TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Annigrete Suimets 206743IABB

Ettevõttesisese riistvara haldussüsteemi kui mikroteenuse arendamine

Bakalaureusetöö

Juhendajad: Jelena Vendelin PhD

Karl Hendrik
Leppmets BSc

Tallinn 2023

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Annigrete Suimets

16.05.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on arendada mikroteenus ettevõtte riistvaraliste varade haldamiseks. Arendatud süsteemis saab lisaks lisamise, kustutamise ning muutmise funktsioonidele näha kõikide objektide nimekirja. Kasutaja saab nimekirja ka filtreerida väga detailsel tasemel. Samuti on näha kõiki süsteemis tehtud muudatusi, mis on seotud riistvara objektiga. Süsteemi abil saab hallata ka varade laoseisu. Täpsemalt, süsteemiadministraator saab märguande, kui teatud tüüpi objekti laoseis on allapoole etteantud kogust. Rakendusega saab kontrollida ka riistvara objektide omadusi. Nimelt, kas esineb identsete seerianumbritega objekte või kas leidub kasutajaid, kelle kasutuses on rohkem kui üks samasuguse objektitüübiga seadet.

Töö käigus uurib autor kahte olemasolevat lahendust ning nende võimalusi, pidades silmas ettevõttele vajalikke nõudeid.

Autor annab ka ülevaate nii arenduseks vajaminevatest tööriistadest kui ka rakenduse arendamise protsessidest. Samuti sisaldab töö ülevaadet rakenduse komponentidest ning nende arhitektuurist.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 6 peatükki, 16 joonist, 3 tabelit.

Abstract

Development of An In-House Inventory Management System as a Microservice

The purpose of this Bachelor's thesis is to develop a microservice for managing company’s hardware assets. The system allows users to view a list of all objects in addition to the functionality for adding, deleting, and modifying objects. Users can filter the list at a very detailed level. The application enables the tracking of all changes made to hardware objects within the system. In addition, the system will assist with inventory management, meaning the system administrator will get a notification if the count of the items in storage of a certain asset type is below the predetermined amount. Additionally, the application can be used to verify the properties of hardware objects, more specifically whether there are objects with identical serial numbers or users with more than one asset of the same type.

During the course of this work, the author analysed two existing solutions and their capabilities, taking into account the requirements of the company. The author also provides an overview of the necessary development tools and the process of developing the application. Moreover, the work includes a summary of the application's components and their architecture.

The thesis is in Estonian and contains 31 pages of text, 6 chapters, 16 figures, 3 tables.

Lühendite ja mõistete sõnastik

Admin	Administraator
API	*Application Programming Interface*, tarkvaravahendaja, mis võimaldab kahel rakendusel omavahel suhelda
Asset	Aineline, füüsiline vara
Auth	*Authentication*, autentimine
Commit	Versiooni kontrolli süsteemi operatsioon, mis saadab viimased koodimuudatused lähtekoodi repositooriumisse
CRUD	*Create, Read, Update and Delete*, programmeerimisel kasutatav tarkvara arhitektuuri stiil
HR	*Human Resources*, personaliosakonna töötaja
HTTP	*Hypertext Transfer Procotol*, sõnumipõhine keel, mida kasutades saavad arvutid omavahel interneti kaudu suhelda
IMS	*Inventory Management System*, varahaldussüsteem
IT	Infotehnoloogia
JavaScript	Programmeerimiskeel veebilehtede skriptimiseks
JSON	*JavaScript Object Notation*, andmevahetusformaat
PHP	Avatud lähtekoodiga üldotstarbeline skriptikeel, mis sobib hästi veebiarenduseks ja mida saab manustada HTML-i
ReactJS	JavaScripti raamistik kasutajaliideste loomiseks
REST	*Representational State Transfer*, arhitektuuri stiil HTTP protokolli kaudu suhtlevate veebiteenuste loomiseks
SPOAP	*Scoro People Operations Automation Platform*
SQL	*Structured Query Language*, päringukeel andmebaasiga suhtlemiseks
Symfony	PHP veebirakenduste raamistik, mis sisaldab korduvkasutatavaid komponentide komplekte
Template	Mall, üldine vorm, mudel
WMS	*Warehouse Management System*, laohaldussüsteem

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva bakalaureusetöö peamine eesmärk on arendada ettevõttele Scoro Software OÜ varade haldussüsteem, *Asset Service,* kui mikroteenus. Eesmärgi saab jagada alameesmärkideks. Nendeks on: anda ülevaadet ettevõttes kasutatavate riistvara objektide üle, näha nende kasutuse ajalugu ning teavitada IT personali, kui kindlat tüüpi objekti laoseis on langenud alla määratud piiri.

Antud töö sisaldab ülevaadet rakenduse arendusprotsessidest ning kasutatud tööriistadest, tehnoloogiatest ja arhitektuurist. Samuti on töös analüüsitud kahte olemasolevat laohaldussüsteemi, nende funktsioone ning sobivust projektiks ettenähtud nõuetega.

1.1 Taust ja probleem

Ettevõte Scoro Software OÜ (edaspidi Scoro) on aastatega kiiresti kasvanud töötajate arvu poolest. Sellega on kaasnenud ka töötajate poolt kasutatava riistvara kasv. Nii ettevõtte IT-personalile kui ka finantsosakonnale on oluline, et oleks olemas täielik ülevaade seadmete kasutajatest, ajaloost ning asukohtadest. Siiamaani on Scoros kasutatud ettevõtte enda poolt arendatavat tööhaldustarkvara, kus *Asset* mooduli kaudu on olemas ülevaade kõikidest riistvara objektidest, neid kirjeldavatest omadustest ja nende praegusest kasutajast. Samuti on võimalus filtreerida nimekirja nii kasutaja kui ka kõikide objekte kirjeldatavate omaduste alusel.

Samas aga puudub praeguse lahenduse puhul piisavalt detailne ülevaade riistvara objektide - arvutid, klaviatuurid, monitorid, arvutihiired, kõrvaklapid, võrguseadmed - haldamiseks. Näiteks on IT personalile oluline filtreerida kasutuses olevaid arvuteid kindla operatiivmälu mahu järgi, mida praegune lahendus ei võimalda. Lisaks ei ole ülevaadet objektide kõikidest eelnevatest kasutajatest ning staatustest, mis näitavad, kas objekt on kunagi olnud katki, paranduses või kadunud.

Töö raames analüüsitakse ühe Eesti ja ühe välismaise ettevõtte pakutavaid tarkvaralahendusi ja nende sobivust ettevõtte nõuetega. Kuid kuna töö eesmärk on täiendada juba varasemalt ettevõttesiseseks kasutuseks arendatud rakendust, mida

kasutab ja arendab *Internal* IT osakond, mikroteenusega, siis juba olemasolevad tarkvaralahendused ei rahulda ettevõtte tänaseid nõudmisi ja ei ole piisavalt paindlik.

1.2 Ülesande püstitus ja eesmärk

Töö eesmärk on arendada veebirakendus ettevõtte riistvarade - arvutid, monitorid, klaviatuurid, hiired, kõrvaklapid ning võrguseadmed - haldamiseks, mis annab selge ülevaate nende hetkesest laoseisust. Lisaks on iga seadme puhul näha selle täpne ajalugu, mis kuvab nii praegu kehtivaid kui ka varasemaid staatuseid ja omanikke. Samuti soovib autor ehitada funktsiooni, mille ülesanne on teavitada, kui mingi riistvaralise objekti laoseis on allpool eelnevalt määratud soovitud piiri. Uue töötaja tulekul määratakse talle süsteemi kaudu seadmed, mida ta kasutama hakkab. Vara seotakse süsteemis konkreetse inimesega ning juhul kui seadmed võeti laost, mitte ei tellitud spetsiaalselt uusi seadmeid, siis muutub ka laoseis. Kui seadme laoseis läheb allapoole eelnevalt määratud piiri, siis annab rakendus sellest märku. Samuti arendatakse süsteemile kaks täiendavat funktsiooni, mille ülesanneteks on teha süsteemis olevatele varadele kontroll. Täpsemalt, esimene funktsioon kontrollib, kas süsteemis on varasid, millel on identne seerianumber, ning teine kontrollib, kas leidub töötajaid, kelle kasutuses on rohkem kui üks sama vara tüübiga objekti. Arendatavate funktsioonidega saab automatiseerida tegevusi, mida seni on ettevõttes tehtud manuaalselt.

2 Taustauuring

Varude haldamine sisaldab ettevõtte varude tellimist, ladustamist, kasutamist ja müümist. See hõlmab toorainete, komponentide ja valmistoodete haldamist, samuti esemete ladustamist ja töötlemist. Varude haldamine oluline igas suuruses ettevõtetele. Tähtis on teada, millal varusid täiendada ning milliseid koguseid osta. Ettevõtte varud on ühed väärtuslikumad varad ning nende puudus võib ettevõttele mõjuda. [1]

2.1 Võrdlus olemasolevate lahendustega

Järgnevates alapeatükkides võrreldakse kahte olemasolevat varahaldustarkvara lahendust. Vaatluse alla on võetud välismaise ettevõtte NetSuite ja Eesti päritolu ettevõtte Astro Baltics poolt pakutavad lahendused. Analüüsi käigus vaadatakse mõlema ettevõtte poolt pakutavat funktsionaalsust, nende positiivseid ja negatiivseid omadusi.

2.1.1 NetSuite Inventory Management

NetSuite on ettevõtte Oracle Corporation Ameerika Ühendriikides tegutsev tütarettevõte, mille laohalduse süsteem pakub ühtset reaalajas vaadet laoseisust kõigis asukohtades. Varude taseme optimeerimine ja toodete kättesaadavuse tagamine mitmes kanalis aitab *NetSuite Inventory Management* tarkvara ettevõtetel hoida laokulusid madalal. [2]

Tabelis 1 on välja toodud NetSuite positiivsed küljed.

Tabel 1. Ettevõtte NetSuite tarkvara positiivsed küljed.

Positiivsed omadused	**Kirjeldus**
Üldnimekiri	Antud süsteemiga on võimalik saada ülevaadet laoseisust asukoha põhiselt. Lisaks on olemas numbriline ülevaade, mitu toodet on laos, määratud asukohas.

Laoseisu aruanne	Süsteem teavitab, kui objekti laoseis on langenud alla poole määratud kogust.
Filtreerimine	Süsteemis on võimalik filtreerida objekti seerianumbri järgi.
Kustutatud objektide vaade	On olemas ülevaade objektidest, mis on kustutatud. Seeläbi on olemas ülevaade kõikidest objektidest, mis ettevõttes on kasutatud.
Detailne vaade	Saab vaadata kogu ühe spetsiifilise objekti kohta käivat informatsiooni.

2.1.2 NOOM laotarkvara

NOOM on ettevõtte Astro Baltics välja töötatud mitmekülgne majandustarkvara, mis sobib igas suuruses ettevõttele ressursside juhtimiseks, planeerimiseks ning analüüsimiseks. Üheks NOOM-i mooduliks on laotarkvara (WMS/IMS), mis on antud analüüsis vaatluse all. Antud mooduliga saab ettevõte hallata nii lao kui ka kogu ettevõtte riistvara. [3]

Tabelis 2 on välja toodud NOOM-i eelised.

Tabel 2. Ettevõtte Noom laotarkvara positiivsed küljed.

Positiivsed omadused	**Kirjeldus**
Inventuur erinevat tüüpi objektidele	Antud tarkvara võimaldab teha inventuuri eri objektidele, eri kaubagruppidele ning filtreeritud kaupadele.
Üldnimekiri	Tarkvara võimaldab näha üldnimekirja kõikidest toodetest.
Laojäägi aruanne	Peale inventuuri koostamist ja üldnimekirja koostamist on võimalik näha välja laojäägi aruanne. See sisaldab

	kaupade jääki laos ning võimaldab filtreerida negatiivse jäägiga kaupu.
Määratud miinimum kogus	Programmis on võimalik määrata kaupade miinimum kogused. Pärast miinimum koguse määramist näitab aruanne ka tooteid, mille jääk on alla määratud miinimumkoguse.
Kasutajate ligipääsude piiramine	Varade halduse õigust ja andmete nägemist on võimalik piirata kasutaja õigustega.

2.2 Analüüsi kokkuvõte

Tabelisse 3 koondatud negatiivsed küljed on funktsioonid, mis on antud projekti tegemiseks vajalikud, kuid puudulikud mõlemal eelnevalt analüüsitud tarkvaral [2], [3].

Tabel 3. Ettevõtete Noom ja NetSuite ühised negatiivsed küljed.

Negatiivsed omadused	**Kirjeldus**
Puuduvad kasutajad	Ei ole võimalik lisada kasutajaid ning neid siduda seadmetega, mida nad kasutavad.
Detailse filtreerimise puudumine	Antud tarkvarade puhul ei ole võimalik filtreerida ettevõttele vajaliku detailsusega. Näiteks ei saa filtreerides otsida kõiki arvuteid operatiivmälu suuruse alusel.
Puuduvad väljad, mis kirjeldavad objekti staatusi	Seadme kirjeldamise puhul ei ole staatusi kirjeldavad välju. Antud projekti juures on vajalik näha finants-, lao- ja asukoha staatust. Samuti ei ole võimalik neid ise juurde lisada.
Puuduvad rollid ning seadme komplektid	Puudub võimalus lisada rolle ning määrata neile vastavaid seadmete komplekte, mis sisaldavad kindlaid ettemääratud riistvaralisi objekte.

Üleliigsed funktsioonid	Tarkvarade puhul on palju üleliigseid funktsioone, mille eest tuleks siiski tasuda. Nendeks on laodokumentite haldamine, omahinna arvestus ning varustamine.

3 Metoodika

3.1 Projekti detailne kirjeldus

Projekti eesmärk oli arendada laohaldussüsteem, mis võimaldaks omada detailset ülevaadet kõikidest riistvaralistest objektidest, mis ettevõttes kasutatakse. Lõputöö raames arendati *Asset* mikroteenus, mis on osa sisehaldustarkvarast SInMan ehk Scoro *Internal Manager*. Mikroteenuste arhitektuuri stiil, mis struktureerib rakenduse erinevate teenuste kogumina, võimaldab pakkuda keerulisi lahendusi kiiresti, usaldusväärselt ning sageli [4]. Iga teenus keskendub kindlale objektile või funktsioonile ning sisaldab vaid antud osale vajalikke funktsioone. Niimoodi saab teha muudatusi keerulistes süsteemides ilma, et see mõjutaks teiste teenuste toimimist.

Lisaks ülevaate omamisele, andmete lisamisele, kustutamisele ning muutmisele täidab süsteem kolme järgnevat suuremat funktsiooni. Esmalt peab süsteem salvestama kogu objektiga seotud ajaloo. See väljendub selles, et iga objekti juures on näha, mis omadust on muudetud, mis oli eelmine omaduse väärtus ning milline on uus väärtus. Niimoodi on võimalik näha seadme kõiki kasutajaid ning milliste kasutajate vahel seade on liikunud. Samuti saab näha ülevaadet seadme staatuste muutustest ning sellest, kas seade on mingil hetkel olnud katki. Lisaks väärtustele salvestatakse ka muudatuse tegemise kuupäev, mis võimaldab omada veelgi detailsemat ülevaadet.

Teiseks funktsionaalsuseks on laoseisu haldamine, mille ülesanne on teavitada, kui mingi riistvaralise objekti laoseis on allpool eelnevalt määratud soovitud piiri. Kasutaja määrab, millise objekti tüübi kohta ta laoseisu näha tahab. Seejärel võrdleb süsteem vastava objektitüübi laoseisu nõuet reaalse seisuga. Nimelt võrreldakse laos olevate objektide kogust, mille *storageStatus* ID väärtus on võrdne *inStorage* väärtuse ID-ga, nõudes määratud koguse arvuga. Juhul kui laos olevate objektide kogus on väiksem, siis saadetakse süsteemis admin rolli kandvale kasutajale teavitav meil. Meili saatmine toimub *Email Service* teenuse kaudu, mida *Asset Service* kutsub välja, kui vastav olukord tekib.

Kolmandaks on võimalik süsteemis viia läbi kontrolle. Esimene neist kontrollib, kas süsteemis leidub objekte, millel on samasugune seerianumber. Antud kontroll võimaldab leida vead kiiresti ning seeläbi vähendab segaduse tekkimise tõenäosust. Teise funktsionaalsusega on võimalik tuvastada töötajad, kelle käes seadmeid, millel on sama objektitüüp. Seeläbi on võimalik tuvastada töötajad, kellel on käes mitu arvutit. Niimoodi saab ettevõtte varasid optimaalselt kasutada.

3.2 Tööriistade kirjeldus

SInMan rakenduse arendamiseks kasutati mitmeid erinevaid tarkvaraarenduse platvorme ning erinevaid programmeerimise keeli. Tööriistade valiku tegemisel võeti arvesse nii autori tööpoolse juhendaja eelistusi kui ka juba ettevõttes varasemalt kasutuses olevaid vahendeid.

Front-end projekti arendamiseks kasutati Microsoft Visual Studio Code-i ning ReactJS raamistikku. Kõik komponendid ja vajalikud navigeerimised veebilehel kirjutati kasutades JavaScripti. Samuti kirjutati JavaScriptis vajalikud koodiread andmebaasiga ühenduse saamiseks, et andmebaasist andmeid kätte saada ning veebilehel kuvada.

Back-end arendamiseks kasutati JetBrains väljatöötatud arendamistarkvara PhpStorm ning raamistikuks valiti Symfony. Programmeerimiskeeleks oli PHP (*PHP: Hypertext Preprocessor)*. *Back-end* projekt sisaldab kogu andmetega seotud loogikat ning suhtlust andmebaasiga. Andmebaasi valikul otsustati MySQL kasuks. Selle haldamiseks kasutas autor Admineri.

Back-end testimiseks kasutati Postmani, mis võimaldab teha koostööd API-dega. Postmani abil kontrolliti, kas ühendus andmebaasiga on olemas ning kas kõik API funktsioonid töötavad nii nagu on ettenähtud ehk kas kõik brauserisse sisse kirjutatud aadressid tagastavad õiget informatsiooni.

Koodi hoidmiseks kasutati keskkonda GitLab. Seal saab vaadata kõiki *commite*. Igale *commitile* prooviti panna võimalikult täpne nimi, et oleks arusaadav, mis antud hetkel koodis tehti.

Projekti ning seda sisaldavate ülesannete haldamiseks kasutati Confluence keskkonda. Sinna kirjutati üles kõik ülesanded, mis tuleb teha, täpsustati ka nende tegemise järjekord. Lisaks sai näha, kuidas edeneb mingi ülesande tegemine ning osade puhul täpsustati ka kuupäev, mis ajaks antud ülesanne peab valmis olema, et püsida graafikus (Lisa 2).

Jooniste tegemiseks kasutati Visual Paradigm tööriista. See sisaldab hõlpsasti kasutatavat veebipõhist joonistustööriista, millega tehti valmis antud projekti puudutavad diagrammid – kontekstidiagramm, kasutusjuhtude diagramm.

Töötunnid logiti kasutades Scoro tööhaldustarkvara. Antud arenduse jaoks koostati keskkonnas eraldi projekt, mille alla logiti kogu aeg, mis kulus projekti arendamiseks. Kuna enamasti logiti kindla funktsionaalsusega seotud aega sama sündmuse alla, siis on olemas ka ülevaade, kui palju aega kulus kindla etapi arendamiseks.

3.3 Tööprotsessi kirjeldus

Projekti idee pakuti välja autori tööpoolse lõputöö juhendaja poolt. Vajadus taolise rakenduse järgi oli antud meeskonnas, kus töötab nii töö autor kui ka autori lõputöö praktilise osa juhendaja, suur. Projekti tegemisega alustati 23-ndal jaanuaril 2023, millele eelnes ühe kuupikkune periood, mille jooksul läbis autor kaks Udemy keskkonnas pakutavat kursust. Antud kursused sisaldasid nii PHP keele kui ka Symfony raamistiku õppimist [5], [6]. Kursused olid vajalikud antud projekti jaoks, sest autor ei olnud varasemalt kokku puutunud nii PHP keele ega Symfony raamistikuga.

Projekti ehitati üles järgides klassikalist tarkvaraarenduse metoodikat. Nimelt jagati arenduse protsess erinevatesse etappidesse [7]. Esmalt tuli kaardistada antud süsteemi vajadused ning selle põhjal luua ülevaatlikud diagrammid. Lisaks tuli välja selgitada, milliseid klasse on vaja, millised on nende omadused ning millised on erinevate klasside vahelised relatsioonid. Antud protsessi käigus koostati nii kasutusjuhtude diagramm kui ka konteksti diagramm. Kasutusjuhtude diagrammi tegemisel tuli kirja panna peamised tegevused, mida süsteemis teha saab. Seejärel tuli mõelda rollidele ja ligipääsudele, sest antud süsteem on piiratud juurdepääsuga olenevalt isiku rollist. Konteksti diagrammi tegemine oli vajalik, et panna paika, milliste teiste mikroteenustega arendatav teenus suhtlema hakkab.

Teine etapp koosnes *back-end* koodi kirjutamisest. Koostati vajalikud klassid, lisati nõutud omadused ning relatsioonid. Seejärel kirjutati igale klassile API *endpointid*, mis täidavad CRUD funktsioone. Järgmiseks lisati süsteemile filtreerimise funktsioon, mis võimaldas filtreerida riistvaralisi objekte kõikide tema omaduste järgi. Samuti arendati süsteemile kaks kontrolliva ülesandega funktsiooni. Esimese eesmärk on tuvastada objekte, millel esineb sama seerianumber. Teise funktsionaalsuse ülesanne on tuvastada isikud, kelle kasutuses on rohkem kui üks sama objektitüübiga seadet. Järgmiseks arendati *Asset History* funktsionaalsus, mis võimaldab näha objektidega seotud ajalugu.

Kolmas etapp sisaldas peamiselt *front-end* arendust. Kuna osa tabeleid koondati ühte vaatesse, siis ei olnud vajalik teha igale ühele eraldi vaateid, seega tehti valmis vaated kõikidele autori poolt välja valitud objektidele. Vaateid tehti kahte erinevat moodi. Esmalt tehti nimekirja vaade (Lisa 3), kus on näha kõiki objekte, ning seejärel tehti valmis detailsed vaated, kus on näha kõiki detaile, mis puudutavad kasutaja poolt valitud kindlat objekti (Lisa 4). Samuti tehti valmis ka vormid, mis kuvatakse kasutajale, kui toimub uue objekti lisamine või valitud objekti kustutamine (Lisa 5, Lisa 6). Objekti lisades saab määrata objekti tüübi, mille valimisel avanevad väljad, kuhu saab kirjutada detailsemad väärtused (Lisa 7). Väljad, mis avanevad kindla tüübi puhul määratakse *Asset Type* vaates (Lisa 8, Lisa 9).

Neljandas etapis ühendati *Asset* teenus SPOAP *(Scoro People Operations Automation Platform)* teenusega, kus päriti andmed nii kasutajate, asukohtade kui ka rollide kohta. Andmete pärimine toimub *back-endis*, kus pannakse kokku nii *Asset* andmebaasis olevad andmed kui ka SPOAP teenusest päritud andmed.

Viiendas etapis arendati laoseisu haldamise funktsionaalsus. Arendati funktsioon, mis pärib andmebaasist andmed kasutaja poolt määratud objekti tüübi kohta. Vastava tüübi kohta saadakse nii vajalik kogus *warehouseRequirement* tabelist kui ka tegelik laoseis *asset* tabeli põhjal.

Koodi kirjutamisel kasutas autor domeenipõhist disaini (*Domain-Driven Design*). Domeenimudel loob omavahel relatsioonis olevate objektide võrgu, kus iga objekt esindab mõnda tähendusrikast üksust, näiteks ettevõte või üks rida tellimusel [8]. Antud projektis on kasutatud domeenipõhise disaini *Aggregate* mustrit. See on domeeniobjektide kogum, mida saab käsitleda ühe üksusena [9]. Näiteks võib olla

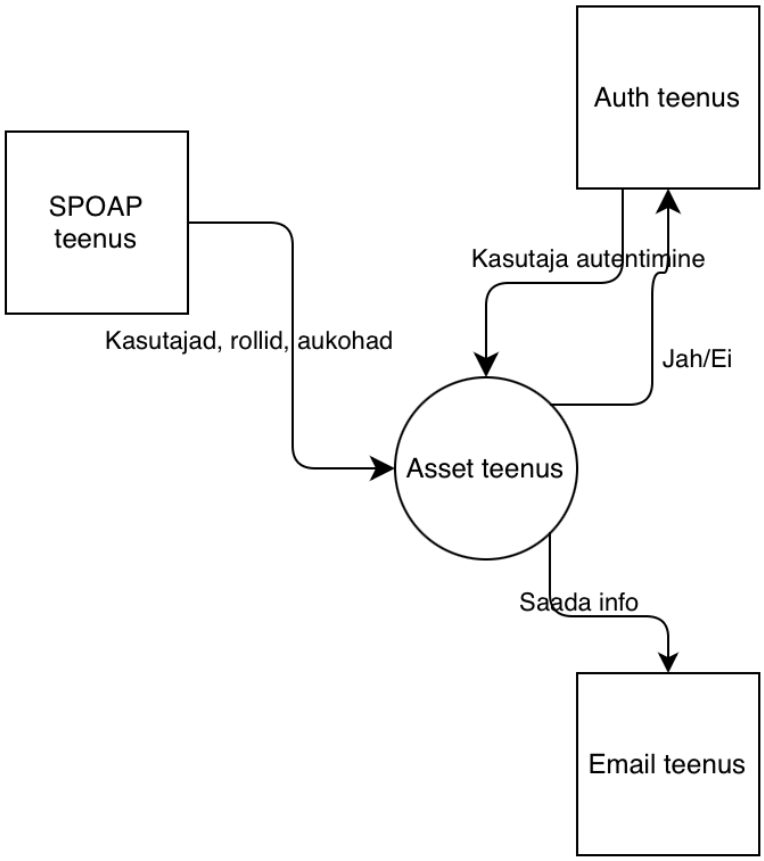
tellimus ja selle reaüksused, mis on eraldi objektid, kuid tellimust koos selle reaüksustega on kasulik käsitleda ühe koondina [9]. Samamoodi on antud töö raames arendatud rakendusega, kus keskmes on *asset* objekt, millel on kirjeldavad omadused nagu tüüp, laoseisu staatus ja asukoht, kuid andmebaasi ja koodi tasemel on tegu eraldi objektidega.

4 Tulemused

4.1 Funktsionaalsed nõuded

4.1.1 Konteksti diagramm

Konteksti diagramm näitab süsteemi andmeid lisavaid lähtesüsteeme, samuti peamisi kasutajakomponente ja toetatud allavoolu infosüsteeme [10]. Joonisel 1 on välja toodud lõputöö raames arendatud süsteemi, *Asset Service*, konteksti diagramm. *Asset* mikroteenus suhtleb kokku kolme mikroteenusega. *Auth service* kaudu toimub kasutajate autentimine. SPOAP mikroteenuse kaudu saab *Asset* teenus andmed ettevõttega seotud töötajate, rollide ning asukohtade kohta. *Email Service* kaudu toimub süsteemi administraatori teavitamine. Andmed on JSON formaadis.



Joonis 1. Konteksti diagramm.

4.1.2 Rollid ja tegevused

Antud rakenduses on juurdepääsud ja tegevused piiratud rollide alusel. Rolle süsteemis on kolm: admin, HR ning tavakasutaja.

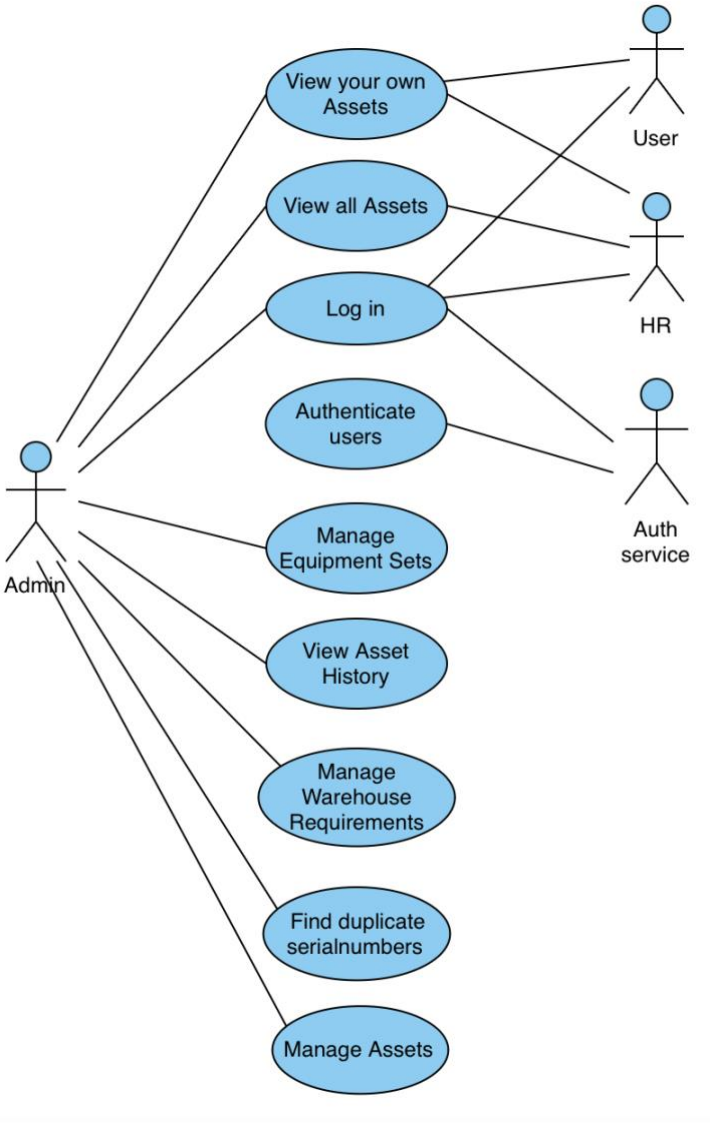
Joonisel 2 on esitatud antud lõputöö raames arendatava projekti kasutusjuhtude diagramm. Kasutusjuhtude diagrammi ülesanne on kirjeldada süsteemi funktsionaalset käitumist [11]. Kasutusjuhud kirjeldavad tegevusi, mida kasutaja süsteemis teha saab. Iga tegevuse juures tuuakse välja, mis rollid antud tegevusi teha saavad. Kasutusjuhud annavad ülevaate, kuidas ning milliste tegevuste juures tuleb rollide õigusi süsteemis piirata. [12]

Kõige vähem on õigusi antud süsteemis tavakasutajal, kellel on vaid õigus sisse logida ning näha kasutaja enda kasutuses olevaid varasid (Joonis 2).

HR ehk personaliosakonna töötaja puhul on ligipääs piiratud ning ainult kindlad tegevused on lubatud. Nimelt saab antud rolliga töötaja süsteemi sisse logida, näha nii HR enda nimel olevad objekte kui ka näha tervet nimekirja kõikidest ettevõtte riistvara objektidest (Joonis 2).

Kõige rohkem on õigusi admin rolli omaval isikul. Admin rolli all on mõeldud IT personali, kes haldab andud süsteemi. Antud rollil on juurdepääs kõikjale ning samuti on neil luba teha kõiki süsteemis olevaid tegevusi. Lisaks eelnevalt mainitud tavakasutaja ning HR rollile, saab admin veel hallata rollipõhiseid riistvara komplekte, näha seadmeid puudutavat ajalugu ning hallata erinevate seadmete laoseisu nõudeid. Samuti saab admin riistvara objekte. Lisaks eelnevalt mainitud rollidele toimub ka suhtlus teise mikroteenusega, mille eesmärk on süsteemi ligipääsu sooviv kasutaja autentida (Joonis 2).

Kasutajate ligipääsud on piiratud mitmel tasemel. *Front-endis* on piiratud ligipääs erinevatele funktsioonidele, mis tähendab, et kasutaja ei näe leheküljel kõiki nuppe, mistõttu ei saa neile ka klikkida. Lisas 10 on näidatud, kuidas on kasutaja eest peidetud lisamise nupp. *Back-endis* on kasutajale piiratud andmete nägemine. Lisas 11 on näha, kuidas on piiratud *Asset History* klassi andmete nägemine.



Joonis 2. Kasutusjuhtude diagramm.

4.1.3 Kasutuslood

Järgnevalt on välja toodud põhilised kasutuslood. Kasutuslugu on süsteemi omaduse või funktsionaalsuse kirjeldus, mis on kirjutatud kõnekeeles. Niimoodi tagatakse, et ka projekti klient või süsteemi kasutajad oleksid võimelised omalt poolt esitama ootuseid süsteemi funktsionaalsuste ja omaduste osas. Lisaks muudab see loodava süsteemi nõuded arusaadavamaks, sest nendest arusaamine ei vaja IT-erialaseid teadmisi. [13]

Kasutuslood: Sisselogimine

Tegutsejad: Admin, HR, Tavakasutaja

Lühikirjeldus: Kasutaja saab süsteemi sisse logida, et näha sealseid andmeid ning teha tema rollile lubatud tegevusi.

Kasutuslood: Ülevaade enda riistvarast

Tegutsejad: Admin, HR, Tavakasutaja

Lühikirjeldus: Kasutaja saab vaadata nimekirja, kus on ülevaade kõikidest riistvara objektidest, mis on talle kasutamiseks määratud. Admini ja HR rollis olles leiab antud nimekirja filtreerides kasutaja järgi.

Kasutuslood: Kõikide riistvara elementide ülevaade

Tegutsejad: Admin, HR

Lühikirjeldus: Kasutaja saab vaadata nimekirja, kus on ülevaade kõikidest riistvara objektidest.

Kasutuslood: Objekti ajaloo vaatamine

Tegutsejad: Admin

Lühikirjeldus: Admin saab iga riistvara objekti juures näha sellega seotud ajalugu: kes on olnud kasutajad, kas seade on olnud katki ning kas ta on käinud paranduses.

Kasutuslood: Rollipõhise riistvara komplekti loomine

Tegutsejad: Admin

Lühikirjeldus: Kasutaja loob uue komplekti. Seejuures on tal vaja anda sellele nimi. Samuti on vaja määrata, mis tüüpi objektid sinna kuuluvad ning millised on nende objektide alamtüübid ja nende väärtused. Antud komplekt seotakse kindla rolliga.

Kasutuslood: Riistvara objekti loomine

Tegutsejad: Admin

Lühikirjeldus: Admin loob uue objekti, mille puhul määrab ta objektile kasutaja, tüübi ning staatuse, mis näitab, kas objekt on terve. Samuti määrab asukoha, laos oleku staatuse,

finantsilise staatuse ja inventari staatuse. Admin lisab veel ka vöötkoodi, seerianumbri, ostukuupäeva ning ka arve numbri.

Kasutuslood: Laoseisu nõude loomine

Tegutsejad: Admin
Lühikirjeldus: Kasutaja loob uue nõude, mille puhul on vaja määrata, millise riistvara objekti ning selle alamtüübi kohta nõue kehtib. Samuti tuleb määrata riik, mille kohta nõue käib.

Kasutuslood: Seerianumbrite duplikaatide otsimine

Tegutsejad: Admin
Lühikirjeldus: Admin saab läbi viia kontrolli, mille käigus tuleb välja, kas süsteemis esineb seadmeid, millel on samasugune seerianumber.

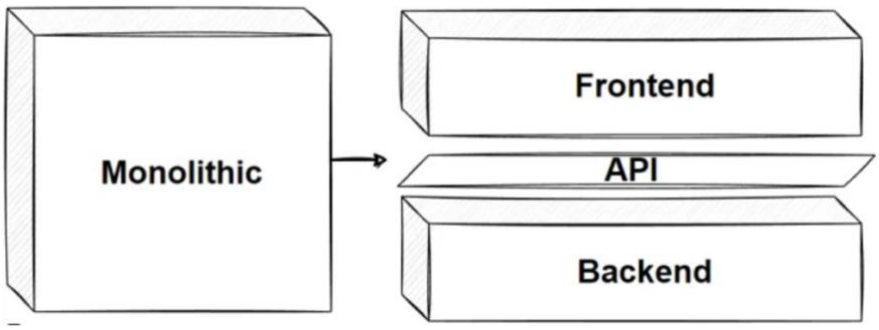
4.2 Arhitektuur

4.2.1 Ülevaade

Arendatud projekti arhitektuur koosneb kolmest suuremat komponendist – MySQL andmebaas, *back-end* projekt ning *front-end* projekt.

Arendamisel kasutati *Headless* arhitektuuri kontseptsiooni, mis on üks lähenemistest mikroteenuste arhitektuuri puhul. See on tarkvaraarenduse kontseptsioon, mis eraldab *front-endi* ja *back-endi* üksteisest. See tähendab, et neid arendatakse ja hooldatakse iseseisvalt ning nad suhtlevad üksteisega API-de kaudu. Sel viisil arendades on rakendus paindlikum ning arendusprotsess on agiilsem. [14]

Joonisel 3 on näha erinevus monoliitse rakenduse, kus kõik komponendid on koos, ja mikroteenuse arhitektuuri vahel, kus komponendid on eraldatud.



Joonis 3. Erinevus monoliitse rakenduse ja mikroteenuse arhitektuuri vahel [14].

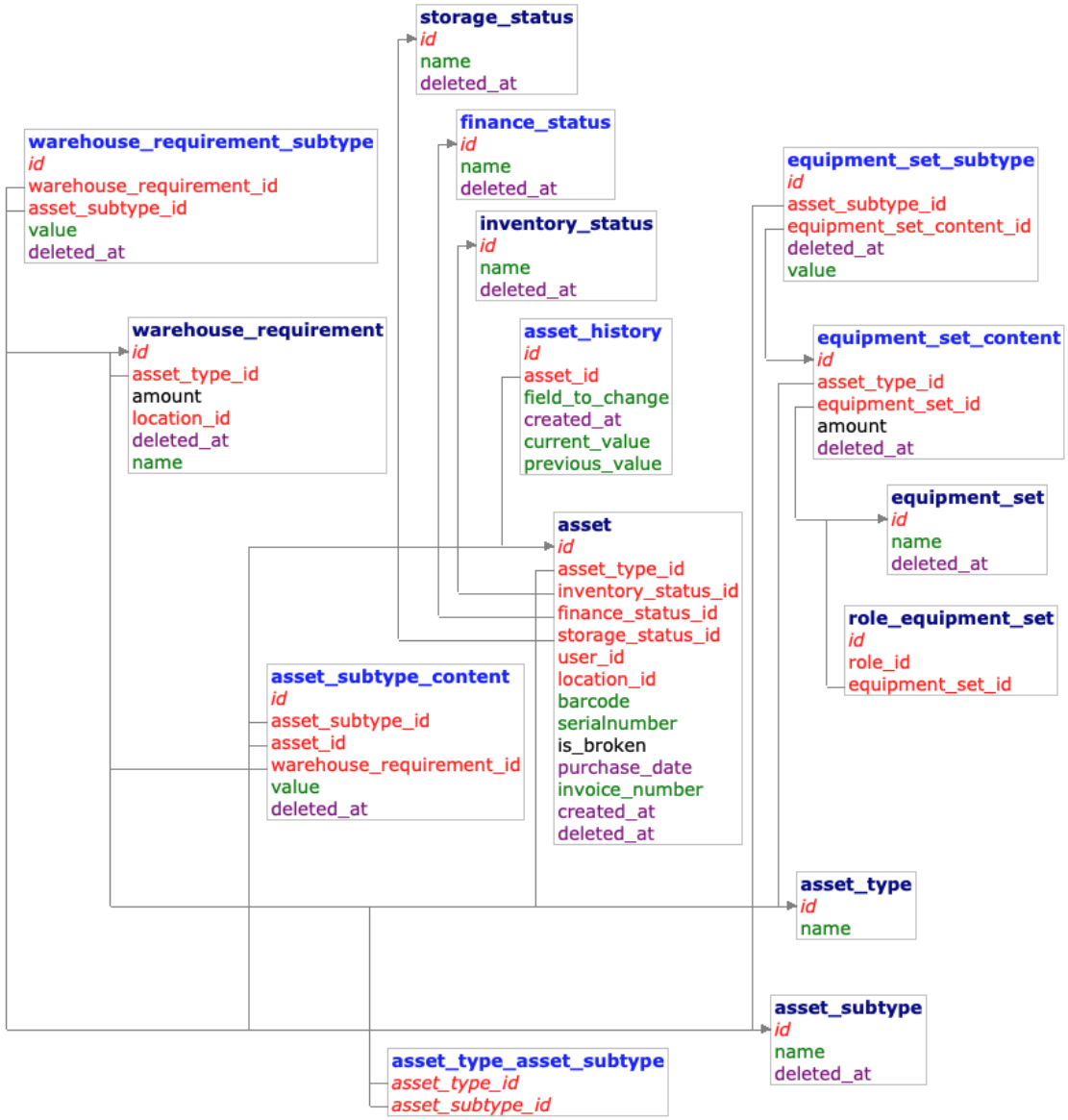
Arhitektuur on ülesehitatud võimalikult lihtsalt. *Back-end* on ühenduses nii SQL andmebaasiga kui ka *front-end* projektiga. *Front-end* projektide ja andmebaasi vahel otsene ühendus puudub. Arendusprojektis kasutatakse REST arhitektuuri stiili CRUD funktsioonidega, mis tähistab loomise, lugemise, uuendamise ja kustutamise päringuid. Andmete kuvamiseks tuleb liideses teha lugemise *(read)* toimingut, et andmebaasist andmeid saada. Uute andmete lisamiseks andmebaasi tuleb teha lisamise *(create)* toiming. Andmete uuendamiseks ja kustutamiseks tehakse vastavalt CRUD muutmise (*update)* ja kustutamise (*delete)* päringuid.

4.2.2 Andmebaas

Andmebaasiks valiti MySQL andmebaas, mille haldamiseks kasutati Admineri. MySQL on avatud lähtekoodiga SQL-i relatsioonilise andmebaasi haldussüsteem [14]. Relatsiooniline andmebaas on üksuste kogum, mille vahel on eelnevalt loodud seosed. Kõik objektid on jagatud eraldi tabelitesse, kus veerud tähistavad üksuste atribuute. Iga tabeli rida tähistab ühte andmeüksust. [16]

Antud projekti jaoks valiti relatsiooniline andmebaas. Selline valik tehti seetõttu, et relatsioonilises andmebaasis on andmed paremini struktureeritud ning seetõttu on lihtsam teha keerukaid andmebaasi päringuid. Samuti suurendab see andmete turvalisust, sest kasutaja saab kätte vaid andmed, mis ta küsib päringuga. Ehk kasutajale ei näidata üleliigseid andmeid. [16]

Kokku on andmebaasis 15 tabelit (Joonis 4). Andmebaasi tabelite loomiseks kasutati SQL (*Structured Query Language)* programmeerimiskeelt.



Joonis 4. Andmebaasi tabelid ning nende vahelised relatsioonid.

SQL lausetega saab luua tabeleid, mis sisaldavad vajalikke veerge vastavate andmetüüpidega. Joonisel 5 on kuvatud SQL lause, mis loob andmebaasi *Asset History* tabeli. Antud tabelisse lisatakse ka tabeli atribuudid, mis on määratud *Asset History Entity* klassis. Iga atribuudi andmetüüp on samuti määratud *Entity* klassis. Samuti luuakse relatsioon *Asset History* ja *Asset* tabelite vahel.

```
Public function up(Schema $schema): void
{
	$this->addSql('CREATE TABLE asset_history (
	id BINARY(16) NOT NULL COMMENT \'(DC2Type:uuid)\',
	asset_id BINARY(16) DEFAULT NULL,
	field_to_change VARCHAR(255) DEFAULT NULL,
	created_at DATETIME DEFAULT NULL,
	current_value VARCHAR(255) DEFAULT NULL,
	previous_value VARCHAR(255) DEFAULT NULL,
	INDEX IDX_4454311D5DA1941 (asset_id),
	PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE
	`utf8mb4_unicode_ci` ENGINE = InnoDB');

	$this->addSql('ALTER TABLE asset_history ADD CONSTRAINT
	FK_4454311D5DA1941 FOREIGN KEY (asset_id) REFERENCES asset (id)');
}
```

Joonis 5. SQL lause *Asset History* tabeli loomiseks.

Eelnevalt kirjeldatud SQL lause tulemuseks on tabel, mille struktuur andmebaasis on näidatud Joonisel 6.

Column	Type	Comment
id	binary(16)	(DC2Type:uuid)
created_at	datetime *NULL*	
previous_value	varchar(255) *NULL*	
field_to_change	varchar(255) *NULL*	
asset_id	binary(16) *NULL*	(DC2Type:uuid)
current_value	varchar(255) *NULL*	

Joonis 6. *Asset History* tabeli struktuur andmebaasis.

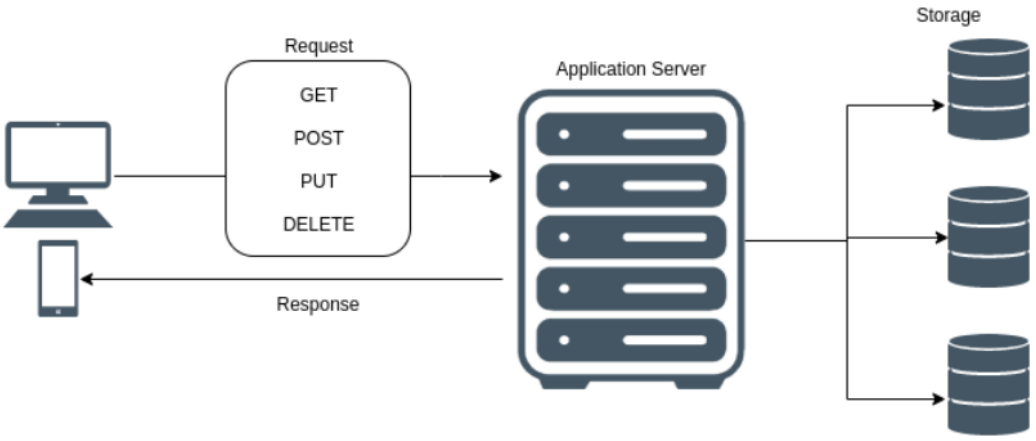
SQL programmeerimiskeelt kasutatakse ka tabelite ja andmetüüpide muutmiseks. Joonisel 7 on näha SQL lause, millega lisatakse tabelisse *Asset History* uus omadus *current_value*, mille andmetüüp on VARCHAR ning vaikimisi väärtus on NULL.

```
Public function up(Schema $schema): void
{
        $this->addSql('ALTER TABLE asset_history ADD current_value
        VARCHAR(255) DEFAULTT NULL);
}
```

Joonis 7. SQL lause *Asset History* tabelisse uue omaduse lisamiseks.

4.2.3 Back-end

Back-end projekti arendamiseks valiti Symfony raamistiku veebi API mall, mis kasutab REST arhitektuuristiili (Joonis 8). See võimaldab integreerida tarkvara süsteeme üle interneti. REST töötab koos HTTP protokolliga, mis sisaldab *get, post, put* ja *delete* meetodeid, mille kaudu on kliendil võimalik süsteemiga suhelda. See protokoll määrab ära sõnumite edastamise ja toimingute vastamise erinevatele käskudele. [17]



Joonis 8. REST arhitektuuri diagramm [18].

Back-end koosneb erinevatest kihtidest, kus igal kihil on oma kindel ülesanne. Nendeks kihtideks on: *Contoller*, *Entity* ja *Repository*. *Controller* kaustas on klassid, mis sisaldavad funktsioone, mille kaudu on kasutajal võimalik teha süsteemis päringuid. Iga funktsiooni juures on välja toodud aadress, mille kaudu saab näha veebis andmebaasist tulevaid andmeid. Joonisel 9 on välja toodud *Asset Type Controlleri* klassis olev *get* päring, mille kaudu saab süsteemi kasutaja kätte kõik andmebaasis olevad *Asset Type* objektid.

```
#[Route('/asset/types', name: 'app_asset_types', methods: "GET")]
public function get(EntityManagerInterface $entityManager): Response
    {
        try {
             $assetTypes=$entityManager-
             >getRepository(AssetType::class)->findAll();
             $returnAssetTypes = [];
             foreach ($assetTypes as $assetType)
             {
                $returnAssetTypes[] = $assetType->getJson();
             }
             return $this->json([
                'items' => $returnAssetTypes,
             ]);
         }
         catch (Exception $exception) {
             return $this->json([
                "error" => $exception->getMessage(),
             ]);
         }
    }
```

Joonis 9. *Asset Type* klassi *get* päring.

Joonisel 10 on näha objekti *Asset History Entity* klassi. Klass sisaldab objekti omadusi ning neid kirjeldavaid andmetüüpe, mis salvestatakse andmebaasi. Igal objektil on olemas unikaalne ID, mille kaudu see andmebaasis tuvastatakse. Antud süsteemis kasutatakse ID-na Uuid. UUID (*Universally Unique Identifier*) on 36-kohaline tähtnumbriline string, mida saab kasutada teabe tuvastamiseks [19].

```
#[ORM\Id]
#[ORM\Column(type: 'uuid', unique: true)]
#[ORM\GeneratedValue(strategy: 'CUSTOM')]
#[ORM\CustomIdGenerator(class: 'doctrine.uuid_generator')]
private ?Uuid $id = null;

#[ORM\ManyToOne(inversedBy: 'assetHistory')]
private ?Asset $asset = null;

#[ORM\Column(length: 255, nullable: true)]
private ?string $fieldToChange = null;

#[ORM\Column(type: Types::DATETIME_MUTABLE, nullable: true)]
private ?DateTimeInterface $createdAt = null;

#[ORM\Column(length: 255, nullable: true)]
private ?string $currentValue = null;

#[ORM\Column(length: 255, nullable: true)]
private ?string $previousValue = null;
```

Joonis 10. *Asset History Entity* klass.

Repository klassid sisaldavad andmebaasi tehtavaid päringuid. Nendeks päringuteks on näiteks lisamine, kustutamine, duplikaatide leidmine. Joonisel 11 on kujutatud *Asset Repository* klassi andmebaasi päring, et leida andmebaasist üles kõik objektid, millel on samasugune seerianumber. Lisas 12 on näha *Asset Repository* klassi funktsioone. Üks neist tegeleb andmete salvestamisega andmebaasi ning teine kustutamisega andmebaasist.

```
public function findAssetDuplicates()
    {
        $qb = $this->createQueryBuilder("asset")
            ->select('asset.userId')
            ->addSelect('assetType.id AS assetTypeId')
            ->addSelect('COUNT(asset.id) AS count')
            ->leftJoin('asset.assetType', 'assetType')
            ->groupBy('asset.userId', 'asset.assetType')
            ->having('count > 1');
        return $qb
            ->getQuery()
            ->getResult();
    }
```

Joonis 11. *Asset Repository* andmebaasi päring identsete seerianumbritega objektide leidmiseks.

4.2.4 Front-end

Front-end arendamiseks kasutati ReactJS raamistikku. Iga lehekülje haldamiseks tehti eraldi JavaScripti fail. Kõik failid, mis sisaldavad lehekülgede vaateid, koondati *Pages* kausta. Lisamise ja kustutamise jaoks tehti eraldi failid, mis koondati *Modals* kausta. Samuti on projektis olemas vaated erandjuhtude – näiteks lehekülge, mida otsitakse, ei eksisteeri – jaoks.

Vaade, mis saab kätte kõik objektid, koosneb erinevatest osadest. Joonisel 12 on näha, kuidas saadakse kätte andmed. Tehakse API päring *back-endi*, mis omakorda teeb päringu andmebaasi.

```
const requestToken = Axios.CancelToken.source();
async function fetchResults() {
    try {
       const response = await Axios.get("http://localhost:9090/assets", {}, {
       cancelToken: requestToken.token });
       dispatch({ type: "setAssets", value: response.data.items });
    } catch (e) {
       console.log("There was a problem or the request was cancelled. " +
       e.message);
    }
  }
```

Joonis 12. Andmete pärimine *front-endis*.

Seejärel tehakse valitud ReactJS *template* stiili järgi tabel ning kuvatakse veergude pealkirjas. Nagu on näha Joonisel 13.

```
<Table striped bordered hover>
      <thead>
        <tr>
          <th>#</th>
          <th>ID</th>
          <th>User</th>
          <th>Asset Type</th>
          <th>Serialnumber</th>
          <th>Barcode</th>
          <th>Inventory Status</th>
          <th>Storage Status</th>
          <th>Finance Status</th>
          <th>Location</th>
          <th>Purchase Date</th>
          <th>Invoice Number</th>
          <th>Is Broken</th>
          <th>Created at</th>
          <th>Deleted at</th>
        </tr>
       </thead>
</Table>
```

Joonis 13. *Asset* vaate tabeli pealkirjad.

Seejärel kuvatakse tabelisse andmed, mis saadi päringuga andmebaasist. Joonisel 14 on näha, kuidas toimub *Asset* objektide andmete kuvamine rakenduses kasutajale. Lisas 3 on näha antud tabel süsteemi kasutaja vaates.

```
{state.assets &&
  state.assets.map((asset, index) => (
    <tr key={asset.id}>
      <td>{index + 1}</td>
      <td>
        <Link to={"/asset/assets/" + asset.id}>{asset.id}</Link>
      </td>
      <td>{asset.userName}</td>
      <td>{asset.assetType?.name}</td>
      <td>{asset.serialnumber}</td>
      <td>{asset.barcode}</td>
      <td>{asset.inventoryStatus?.name}</td>
      <td>{asset.storageStatus?.name}</td>
      <td>{asset.financeStatus?.name}</td>
      <td>{asset.locationName}</td>
      <td>
        <Moment date={asset.purchaseDate} />
      </td>
      <td>{asset.invoiceNumber}</td>
      <td>{asset.is_Broken ? "Yes" : "No"}</td>
      <td>
        <Moment date={asset.createdAt} />
      </td>
      <td>{asset.deletedAt}</td>
    </tr>
  ))}
```

Joonis 14. Andmete kuvamine *Asset* vaate tabelis.

4.2.5 Sisselogimine

Kasutaja sisselogimine toimub *front-endi* kaudu. Kasutajale kuvatakse vorm, kuhu tuleb sisestada andmed (Lisa 13). Kasutaja autentimine toimub *Auth Service* mikroteenuse kaudu. Joonisel 15 on kuvatud funktsioon, mis kutsub välja *Auth* teenuse. Antud teenuse ülesanne on kontrollida, kas sisestatud andmed on korrektsed. Korrektsuse korral antakse kasutajale juurdepääs süsteemile.

```
async function handleSubmit(e) {
e.preventDefault();
try {
      const response = await Axios.post("auth/auth", { username,
      password });
      if (response.data.error) {
        console.log("Error!");
      } else {
        appDispatch({ type: "login", data: response.data });
        navigate("/");
      }
    } catch (e) {
      console.log("There was a problem!");
      console.log(e);
    }
 }
```

Joonis 15. Auth Service välja kutsumine.

4.2.6 Kasutajate, rollide ja asukohtade pärimine

Nii kasutajate, rollide kui ka asukohtade andmed päritakse SPOAP teenusest. Andmete pärimine toimub *back-endi* kaudu, kus ühendatakse andmed nii *Asset* tabelist kui ka teisest teenusest päritud andmed. Joonisel 16 on näha päring, mis tehakse SPOAP teenuse pihta, mille kaudu saadakse kasutajate andmed.

```
$responseUser = $client->request(
   "POST",
   "http://spoap-web/people/byids",
   [
       'json' => [
           "ids" => $returnUsers,
       ]
   ]
)->toArray();
```

Joonis 16. Kasutajate pärimine SPOAP teenusest.

5 Analüüs

5.1 Nõuete täitmine

Süsteem valmis vastavalt töö alguses esitatud nõuetele ja eesmärgid täideti. Lisaks objektide lisamine, kustutamise, muutmise ning nimekirja vaate funktsioonidele arendati valmis ka kolm suuremat funktsionaalsust. Süsteemiga on võimalik saada iga objekti kohta infot nõutud detailsusega. Samuti on võimalik näha kõiki muudatusi, mis on tehtud objektiga, andes ülevaate nii staatuste, asukohtade kui ka kasutajate kohta. Lisaks teavitab süsteem admini kui laos on vähem kindla objektitüübiga varasid, kui on ettenähtud. Süsteem täidab ka kontrollülesandeid täitvaid funktsionaalsusi. Nimelt saab kontrollida, kas süsteemis on duplikaatsete seerianumbritega objekte ning kas leidub kasutajaid, kelle kasutuses on mitu sama tüüpi objekti.

5.2 Tööle kulunud aeg

Arendusprojektiks kulunud aega logiti Scoro *Calendar* moodulisse. Logide alla koguti kogu arendusele kulunud aeg, mis toimus vahemikus 23. jaanuar 2023 kuni 6. mai 2023. Sinna hulka logiti ka koosolekud ning koodi ülevaatamised, mis tehti koos tööpoolse juhendajaga. Kokku kulus arendusele 240 tundi (Lisa 14).

5.3 Hinnang teostusele

Töö protsess oli hästi planeeritud ning sujus ilma suuremate probleemideta. Antud projekti tegemiseks kulus 15 nädalat. Sinna hulka on arvatud nii nõuete koostamine ning koodi kirjutamine. Autor sai asjakohast tuge ja nõu mõlemalt juhendajalt. Nii tööpoolse kui ka koolipoolse juhendajaga sujus autori arvates koostöö hästi ning suhtlus mõlemaga oli pidev.

Arenduse käigus esines ka probleeme, mis tulenesid sellest, et nii programmeerimiskeel PHP kui ka Symfony ja ReactJS raamistikud olid autori jaoks uued. Seetõttu oli algne arendamiseprotsess aeglasem kui algselt autor arvas. Probleemide esinemisel pöördus autor tööpoolse juhendaja poole, kellega koos leiti neile lahendused. Saab väita, et

kindlasti sai autor arendamisekäigus juurde palju uusi teadmisi nii raamistike kui ka erinevate viiside kohta, kuidas antud raamistiku puhul koodis vigu leida.

Kõige keerukamaks osutus autori jaoks *front-end* arendus. ReactJS raamistik erines üsna palju varasemalt kasutatud raamistikest ning selle õppimiseks ja arusaamiseks kulus oodatust rohkem aega. Siinkohal oli palju abi tööpoolsest juhendajast, kellel oli varasem kogemus antud raamistikuga, kes oli abiks koodist arusaamisel ning komponentide seletamisel. Samuti esines olukordi, kus tuli parandada vigu *back-end* koodis, sest *front-endiga* ühendades tekkis probleeme andmete näitamisega.

5.4 Edasiarendus

Antud projekti koostamisel paika pandud nõuded said täidetud ning tulemusega olid rahul nii töö autor kui ka tööpoolne juhendaja. Küll aga läheb projekti arendus edasi, sest tekkis mitmeid ideid, kuidas süsteemi edasi arendada.

Plaanis on arendada juurde funktsionaalsus, mis parendab eelarve planeerimist. Antud ettevõttes vahetatakse arvuteid eluea lõppedes või juhul, kui seadme võimsus ei ole enam piisav vastavas töörollis. Varasem kogemus antud ettevõttes on näidanud, et arvuti eluiga on ligikaudu kolm aastat. Iga objekti juures on olemas ka ostukuupäev, mille kaudu saab ligikaudu arvutada, mis ajal tuleb seade välja vahetada. Antud funktsiooni abil on võimalik ligikaudselt arvutada, mitu seadet tuleb kindlal aastal välja vahetada, mille kaudu saab välja arvutada ligikaudse kulu uutele seadmetele.

Lisaks on plaan ühendada antud süsteem ettevõttes kasutatava arvutite keskhaldussüsteemiga. Keskhaldussüsteemis on näha arvuti seerianumber ning kasutaja. Süsteemide ühendamise eesmärk on näha, kas keskhaldussüsteemi all olevad arvutid ja nende kasutajad on kooskõlas arendatud süsteemis määratud kasutajate ning seadmetega. Seeläbi on võimalik näha, kas mõnel kasutajal on käes mitu arvutit, kuigi reaalsuses kasutab vaid ühte seadet.

Ettevõttes viiakse üks kord aastas läbi riistvara audit. Selle käigus kontrollitakse, kas töötajate kasutuses on seadmed, mis on neile määratud. Selle abil saab tuvastada seadmed, mis on kadunud, lähevad müüki või mahakandmisele. Siiamaani on kontroll toimunud manuaalselt, kus kasutajad saadavad IT meeskonna liikmetele pildid seerianumbritest, mida IT tiim hakkab võrdlema süsteemis oleva infoga. Seetõttu on plaan *Asset* teenusele

arendada *Asset Check* funktsioon, mis automatiseeriks praegu kasutusel olevat manuaalset tööd.

Samuti sooviks autor parendada ka süsteemi disaini. Kuna arenduse käigus keskenduti pigem funktsionaalsuste arendamisele, siis jäi disain pigem tagaplaanile.

6 Kokkuvõte

Käesoleva töö raames arendati ettevõtte Scoro Software OÜ *Internal* IT tiimile varade haldussüsteem kui mikroteenus, *Asset Service*, osana sisehaldusportaalist SInMan. Töös on analüüsitud kahe olemasoleva tarkvara lahendusi, mille põhjal on väljatoodud arendatud süsteemi vajalikkus.

Töö eesmärk oli arendada rakendus, mille kaudu on võimalik saada detailne ülevaade kõikidest ettevõttes kasutatavatest riistvaralistest objektidest. Lisaks on võimalik saada ülevaade kõikidest muudatusest, mis tehti *asset* objektiga. Ajaloos on näha nii staatuste muutusi, kõiki seadme kasutajaid kui ka asukohti, kus seade kasutuses on olnud. Samuti on näha, kas seade on olnud mingil hetkel katki. Lisaks saab efektiivsemalt hallata laos olevaid seadmeid. Nimelt võrreldakse kasutaja poolt valitud objektitüübi tegelikku laoseisu vastava objektitüübi nõudega. Juhul kui tegelikult on laos olev kogus väiksem kui nõudes määratud, siis saadetakse süsteemi adminile vastav teade. Samuti saab süsteemis läbi viia kahte erinevat kontrolli. Nimelt saab kontrollida, kas leidub objekte, millel on identsed seerianumbrid, ning seda, kas leidub kasutajaid, kelle kasutuses on rohkem kui üks sama objektitüübiga seadet.

Projekt jaotati kaheks osaks: *back-end* ehk API ja *front-end*. *Back-endi* arendus toimus PhpStormis kasutades PHP programmeerimiskeelt ning Symfony raamistikku. *Front-endi* arendati Visual Studio Code'is kasutades JavaScripti ning ReactJS raamistikku.

Arendusprojekti lõpuks sai valmis veebirakendus, millel kuvatakse andmebaasist tulevat informatsiooni. Projekt ühendati kahe mikroteenusega. Nendeks on SPOAP ja *Auth service*, mis on samuti ettevõttesiseselt arendatud süsteemid. SPOAP teenuse kaudu päritakse andmed kasutajate, asukohtade ning rollide kohta. Juurdepääs süsteemile on piiratud ning seda kontrollib *Auth service* teenus.

Kasutatud kirjandus

[1] A. Hayes, „Inventory Management Defined, Plus Methods and Techniques,“ *Investopedia,* 28. Märts 2023. [Online]. Loetud aadressil: https://www.investopedia.com/terms/i/inventory-management.asp Kasutatud: 16. Aprill 2023.

[2] Oracle Netsuite, *NetSuite Inventory Management Systems Software,* [Online]. Loetud addressil: https://www.netsuite.com/portal/products/erp/warehouse-fulfillment/inventory-management.shtml Kasutatud: 16. Aprill 2023.

[3] Astro Baltics, *Majandustarkvara- teejuht tulevikku,* [Online]. Loetud addressil: https://astrobaltics.eu/noom/?gclid=Cj0KCQiAjbagBhD3ARIsANRrqEs1pSYgudRBP6nw_LbRaCL3hnZthqEhK2SOiBk9gNr3uNXb4c-oJbwaAhcQEALw_wcB Kasutatud: 16. Aprill 2023.

[4] C. Richardson, „Microservice Architecture,“, *Microservices Architecture,* [Online]. Loetud addressil: https://microservices.io Kasutatud: 16. Aprill 2023.

[5] P. Jura, *Symfony 6 Framework Hands-On 2023*, Udemy, Inc., 2022.

[6] T. Buchalka, *PHP for Beginners,* Udemy, Inc., 2022.

[7] M. Fowler, „Waterfall Process,“ 13. November 2019. [Online]. Loetud addressil: https://martinfowler.com/bliki/WaterfallProcess.html Kasutatud: 8. Mai 2023.

[8] M. Fowler, „Domain Model,“ [Online]. Loetud addressil: https://martinfowler.com/eaaCatalog/domainModel.html Kasutatud: 8. Mai 2023.

[9] M. Fowler,“DDD_Aggregate“, 23. Aprill 2013. [Online]. Loetud addressil: https://martinfowler.com/bliki/DDD_Aggregate.html Kasutatud: 9. Mai 2023.

[10] R. Hughes, „Context Diagram,“ *Science Direct,* 2016. [Online]. Loetud addressil: https://www.sciencedirect.com/topics/computer-science/context-diagram Kasutatud: 16. Aprill 2023.

[11] S. Adolph, P. Bramble, A. Cockburn and A. Pols, *Patterns for Effective Use Cases*, Addison-Wesley Professional, 2002. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/patterns-for-effective/0201721848/ Kasutatud: 16. Aprill 2023.

[12] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd*, Prentice Hall, 2001. [E-book]. Loetud aadressil: https://learning.oreilly.com/library/view/applying-uml-and/0130925691/ Kasutatud: 16. Aprill 2023.

[13] V. Vorteil ja J. Laanpere, "Tarkvara arendusnõuded" in *Tarkvara analüüs ja testimine,* [Online]. Loetud addressil: https://web.htk.tlu.ee/digitaru/testimine/chapter/tarkvara-arendusnouded/ Kasutatud: 16. Aprill 2023.

[14] M. Ozkaya, „Headless Architecture with Separated UI for Backend and Frontend," *Medium,* 17. Veebruar 2023. [Online]. Loetud addressil: https://medium.com/design-microservices-architecture-with-patterns/headless-architecture-with-separated-ui-for-backend-and-frontend-f9789920e112 Kasutatud: 16. Aprill 2023.

[15] S. Ravoof, „What Is MySQL? A Beginner-Friendly Explanation," *Kinsta,* 11. Aprill 2022. [Online]. Loetud addressil: https://kinsta.com/knowledgebase/what-is-mysql/ Kasutatud: 23 Aprill 2023.

[16] A. Lo Duca, „Relational VS Non Relational Databases," *Medium,* 23. September 2021. [Online]. Loetud addressil: https://towardsdatascience.com/relational-vs-non-relational-databases-f2ac792482e3 Kasutatud: 23 Aprill 2023.

[17] L. Xiao-Hong, *Research and Development of Web of Things System Based on Rest Architecture,* 2014. [E-book]. Loetud addressil: https://ieeexplore.ieee.org/document/6977704 Kasutatud: 16. Aprill 2023.

[18] M. Moody, „GraphQL vs. REST API Architecture - Which Should You Use for Your API?," *Codesmith,* 27. November 2019. [Online]. Loetud addressil: https://www.codesmith.io/blog/graphql-vs-rest Kasutatud: 16. Aprill 2023.

[19] C. Custer, „What is a UUID, and what is it used for?," *Cockroach Labs,* 16. Märts 2023. [Online]. Loetud addressil: https://www.cockroachlabs.com/blog/what-is-a-uuid/#what-is-a-uuid Kasutatud: 16. Aprill 2023.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Annigrete Suimets

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ettevõttesisese riistvara haldussüsteemi kui mikroteenuse arendamine“, mille juhendajad on Jelena Vendelin ja Karl Hendrik Leppmets.
   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.

16.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Näide projekti ning seda sisaldavate ülesannete haldamisest

Task	Status	Latest deadline
☑ Fixes to other functions and test with Postman	DONE	09.03.2023
☑ getJson inside getJson function to get related entities	DONE	09.03.2023
☑ Write CRUD functions to all entities	DONE	09.03.2023
☑ Deleted boolean value to all entities, but don't show it in *View all*	DONE	09.03.2023
☑ Filters for list view	DONE	
☑ Basic "smartness" - find duplicate serial numbers, people with multiple same type assets, etc.	DONE	
☑ Asset history	DONE	
☐ Permissions - who can view, edit, delete what etc.	PENDING	
☐ Lil bit of front-end	PENDING	
☐ Connecting with SPOAP for users, locations and roles.	PENDING	

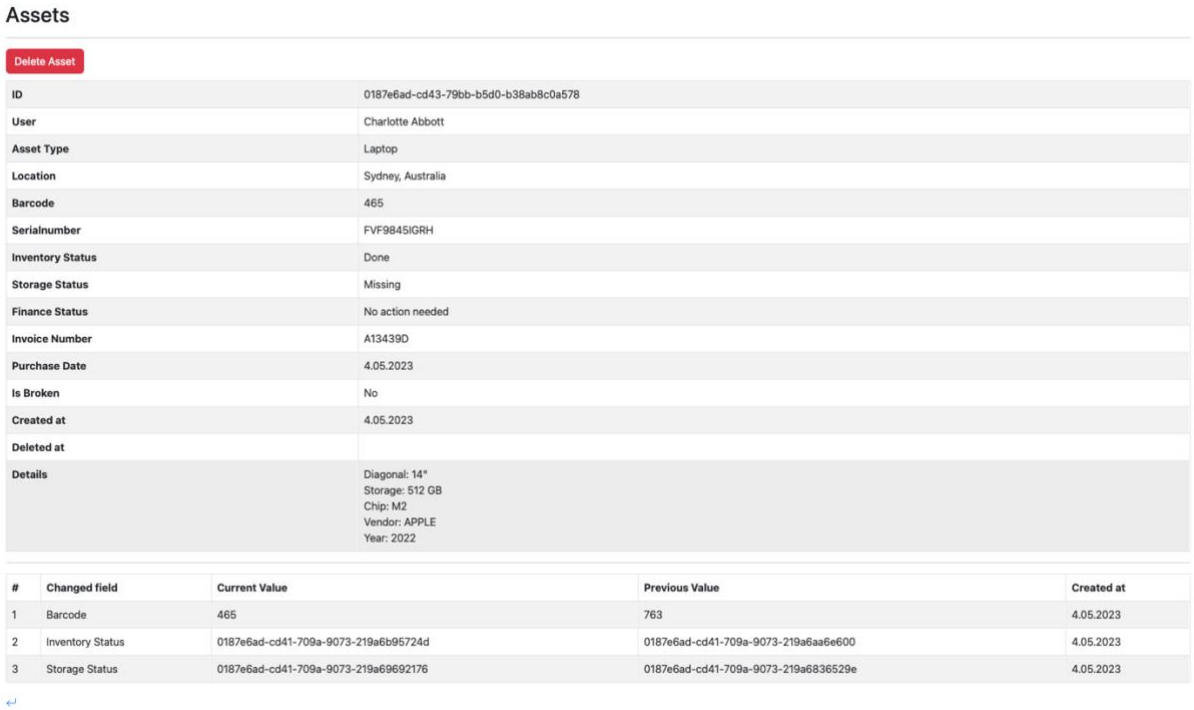
Lisa 3 – *Asset* tabeli vaade kasutaja vaatepunktist

Assets

Add Asset

#	ID	User	Asset Type	Serialnumber	Barcode	Inventory Status	Storage Status	Finance Status	Location	Purchase Date	Invoice Number	Is Broken	Created at	Deleted at
1	0187e6ad-cd43-79bb-b5d0-b38ab8c0a578	Duke	Laptop	FVF9845IGRH	465	Done	Missing	No action needed	London, UK	4.05.2023	A13439D	No	4.05.2023	
2	0187e6ad-cd43-79bb-b5d0-b38ab8fbfb40		Monitor	LE2FC3KEV7RI	123	Done	In repair	No action needed	Sydney, Australia	4.05.2023	2837284	Yes	4.05.2023	
3	0187e6ad-cd43-79bb-b5d0-b38ab9a859a6	Charlotte Abbott	Laptop	KJEG34BEVK	827	Done	Not yet received	No action needed	London, UK	4.05.2023	14784242	No	4.05.2023	
4	0187e6ad-cd43-79bb-b5d0-b38ab9af9c1e	Charlotte Abbott	Mouse	KDEMNF-WEF4W	827	Done	With user	No action needed	London, UK	4.05.2023	49348	No	4.05.2023	
5	0187e6ad-cd43-79bb-b5d0-b38aba2d2a74	Charlotte Abbott	Mouse	KDEMNF-WEF4W	2222223	Done	In repair	No action needed		4.05.2023	124234	No	4.05.2023	
6	0187e6ad-cd43-79bb-b5d0-b38aba8db134	Shannon Anderson	Laptop	LE2FC3KEV7RI	333		With user	Written off		4.05.2023	34551	No	4.05.2023	

Lisa 4 – Üksiku *Asseti* detailne vaade



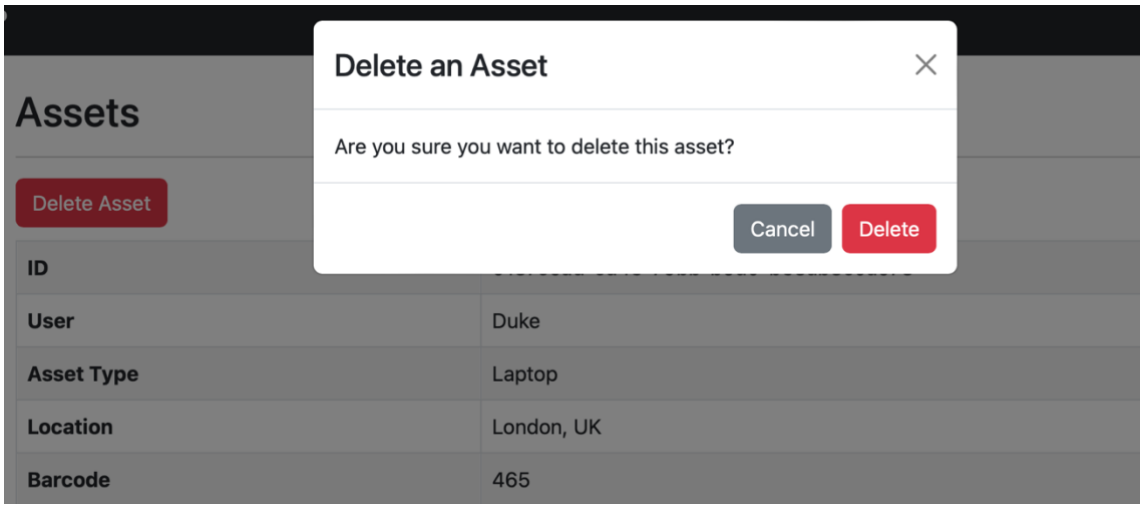
Assets

Delete Asset

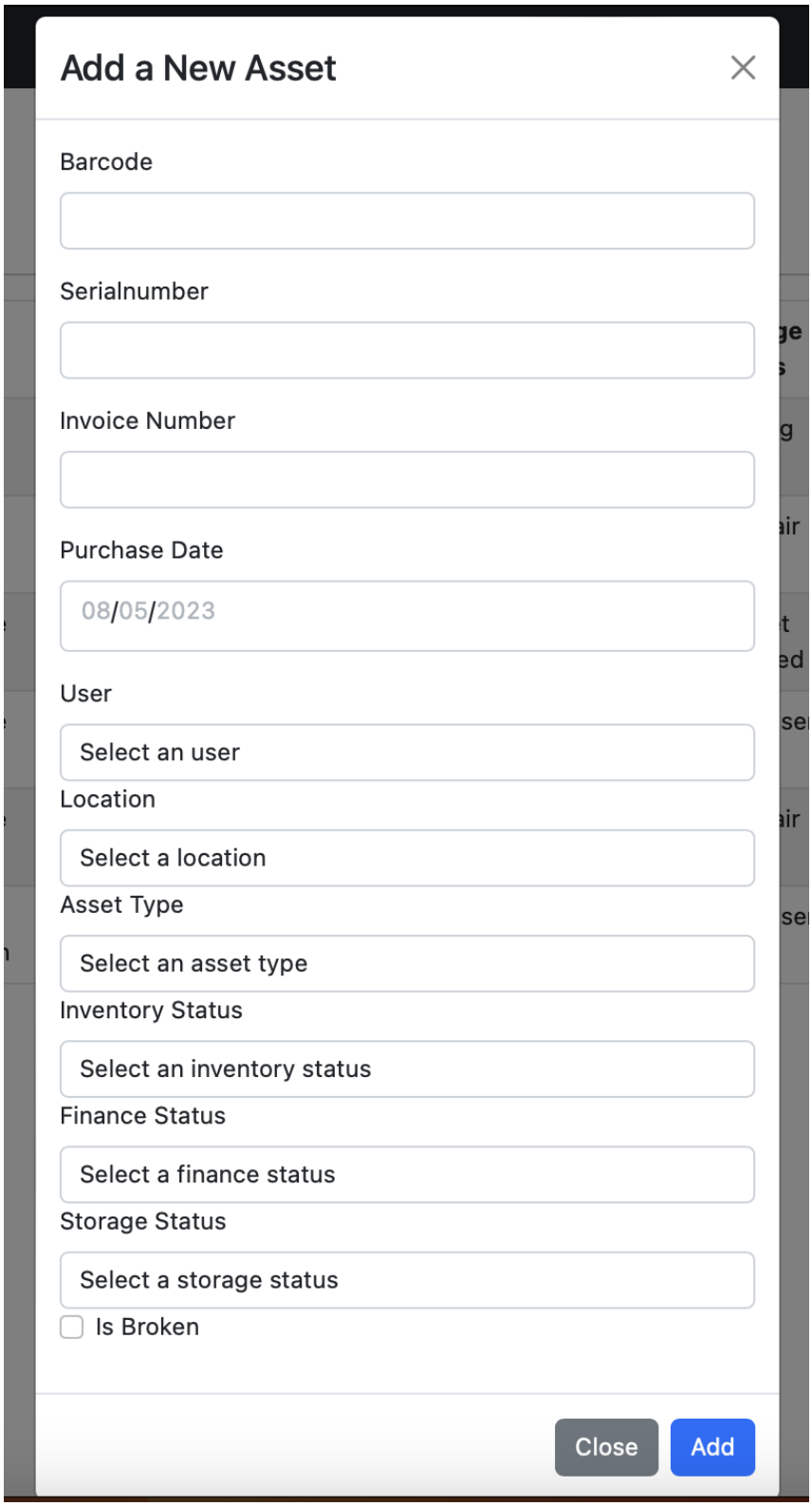
ID	0187e6ad-cd43-79bb-b5d0-b38ab8c0a578
User	Charlotte Abbott
Asset Type	Laptop
Location	Sydney, Australia
Barcode	465
Serialnumber	FVF9845IGRH
Inventory Status	Done
Storage Status	Missing
Finance Status	No action needed
Invoice Number	A13439D
Purchase Date	4.05.2023
Is Broken	No
Created at	4.05.2023
Deleted at	
Details	Diagonal: 14" Storage: 512 GB Chip: M2 Vendor: APPLE Year: 2022

#	Changed field	Current Value	Previous Value	Created at
1	Barcode	465	763	4.05.2023
2	Inventory Status	0187e6ad-cd41-709a-9073-219a6b95724d	0187e6ad-cd41-709a-9073-219a6aa6e600	4.05.2023
3	Storage Status	0187e6ad-cd41-709a-9073-219a69692176	0187e6ad-cd41-709a-9073-219a6836529e	4.05.2023

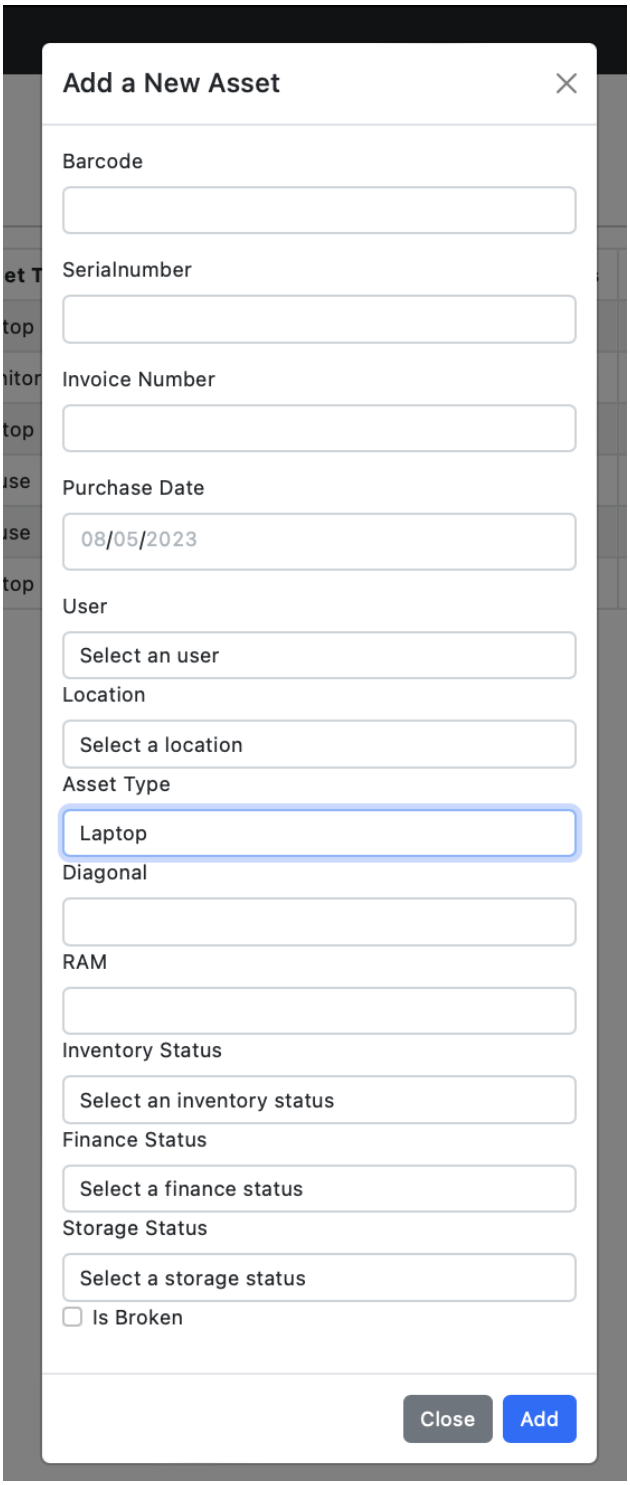
Lisa 5 – *Asset* objekti kustutamise vorm



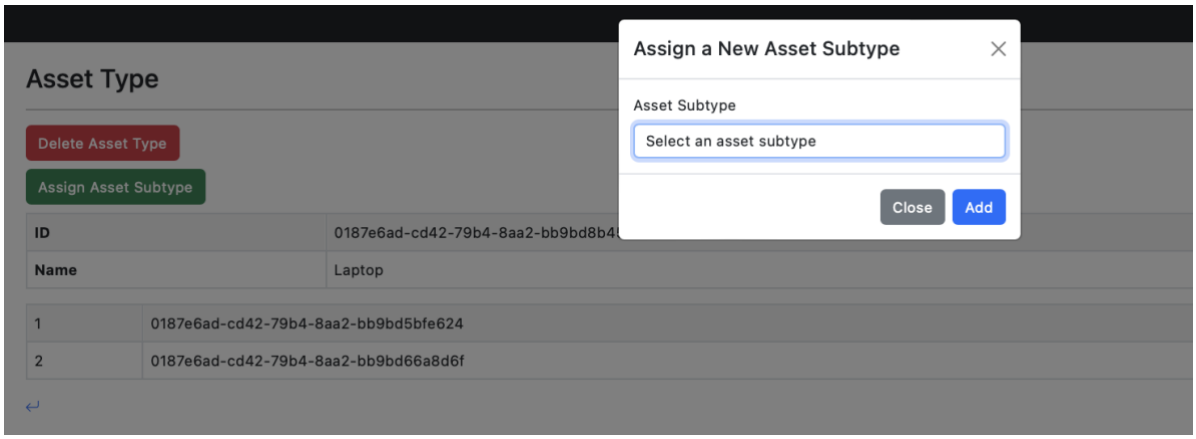
Lisa 6 – *Asset* objekti lisamise vorm



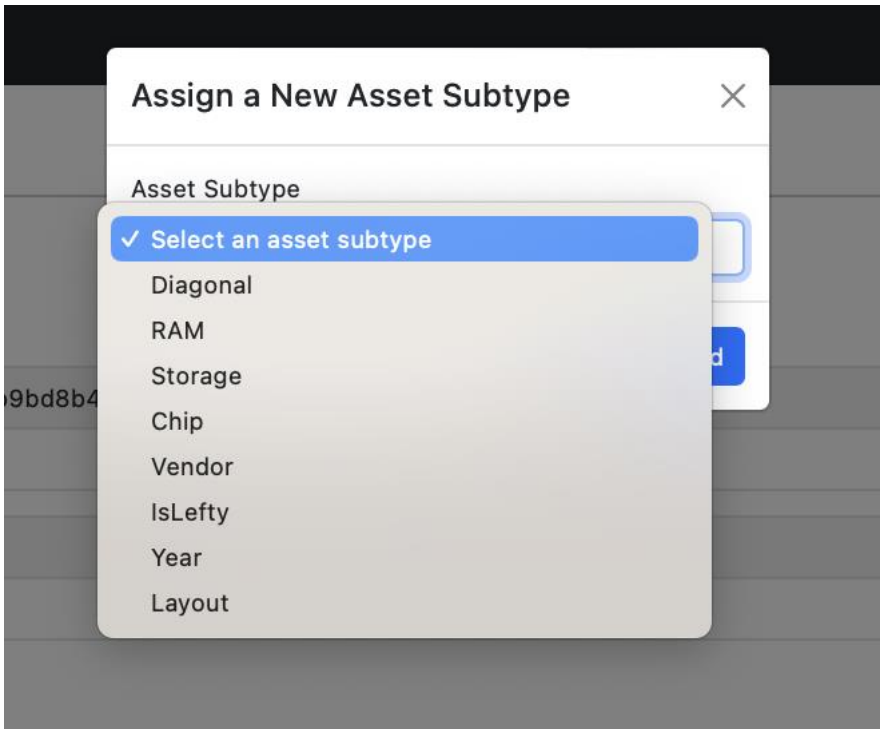
Lisa 7 – *Asset* objekti lisamise detailne vorm



Lisa 8 – Objekti tüübile detailsete väljade määramine



Lisa 9 – Objekti tüübile detailsete väljade valimine



Lisa 10 – Õiguste piiramine *front-endis Asset* klassi näitel

```
<h2>Assets</h2>
      {appState.user.roles.includes("ROLE_AUTH_ASSETS_ADD") && <AddRoleModal
      parentDispatch={dispatch} updateData={fetchResults} />}
<hr />
```

Lisa 11 – Õiguste piiramine *back-endis Asset History* klassi näitel

```
try {
     $this->denyAccessUnlessGranted("ROLE_AUTH_ASSET_HISTORIES_VIEW");
     return $this->json($assetHistoryService->getAll());
 }
 catch (AccessDeniedException) {
     return $this->json ([
         'error' => "Access denied.",
     ], 403);
 }
```

Lisa 12 – Andmebaasi lisamise ja kustutamise meetodid *Asset Repository* klassi näitel

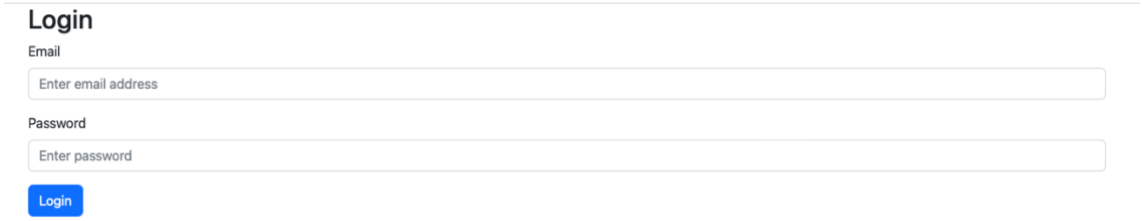
```
public function save(AssetHistory $entity, bool $flush = false): void
    {
        $this->getEntityManager()->persist($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }

public function remove(AssetHistory $entity, bool $flush = false): void
    {
        $this->getEntityManager()->remove($entity);

        if ($flush) {
            $this->getEntityManager()->flush();
        }
    }
```

Lisa 13 – Sisselogimise vaade



Lisa 14 – Arendusele kulunud aeg vahemiku 01.05-04.05 näitel ning kogu arendusele kulunud aeg

Date	User		Title and description	Activity type	Contact	Duration
04.05.2023	AS		users, roles, locations to singlepages	Initiatives/Improvements		02:30 h
04.05.2023	AS		Initiatives/Improvements	Initiatives/Improvements		02:30 h
03.05.2023	AS		Add function with users, loc, roles	Initiatives/Improvements		01:45 h
03.05.2023	AS		Connecting with SPOAP	Initiatives/Improvements		00:00 h
02.05.2023	AS		Connecting with SPOAP	Initiatives/Improvements		02:03 h
02.05.2023			Project review	Initiatives/Improvements		01:00 h 2 x 00:30 h
02.05.2023	AS		Connecting with SPOAP	Initiatives/Improvements		00:35 h
02.05.2023	AS		Connecting with spoap	Initiatives/Improvements		00:30 h
02.05.2023	AS		Connecting with SPOAP	Initiatives/Improvements		02:01 h
01.05.2023	AS		Connecting with SPOAP	Initiatives/Improvements		00:13 h
					Total (150):	240:12 h