, tugevaid turvafunktsioone ning stabiilsust ja töökindlust. PostgreSQL sobib hästi andmete säilitamiseks ning tagab rakenduse efektiivse ja tõrgeteta toimimise. [22]

Kuigi sama võib saavutada ka näiteks MySQL-iga või mõne kolmanda andmebaasihaldussüsteemiga, siis autor lähtus valikul enda isiklikust pikaajalisest kogemusest PostgreSQL-iga.

6.3 Andmemudelite integreerimine PostgreSQL andmebaasiga

Andmemudelid integreeritakse PostgreSQL andmebaasiga, et võimaldada andmete salvestamist, otsimist, värskendamist ja kustutamist. Selleks kasutatakse Spring Booti ja Flyway Java teeki, mis hõlbustab andmebaasiühenduste loomist ja haldamist.

6.3.1 Flyway Java teek

Flyway on avatud lähtekoodiga andmebaasimigratsiooni tööriist, mis on loodud selleks, et lihtsustada andmebaasi skeemi versioonihaldust. Flyway abil saab hõlpsasti luua, hallata ja järjekindlalt teostada andmebaasi muudatusi. Flyway on integreeritud Spring Bootiga ja seda saab kasutada PostgreSQL ja teiste populaarsete andmebaasihaldussüsteemidega. [23]

Flyway toimimispõhimõte põhineb SQL-skriptidel, mis on järjestatud versiooninumbri järgi. Kui rakendus käivitub, kontrollib Flyway, milliseid skripte pole veel andmebaasis käivitatud, ning rakendab need järjekorras. Flyway jälgib, milliseid skripte on juba käivitatud, luues andmebaasis eraldi tabeli flyway_schema_history, kus hoiustatakse info rakendatud migratsioonide kohta.

Põhjused, miks Flyway sai valitud:

- **SQL ajalugu** - kõik andmebaasimuudatused on SQL failidena salvestatud ja järjestatavad vastavalt kuupäevale. [23]
- **Korratavus** - lihtne luua samasugune andmebaasi struktuur uues andmebaasis. Kasulik näiteks testimisel või uue keskkonna loomisel. [23]

6.3.2 Loodud SQL

Järgmisel joonisel 6.2 on kujutatud SQL faili, millega Flyway Java teek tabelid loob.

Joonis 6.2 SQL fail andmebaasi loomiseks Flyway abil

6.4 Andmete indekseerimine

Andmete indekseerimine on võtmetähtsusega strateegia andmebaasi jõudluse optimeerimiseks. See aitab kiirendada päringuid, vähendades vajalike andmeotsingute aega. PostgreSQL pakub erinevaid indekseerimisvõimalusi, sealhulgas B-puu, hash, GiST, SP-GiST, GIN ja BRIN. [24]

Antud töös on kasutatud vaid B-puu indekseerimist, sest see on PostgreSQL vaikimisi indekseerimise meetod ja keerukama meetodi jaoks vajadus puudub.

6.5 Tulemused ja õppetunnid

Andmemudeli loomise ja integreerimise protsessi tulemusel valmis seitsme tabeliga andmemudel, mis sai integreeritud PostgreSQL andmebaasi. Iga tabel vastab konkreetsele funktsionaalsusele süsteemis.

Üldiselt näitas see protsess, kui oluline on hoolikas andmekorraldus ja efektiivne andmebaasistrateegia tarkvararakenduse jõudluse ja stabiilsuse tagamisel. Arenduse käigus oli autoril vaja mitu korda andmebaasi struktuuri muuta. Näiteks, ei taibanud autor esialgsesse andmemudelisse kaasata tabelit „aircraft_template_task_template“, mis määrab mitu-mitmest suhte kahe vastava tabeli vahel, mis tekitas probleeme lennukite ja hooldusprotsessi mallide ühendamisel.

7 RAKENDUSE JUURUTAMINE KESKKONDA

Rakenduse juurutamise protsess selles projektis hõlmas kolme peamise etapi läbimist: Arenduskeskkondade loomist, konfigureerimist ja selle üleslaadimist Heroku platvormile.

7.1 Heroku tutvustus ja valiku põhjendus

Heroku on pilvepõhine PaaS lahendus, mis toetab mitmeid programmeerimiskeeli, sealhulgas Java ja Node.js, mida kasutatakse selles lõputöös. Heroku võimaldab arendajatel hõlpsasti juurutada, skaleerida ja hallata rakendusi. [25]

Heroku juurutamine on lihtne ja automatiseeritud. See on integreeritud Git versioonihaldussüsteemiga, mis tähendab, et juurutamine on sama lihtne kui oma koodi Git repo pushimine Heroku teenusesse. Herokul on võimalus automaatselt juurutada rakendusi, mis on seotud GitHubi repositooriumiga ja võimaldab automatiseerida juurutamisprotsessi. [25]

Kui rakendus on Heroku'sse üles laaditud, on see saadaval avaliku URL-i kaudu, mis teeb testkeskkonna loomise ja rakenduse jagamise lihtsaks. Samuti pakub Heroku hulgaliselt lisateenuseid, nagu andmebaasiteenused, teatejärjekorra funktsioonid ja monitoorimine. [25]

Heroku platvormi kasutamise eeliseks on ka see, et see haldab infrastruktuuri, võimaldades arendajatel keskenduda rakenduse arendamisele. See tähendab, et arendajad ei pea muretsema selliste küsimuste pärast nagu serveri ülalpidamine või tulemüüri seadistamine, mis võimaldab neil kiiremini ja tõhusamalt töötada. [25]

Lõputöö kontekstis kasutatakse Heroku platvormi rakenduse juurutamiseks ja avalikustamiseks. See võimaldab hõlpsat ligipääsu rakendusele kõigile huvitatud osapooltele ja annab võimaluse rakenduse tööd igal ajal kontrollida ja hinnata.

7.2 Arenduskeskkondade loomine

Esialgne plaan oli luua kolm arenduskeskkonda. Igat keskkonda hakatakse kasutama erinevas arendusetapis ja kasutab eraldi andmebaasi.

1. **DEVELOPMENT** – Lokaalselt arendamiseks oma masinal ja esmaseks testimiseks. Rakendused ja andmebaas käivitatakse lokaalselt.
2. **TEST** – Valmis ehitatud uue funktsionaalsuse testimiseks stabiilses keskkonnas. Heroku andmebaas. Rakendused ehitatakse Heroku serverisse, keskkonna põhine andmebaas. Kliendil ligipääs puudub.
3. **LIVE** – Testitud ja töökindel funktsionaalsus, mida kasutaja reaalmaailmas kasutab. Rakendused ehitatakse Heroku serverisse, keskkonna põhine andmebaas.

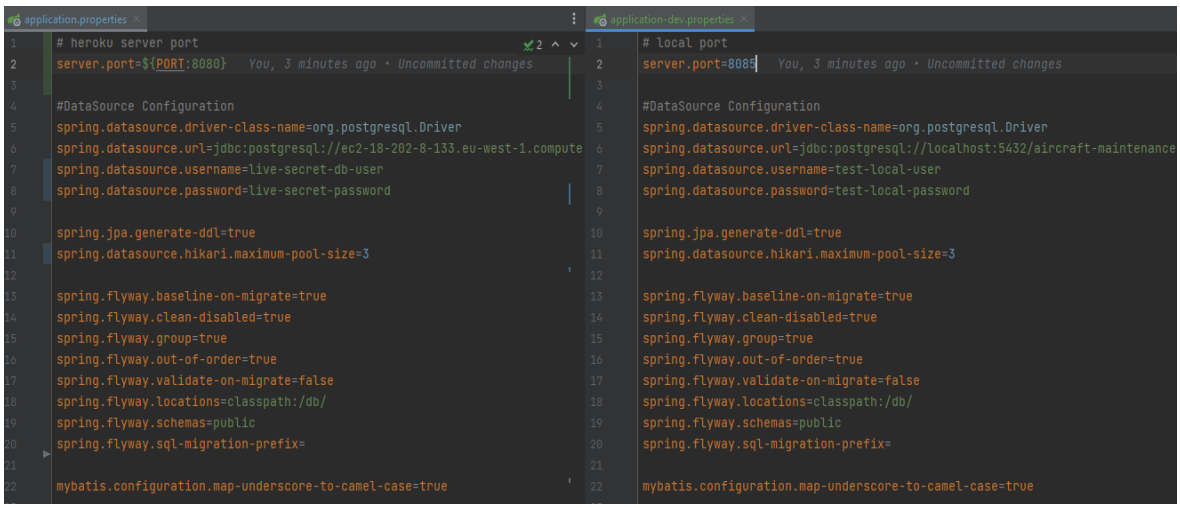
Arenduse käigus leidis autor, et TEST keskkond ei ole antud lõputöö projekti raames vajalik, sest kahe keskkonnaga saab arendusprotsessi etapid tehtud. Ühtlasi on Heroku iga andmebaasi ülespanek tasuline, seega hoiab TEST keskkonnast loobumine ka raha kokku.

Keskkonnad lihtsustati ainult DEVELOP ja LIVE keskkonnale. Need kaks keskkonda on hädavajalikud, sest DEVELOP ja LIVE andmed ei tohiks kindlasti olla samas kohas.

7.3 Konfiguratsioon

Konfiguratsioon hõlmas selliste detailide määramist nagu andmebaasi ühendusinfo, keskkonnamuutujad, portide numbrid ja autentimisandmed.

Näiteks Spring Boot rakendusel saab väga mugavalt kasutada erinevate keskkondade konfigureerimiseks erinevaid (joonis 7.1).



Joonis 7.1 Keskkondade konfigureerimine

7.4 Live keskkonda juurutamine

Rakenduse üleslaadimise etapis kasutati Heroku CLI-d (*Heroku Command Line Interface*), mis on Heroku poolt pakutav käsurea tööriist arendajatele. [26] Käesoleva projekti raames laaditi rakenduse kood GitHubi ja sealt edasi ehitati teenus Heroku pilveteenuse platvormile. Teenuse ehitamise protsessis installitakse vajalikud sõltuvused, konfigureeritakse keskkonda ja lõpuks käivitatakse rakendus.

Rakenduse üleslaadimise protsess automatiseeriti ka Gitiga, mis oli konfigureeritud nii, et Heroku serverisse juurutamise protsess käivitatakse automaatselt, kui kasutatakse „*git push*" operatsiooni.

Rakenduse prototüüp on ligipääsetav ja avalikult kättesaadav järgneval veebiaadressil: https://kv-air-maintenance.herokuapp.com/

8 TULEVASTE ARENDUSTE PLAANID

Rakenduse arendamisel on alati oluline pöörata tähelepanu tulevikule, kaaludes võimalikke uuendusi ja arendusi, mis võivad rakendust paremaks muuta ja kasutajatele rohkem väärtust pakkuda. Allpool on toodud mõned valdkonnad, kus tulevikus võiks toimuda täiustusi ja arendusi.

8.1 Teavituste saatmine

Üheks potentsiaalseks edasiarenduseks oleks teavituste süsteemi lisamine, mis informeeriks kasutajaid eelseisvatest hooldustöödest. Kujutame ette funktsionaalsust, mis iga päev teatud kellajal koguks infot kasutajate kohta, kelle lennukid vajavad lähiajal hooldust. Selle info põhjal saadetakse kõigile vastavatele kasutajatele meilile teated tulevaste hooldustööde kohta.

Selle funktsionaalsuse saaks realiseerida Spring Boot rakenduse laiendamisega või uue teenuse loomisega. Tuleks luua töö, mis näiteks Quartz Java raamistiku abil koguks andmebaasist vajaliku info. [27] Seejärel kasutataks mõnda levinud Java teeki e-kirjade saatmiseks. Üheks selliseks Java teegiks on näiteks Angus-Mail Java teek, mis pakub platvormist sõltumatut raamistikku, et ehitada sõnumeid või meile saatvaid rakendusi. [28]

8.2 Parema kasutajakogemuse loomine

Antud bakalaureusetöö raames valminud kasutajaliides on esialgne ja suunatud esmajoones funktsionaalsusele, mitte visuaalsele atraktiivsusele. Seega võib see vajada edasist arendust, et olla konkurentsivõimeline tänapäeva veebirakenduste seas.

Tulevikus võiks teha täiendavaid uurimusi, et mõista paremini, kuidas kasutajad rakendust kasutavad, ja leida võimalusi, kuidas muuta kasutajaliidest intuitiivsemaks ja kasutajasõbralikumaks. Üks võimalus selle saavutamiseks võiks olla põhjalikum kasutajate tagasiside kogumine ja analüüs.

Veel võiks rakendus toetada mitut keelt. Hetkel on ainuke valik inglise keel, kuid mitmekeelne kasutajaliides suurendaks potentsiaalset klientide arvu ja kasutajasõbralikkust.

8.3 Tehnilised parendused ja jõudluse suurendamine

Vahemällu salvestamine - Kui infot päritakse andmebaasist tihti, oleks oluliselt mõistlikum salvestada see ajutiselt näiteks Java koodis. Selle tulemusena vähendatakse päringute arvu baasist, mis on üks sagedane pudelikael serverrakendustes.

Mikroteenuste suhtlemine läbi GRPC - Seni valminud arenduses suhtlevad teenused omavahel üle REST API. Kui kasutajate arv ja funktsionaalsus suureneb tulevikus, siis vajaks mikroteenuste omavaheline suhtlemine efektiivsemat ühendust, mida võimaldaks GRPC.

Eraldi autentimise teenus - Hetkel kasutatakse kasutajate haldamise teenust autentimiseks. Mikroteenuste arhitektuuri põhimõtetest lähtudes võiks aga kaaluda autentimise eraldamist omaette teenuseks.

KOKKUVÕTE

Käesoleva lõputöö eesmärgiks oli välja töötada veebirakendus, mis aitab optimeerida ja automatiseerida väikeste lennukite hooldusprotsesside haldamist, säilitades samal ajal kasutajamugavuse ja lihtsuse. Projekti käigus loodi täisfunktsionaalne veebirakendus, mis on tõhus, turvaline ja kasutajasõbralik.

Töö teoreetilises osas tutvustati kasutatud tehnoloogiaid ja raamistikke, mis hõlmasid Vue.js 3 ja PrimeVue frontendi, Spring Boot backendi ning PostgreSQL andmebaasi. Turvalisuse ja autentimise tagamiseks kasutati JWT tokenit. Praktilises osas kavandati ja arendati kasutajaliides ning loodi serverrakenduse loogika ja teenused.

Rakenduse juurutamise protsessis valiti Heroku pilveplatvorm, mis võimaldas lihtsat konfigureerimist, paindlikku skaleerimist ja automatiseeritud juurutamist. Rakendus on edukalt üles laaditud Heroku pilve, kasutades automaatset juurutamist GitHubist.

Rakenduse prototüüp on ligipääsetav ja avalikult kättesaadav aadressil järgneval veebiaadressil: https://kv-air-maintenance.herokuapp.com/

Lõputöö tulemusena valmis rakenduse protüüp, mis pakub integreeritud lahendust väikeste lennukite hooldusprotsesside haldamiseks. Rakendus on kasutajasõbralik, ja turvaline, võimaldades efektiivselt hallata väikelennukite kohustuslikke hooldusprotsesse, pakkudes lihtsasti mõistetavat kasutajaliidest kliendile.

Vaadates valminud rakendust antud bakalaureusetöö eesmärgi kontekstis, võib öelda, et üldjoontes on saavutatud soovitud tulemus. Siiski on alati ruumi parandamiseks ja täiustamiseks. Näiteks on vaja edasi arendada kasutajaliidese disaini, lisada uusi funktsionaalsusi ning optimeerida rakenduse jõudlust kasutajate arvu suurenemisel. Lisaks võib tulevikus olla vajalik integreerida rakendust teiste süsteemidega, et pakkuda veelgi terviklikumat lahendust. Kõiki neid aspekte tuleks käsitleda tulevastes uuringutes ja arendusprojektides.

SUMMARY

The aim of this thesis was to develop a web application that helps optimize and automate the management of maintenance processes for small aircraft, while maintaining user-friendliness and simplicity. Throughout the project, a fully functional web application was created, which is efficient, secure, and user-friendly.

The theoretical part of the work introduced the technologies and frameworks used, including Vue.js 3 and PrimeVue for the frontend, Spring Boot for the backend, and PostgreSQL for the database. JWT tokens were used to ensure security and authentication. In the practical part, the user interface was designed and developed, and the server application logic and services were implemented.

During the deployment process, the Heroku cloud platform was chosen, enabling easy configuration, flexible scalability, and automated deployment. The application has been successfully deployed to Heroku using automated deployment from GitHub.

The prototype of the application is accessible and publicly available at the following web address: https://kv-air-maintenance.herokuapp.com/

As a result of this thesis, an application was developed that provides an integrated solution for managing maintenance processes of small aircraft. The application is user-friendly, secure, and allows for efficient management of mandatory maintenance processes, offering a user-friendly interface for the clients.

Considering the objective of this bachelor's thesis, it can be said that the desired outcome has been achieved overall. However, there is always room for improvement and enhancement. For example, further development of the user interface design, addition of new functionalities, and optimization of application performance with an increasing number of users are needed. Additionally, in the future, integration with other systems may be necessary to provide a more comprehensive solution. All these aspects should be addressed in future research and development projects.

KASUTATUD KIRJANDUSE LOETELU

[1] S. Clarke, "Measuring API Usability," 2004.

[2] "Git Guide," [Online]. Available: https://github.com/git-guides.

[3] „GitHub," [Online]. Available: https://github.com/about.

[4] "GRPC," [Online]. Available: https://grpc.io/about/. [Accessed 2023].

[5] R. Fielding, M. Nottingham and J. Reschke, "HTTP Semantics," [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9110. [Accessed 2023].

[6] "JWT.io," [Online]. Available: https://jwt.io/introduction/.

[7] B. Butler, "Networkworld, PaaS," [Online]. Available: https://www.networkworld.com/article/2163430/paas-primer--what-is-platform-as-a-service-and-why-does-it-matter-.html.

[8] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," 2000.

[9] "Amazon, What is SQL?," [Online]. Available: https://aws.amazon.com/what-is/sql/.

[10] "AKIT URL," [Online]. Available: https://akit.cyber.ee/term/3138.

[11] J. Hessburg, "What's This 'A' Check, 'C' Check Stuff?," *Aviationpros,* no. https://www.aviationpros.com/aircraft/article/10388655/whats-this-a-check-c-check-stuff.

[12] I. Alomar and I. Yatskiv, "Digitalization in aircraft maintenance processes," 2023.

[13] "Easy Access Rules for Continuing Airworthiness (Regulation (EU) No 1321/2014)," [Online]. Available: https://www.easa.europa.eu/en/document-library/easy-access-rules/online-publications/easy-access-rules-continuing-airworthiness.

[14] M. Fowler, Patterns of Enterprise Application Architecture.

[15] L. Chen, "Microservices: Architecting for Continuous Delivery and DevOps".

[16] I. L. Nicola Dragoni, "Microservices: How To Make Your Application Scale," 2017.

[17] "VueJS," [Online]. Available: https://vuejs.org/guide/introduction.html. [Accessed 2023].

[18] "Primevue," [Online]. Available: https://github.com/primefaces/primevue.

[19] "Veutify Labs," [Online]. Available: https://vuetifyjs.com/en/labs/introduction/. [Accessed 2023].

[20] "Spring Cloud Gateway Documentation," [Online]. Available: https://docs.spring.io/spring-cloud-gateway/docs/current/reference/html/.

[21] "Medium, Benchmarking REST vs GRPC," [Online]. Available: https://medium.com/sahibinden-technology/benchmarking-rest-vs-grpc-5d4b34360911.

[22] "PostgreSQL," [Online]. Available: https://www.postgresql.org/about/.

[23] "Flyway, Why Flyway?," [Online]. Available: https://flywaydb.org/documentation/getstarted/why.

[24] "PostgreSQL Indexing," [Online]. Available: https://www.postgresql.org/docs/current/indexes.html.

[25] "Heroku About," [Online]. Available: https://www.heroku.com/what.

[26] "Heroku CLI," [Online]. Available: https://devcenter.heroku.com/articles/heroku-cli.

[27] "Quartz Scheduler," [Online]. Available: https://www.quartz-scheduler.org/.

[28] "JavaMail API," [Online]. Available: https://www.oracle.com/java/technologies/javamail.html.