Jere Kuokkanen

DYNAAMISEN TIETOKANNAN KEHITTÄMINEN TESTILAITTEILLE

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutus
Huhtikuu 2023

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2023	**Tekijä/tekijät** Jere Kuokkanen
Koulutus Tieto- ja viestintätekniikan koulutus		☒ AMK ☐ YAMK
Työn nimi DYNAAMISEN TIETOKANNAN KEHITTÄMINEN TESTILAITTEILLE.		
Työn ohjaaja Jari Isohanni		**Sivumäärä** 36
Työelämäohjaaja Jani Turpeinen		

Tämä opinnäytetyö tehtiin yhteistyössä Sasken Finland Oy:n kanssa. Opinnäytetyössä kehitettiin yrityksen luoman edellisen tietokantaratkaisun pohjalta uusi, dynaaminen tietokantakokonaisuus.

Työn tavoitteena oli rakentaa edellisen tietokantapalvelimen pohjalta dynaaminen ratkaisu, jota yrityksen tulevaisuuden testilaitteet pystyisivät hyödyntämään.

Opinnäytetyön tekemisen tukena käytettiin Sasken Finland Oy:n toimittamaa dokumentaatiota, joka sisälsi tietoja tietokannan ja tietokantapalvelimen rakenteesta ja toiminnasta. Lisäksi lähteinä käytettiin käytettyihin järjestelmiin liittyvää kirjallisuutta sekä internetsivuja.

Kehitystyö tapahtui työtietokoneella virtualisoidussa ympäristössä. Kehitystyöhön käytettiin lähdekoodin muokkausta mahdollistavaa ohjelmistoa sekä yrityksen tuottamaa valmista tietokantapalvelinohjelmistoa.

Lopputuloksena syntyi tietokantakokonaisuus, jota yrityksen tulevaisuuden testilaitteet voivat hyödyntää. Ratkaisu on myös hyvin jatkokehitettävissä. Tästä opinnäytetyöstä oli hyötyä sekä minulle että yritykselle.

Asiasanat
Dynaamisuus, jatkokehitys, palvelin, tietokanta

ABSTRACT

Centria University of Applied Sciences	Date April 2023	Author Jere Kuokkanen
Degree programme Information and Communications Technology (in Finnish)		
Name of thesis DYNAMIC DATABASE DEVELOPMENT FOR TESTER DEVICES.		
Centria supervisor Jari Isohanni		**Pages** 36
Instructor representing commissioning institution or company Jani Turpeinen		

This thesis was made in collaboration with Sasken Finland Oy. In this thesis, a new and dynamic database solution was made by developing Sasken's old database solution further.

The goal of this thesis was to build a dynamic solution, which could be used by company's future tester devices.

As a support for this thesis, Sasken's documentation was used. The documentation included information about the structure and functionality of the database system. Literature related to used systems and web pages were also used as additional sources of information.

The development happened in virtualized environment. Source code editing software and Sasken's old database solution were used for the development.

As a result, a dynamic database solution, which can be used with the company's upcoming tester devices, was developed. The solution can also be developed further easily. This thesis was beneficial for both me and the company.

Key words
Database, dynamic, further development, server

KÄSITTEIDEN MÄÄRITTELY

ALUSTA

Ks. käyttöjärjestelmä.

ARVO

Tietokannassa johonkin avaimeen tallennettu tieto.

AVAIN

Tietokannan yksittäinen sarake taulussa.

BACK-END

Ohjelmiston tai koodin osa, joka mahdollistaa toiminnan ja jota käyttäjä ei voi nähdä. Usein tämä osa pitää sisällään myös järjestelmän tietovarannon, esim. tietokannan.

CLIENT-SIDE

Käyttäjäpuoli. Toiminta, joka tapahtuu käyttäjän tietokoneella.

DYNAAMINEN

Muuttuva tai mukautuva.

ENSISIJAINEN AVAIN

Myös Primary Key tai PK. Tietokannassa sarake, joka yksilöllistää jokaisen rivin. Tiedot jokaisessa rivissä ovat siis erilaisia tässä sarakkeessa.

FRONT-END

Ohjelmiston tai koodin osa, jonka käyttäjä näkee ja jonka kanssa hän on vuorovaikutuksessa.

GIT

Ohjelmisto, jota käytetään lähdekoodin versioiden hallintaan.

GITLAB

Ohjelmisto, joka käyttää Gitiä ja jota käytetään projektien hallitsemiseen.

ISÄNTÄKONE

Myös isäntätietokone. Tietokone, jolla virtuaalitietokoneet suoriutuvat.

JÄRJESTELMÄNVALVOJA

Henkilö, joka hallinnoi järjestelmää ja voi tehdä tarvittaessa muutoksia siihen.

KIRJASTO

Ohjelmoinnissa resurssi, joka sisältää valmiiksikirjoitettua koodia. Voidaan hyödyntää osana projektia.

KÄYTTÄJÄOHJELMA

Myös asiakasohjelma. Ohjelma, joka käyttää tietoja palvelimelta ja jota tavallinen käyttäjä voi käyttää.

KÄYTTÖJÄRJESTELMÄ

Myös alusta. Ohjelmisto, joka mahdollistaa tietokoneen laitteiston ja sovellusten yhteistyön.

KÄYTTÖLIITTYMÄ

Myös User Interface tai UI. Ohjelmiston osa, jonka käyttäjä näkee ja jolla käyttäjä tekee toimia ohjelmistossa.

LÄHDEKOODI

Ohjelmiston toiminnan mahdollistava kokoelma ohjelmointikielellä kirjoitettua tekstiä.

PALVELIN

Ohjelmisto tai laite, joka tarjoaa palvelun toiselle ohjelmistolle tai käyttäjälle.

PORTTI

Käyttöjärjestelmän hallitsema virtuaalinen verkkoyhteyden alku- tai päätepiste.

PROTOKOLLA

Kokoelma sääntöjä ja tapoja, joilla tieto välittyy verkossa.

RAJAPINTA

Myös Application Programming Interface tai API. Yhteinen kokoelma sääntöjä, jonka kautta useampi erilainen ohjelmisto voi kommunikoida keskenään.

SERVER-SIDE

Palvelinpuoli. Toiminta, joka tapahtuu palvelintietokoneella.

TAULU

Tietokannassa pienempi kokoelma tietoja, joka muodostuu riveistä ja sarakkeista.

TESTILAITE

Laite, jolla tehdään testejä erillisille testattaville laitteille.

TIETOKANTA

Järjestelmällinen kokoelma tallennettua tietoa.

VIERASAVAIN

Myös Foreign Key tai FK. Tietokannassa sarake, joka viittaa yksilölliseen tietoon toisessa taulussa esim. toisen taulun ensisijaisen avaimen avulla.

VIRTUALISOINTI

Toiminta, jossa tehdään virtuaalinen versio esim. tietokoneesta. Tällöin virtualisoitu tietokone ei ole fyysinen laite, vaan se toimii toiseen laitteen sisällä virtuaalisesti ohjelmistossa käyttäen isäntätietokoneen fyysistä laitteistoa.

VPN

Virtual Private Network. Palvelu, jonka avulla voidaan kommunikoida julkisen verkon kautta esim. yrityksen sisäisessä verkossa.

VÄLIOHJELMISTO

Myös Middleware. Ohjelmisto, jota käytetään useamman ohjelmiston välillä, jotta kommunikaatio ohjelmistolta toiselle onnistuu.

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO **1**

2 TOIMINTAYMPÄRISTÖN JA LÄHTÖTILANTEEN KUVAUS **2**
2.1 Palvelimen arkkitehtuuri **3**
2.2 Palvelimen komponentit **5**
2.3 Palvelimen hyödyntämät kirjastot **8**
2.4 REST API-rajapinta **9**
2.5 Graafinen käyttöliittymä (Admin UI) **12**
2.5.1 Uploads **12**
2.5.2 Downloads **14**
2.5.3 Device Search **15**

3 PROSESSIN KUVAUS **16**
3.1 Uuden tietokannan rakenne lähtötilanteessa **20**
3.1.1 Boards **21**
3.1.2 Projects **21**
3.1.3 Tablenames **21**
3.1.4 Projectpools **22**
3.2 Admin Area **22**

4 ARVIOINTI JA POHDINTA **25**

5 YHTEENVETO JA JOHTOPÄÄTÖKSET **27**

LÄHTEET **36**

KUVIOT

KUVIO 1. Kokonaisuuden arkkitehtuuri lähtötilanteessa. 3
KUVIO 2. Palvelimen arkkitehtuuri. 3
KUVIO 3. Koodien jakeluprosessi testattavalle laitteelle. 10
KUVIO 4. Uuden tietokannan rakenne lähtötilanteessa. 20
KUVIO 5. Uuden kokonaisuuden arkkitehtuuri. 24

KUVAT

KUVA 1. Edellisen tietokannan Status-sivu. 12
KUVA 2. Edellisen tietokannan Uploads-sivu. Koodien lähettäminen palvelimelle tiedostosta oli mahdollista. 13
KUVA 3. Edellisen tietokannan Downloads-sivu. Koodialtaiden sisältämien koodien lataaminen oli mahdollista. 14
KUVA 4. Edellisen tietokannan Device Search-sivu. Tähän ei tehty muutoksia tässä opinnäytetyössä. 15
KUVA 5. Kuvakaappaus uudelta Status-sivulta. 18
KUVA 6. Kuvakaappaus uudelta Uploads-sivulta. 19
KUVA 7. Kuvakaappaus uudelta Downloads-sivulta. 20

KUVA 8. Kuvakaappaus uudelta Admin Area-sivulta. 23

TAULUKOT

TAULUKKO 1. Palvelintietokoneen komponentit 2

1 JOHDANTO

Tässä opinnäytetyössä on tarkoitus kehittää Sasken Finland Oy:lle dynaaminen tietokanta sen edellisen tietokantaratkaisunsa pohjalta. Tarkoitus on luoda monipuolinen ja joustava ratkaisu, jota yrityksen tulevaisuudessa käyttämät testilaitteet hyödyntäisivät. Sasken Finlandin käyttämät testilaitteet hyödyntävät tietokantaa siten, että testilaitteet lähettävät pyyntöjä tietokantaa ylläpitävälle palvelimelle. Testatuista laitteista pidetään kirjaa palvelintietokoneella antamalla niille jäljityskoodi (tracecode), joka tallennetaan tietokantaan. Testilaitteella luetaan myös laitteessa valmiiksi oleva yksilöllinen koodi, joka merkitään käytetyksi tietokantaan.

Yrityksen puolesta tulevaisuuden ratkaisun tulisi sisältää useita koodialtaita (pool). Tarkoituksena on kehittää ratkaisu, jolla on mahdollista luoda näitä altaita tietokantaan joko palvelimen alustusvaiheessa tai internet-selaimella käytettävän käyttöliittymän kautta. Ratkaisua täytyy pystyä muokkaamaan helposti vastaamaan yrityksen tulevaisuuden tarpeita siten, että tulevaisuuden laitteita varten samaa dynaamista tietokantaa pystytään hyödyntämään. Tavoitteena on se, että samaa tietokannan ohjelmointi- ja alustustyötä ei tarvitse tehdä useampaa kertaa aina uusien testilaitteiden ilmaantuessa.

Aluksi ratkaisua kehitettäessä on tarkoitus tehdä ainoastaan toiminnallinen logiikka ilman käyttöliittymää. Käyttöliittymän tekeminen samaan aikaan toiminnallisen logiikan lisäksi on kuitenkin mahdollista, sillä se tekee kehittämisestä sekä loogista että helppoa. Se myös selkeyttää työn tekemistä. Alkuperäinen tietokantaratkaisu sijaitsee yrityksen GitLab-sivustolla, josta lähdekoodi ladataan. GitLabiin päivitetään tietokannan versioita Gitin avulla. Kehitystyö tapahtuu virtualisoidussa ympäristössä, jonka tietokoneessa hyödynnettään eri käyttöjärjestelmää isäntäkoneeseen verrattuna. Virtuaalikoneeseen tulee asentaa kehitystyötä varten kehitystyössä tarvittavia työkaluja, joilla muokataan lähdekoodia. Osa palvelinohjelmistosta hyödyntää Docker-ympäristöä, joten näitä ohjelmia ei tarvitse erikseen manuaalisesti asentaa.

2 TOIMINTAYMPÄRISTÖN JA LÄHTÖTILANTEEN KUVAUS

Opinnäytetyön tavoitteena oli tehdä Sasken Finland Oy:n edellisen tietokannan pohjalta dynaaminen tietokanta yrityksen tulevaisuuden testilaitteiden käyttöön. Myös edellinen tietokanta oli yrityksen testilaitteiden hyödyntämä, joten sen pohjalta dynaamisen ratkaisun tekeminen oli järkevää. Tietojen lisääminen tietokantaan ja poistaminen tietokannasta täytyi olla uudessa ratkaisussa mahdollista eli tietokannan rakennetta täytyi pystyä muuttamaan yrityksen ja testattavien laitteiden tarpeiden mukaan. Edellisessä tietokannassa koodien lataaminen jo olemassa oleviin tauluihin onnistui, mutta uudessa dynaamisessa ratkaisussa uusia tauluja tai koodialtaita täytyi pystyä myös lisäämään ja poistamaan. Yksi vaatimuksista opinnäytetyölle yrityksen puolesta oli myös se, että tietokannan pystyi ottamaan käyttöön siten, että lähtötilanteessa tietokanta sisältäisi ainoastaan tietokannan alustavaan rakentamiseen ja välttämättömään toimintaan tarvittavat taulut. Testilaitteiden hyödyntämiä tauluja oli tarkoitus pystyä muokkaamaan käyttöliittymän kautta kunnes käytettävä tietokanta on yrityksen senhetkisiä tarpeita vastaava.

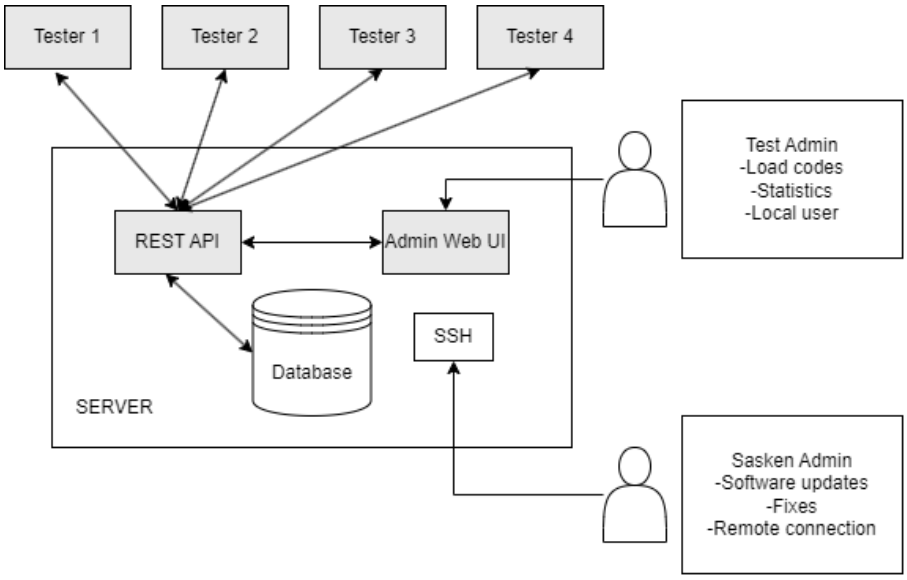
Saskenin dokumentaation perusteella palvelinkäytössä olevan tietokoneen ominaisuudet on ennalta määritelty. Palvelintietokoneen, jossa kokonaisuus suoritetaan, ominaisuudet ovat taulukon 1 mukaiset.

TAULUKKO 1. Palvelintietokoneen komponentit

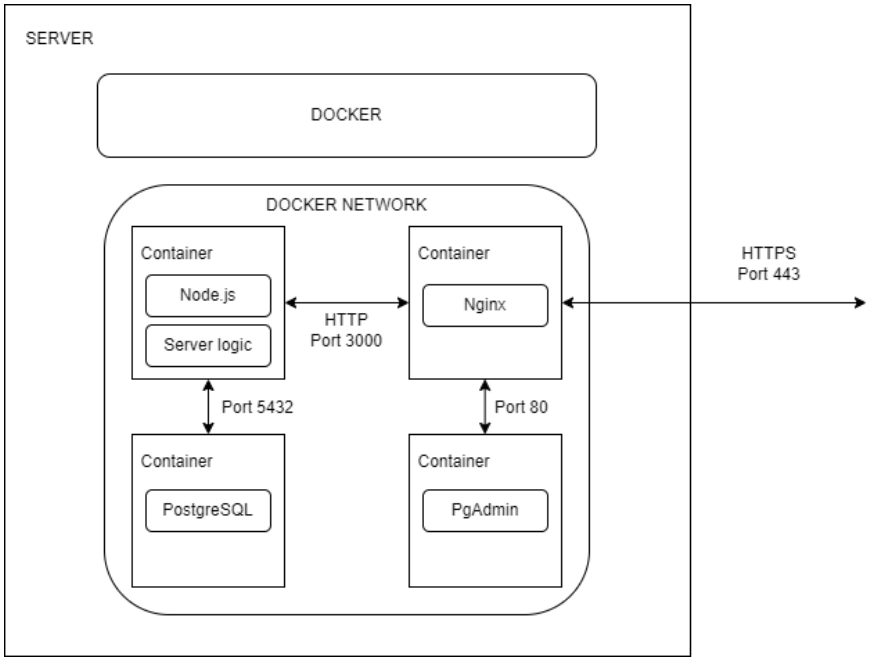
Komponentti	Malli
Prosessori	Intel(R) Core(TM) i5-7500
Näytönohjain	Intel HD Graphics 630
Muisti	8GB DDR4 2133MHz
Kovalevy	2 x Samsung SSD 870 500GB Sata
Verkkokortti	Intel I210 Gigabit Network Connection

2.1 Palvelimen arkkitehtuuri

Palvelimen ympäristö koostuu palvelintietokoneesta, sen käyttäjistä ja testilaitteista (KUVIO 1). Palvelimen komponentit toimivat Dockerin luomassa verkossa, jossa on kontteja (container) (KUVIO 2). Palvelimen ympäristön ja palvelimen arkkitehtuuria selventävät kuviot 1 ja 2.



KUVIO 1. Kokonaisuuden arkkitehtuuri lähtötilanteessa.



KUVIO 2. Palvelimen arkkitehtuuri.

Kuviosta 1 voidaan nähdä, että testien järjestelmänvalvoja (test admin) pystyy tekemään järjestelmänvalvojan web-käyttöliittymän (Admin Web UI) kautta eräitä toimintoja. Hän pystyy lataamaan koodeja tietokantajärjestelmään (load codes) ja tutkimaan koodialtaiden tilastoja (statistics). Hän on paikallinen käyttäjä. SSH-yhteyden (Secure Shell Protocol) avulla Saskenin järjestelmänvalvoja (Sasken Admin) pystyy olemaan etäältä yhteydessä tietokantapalvelimeen. SSH-yhteyden avulla Saskenin järjestelmänvalvoja pystyy päivittämään ohjelmistoja (software updates) ja tekemään korjauksia palvelimeen (fixes). Web-käyttöliittymän kautta ollaan yhteydessä rajapintaan (REST API), jonka kautta saadaan yhteydet testilaitteisiin ja tietokantaan. Rajapintaan lähetetään käyttöliittymän kautta tietoja, jotka sitten rajapinnasta ohjataan tietokannalle. Rajapinnan kautta testilaitteille lähetetään myös koodeja testitilanteissa ja sen kautta saadaan testilaitteilla luetut testattavien laitteiden tunnuskoodit.

Kuvio 2 kuvaa palvelimen arkkitehtuuria. Palvelintietokoneella suoritetaan Docker, joka muodostaa konteista (container) verkon (Docker network). Dockerin verkossa on käynnissä sovelluksia, jotka mahdollistavat tietokannan toiminnan. HTTPS-protokollalla portin 443 kautta voidaan olla yhteydessä Nginx:ään, joka toimii yhdessä Docker-kontissa (container). Toisessa kontissa käytössä on myös PgAdmin, joka on järjestelmänvalvontatyökalu PostgreSQL:lle. PgAdminin avulla voidaan hallita PostgreSQL-tietokantaa. PgAdminiin päästään portin 80 kautta.

Nginxin sisältämän kontin kautta voidaan olla yhteydessä erilliseen konttiin, joka pitää sisällään Node.js:än ja palvelimen logiikan (server logic). Palvelimen logiikka tarkoittaa palvelimen lähdekoodia. HTTP-protokollalla portin 3000 kautta Nginx toimii palvelimen logiikan kanssa (KUVIO 2).

Portin 5432 kautta ollaan lopulta yhteydessä PostgreSQL-tietokannan sisältävään konttiin. Tämä kontti pitää siis sisällään käytännössä tietokantajärjestelmän eli koodialtaat, nimet, projektit ja taulut. Tämä kontti voidaan saavuttaa Nginx:än kontin ja palvelimen logiikan sisältävän kontin kautta. Reitti kaikkiin kontteihin kulkee molempiin suuntiin (KUVIO 2).

2.2 Palvelimen komponentit

Palvelintietokoneessa palvelimen puolella (server-side) tausta (back-end) hyödyntää Node.js:ää. Node.js on palvelinalusta, joka on rakennettu JavaScriptillä. Yksi Node.js:n tärkeimmistä ominaisuuksista on se, että Node on estämätön (non-blocking) palvelin. Tämä tarkoittaa sitä, että paljon resursseja vaativat tehtävät eivät keskeytä palvelimen toimintaa ja siten Node.js pystyy käsittelemään monia yhteyksiä samanaikaisesti. Node.js pystyy myös käsittelemään reaaliaikaista kommunikointia paremmin verrattuna estävään (blocking) palvelimeen. Node.js:ää käytetään pääasiallisesti web-palvelimena (web server), sillä websivujen tarjoaminen (serving) yleensä pitää sisällään tiedostojen lukemista ja yhteyden muodostamista tietokantaan. Useampien käyttäjien palveleminen näiden toimintojen kanssa samanaikaisesti on siis mahdollista, toisin kuin estävässä palvelimessa, jossa lisäpyyntöihin vastaaminen edellyttää resurssien vapautumista. (Johanan, Khan & Zea, 2016.)

Node.js on avoimen lähdekoodin ja monen alustan tukema JavaScript-ympäristö. Se on suosittu työkalu ja sitä voidaan käyttää lähes mihin tahansa projektiin. (Introduction to Node.js.) Node.js:ää hyödynnetään tietokantapalvelimella palvelinpuolen lisäksi myös käyttäjäpuolella (client-side). Yhden kielen käyttö helpottaa sekä palvelin- että käyttäjäpuolen kehitystä.

PostgreSQL on ilmainen, avoimen lähdekoodin objektirelaatiotietokantajärjestelmä (object-relational database), joka käyttää ja laajentaa SQL-kieltä (What is PostgreSQL?). PostgreSQL pitää sisällään kaksiulotteisia tauluja, joissa on tietoa sisältäviä rivejä (data rows) ja sarakkeita (columns), joiden tietotyypit ovat vahvasti määriteltyjä. Relaatiotyylinen tietokanta on kaikista suosituin ja sen uskotaan jatkavan yhtä suosittuna tulevaisuudessakin. Relaatiotietokantojen yleisyys johtuu useista työkaluista (toolkits), tietoturvallisuudesta ja niiden toiminnallisuuden samankaltaisuudesta ohjelmoijien ajatusten perusteella tapahtuvan toiminnan kanssa. Muista tietosäilöistä (datastore) poiketen tietojen käytön suunnittelu etukäteen ei ole tarpeellista. PostgreSQL on ollut olemassa nykyisessä muodossaan vuodesta 1995. Alunperin projekti kirjoitettiin Berkeleyssä 1970-luvun alussa ja sen nimi oli Interactive Graphics and Retrieval System tai Ingres. 1980-luvulla julkaistiin kehittyneempi versio nimeltä post-Ingres. Tämän nimi lyheni muotoon Postgres. Projekti lopetettiin Berkeleyllä vuonna 1993, mutta avoimen lähdekoodin yhteisö jatkoi projektia nimellä Postgres95. Myöhemmin se nimettiin PostgreSQL:ksi vuonna 1996. Nimellä alleviivattiin sen siihen aikaan uutta SQL-tukea. PostgreSQL:n laajennukset mahdollistavat mm. luonnollisen kielen erottamiseen (natural-language parsing), moniulotteiseen indeksointiin (multidimensional indexing), maantieteellisiin kyselyihin (geographic queries) ja

omiin datatyyppeihin (custom datatypes) liittyviä toimintoja. Se sisältää kehittyneen tiedonsiirron (transaction handling), valmiiksirakennettuja proseduureja monille kielille ja sitä voidaan hyödyntää monilla alustoilla. PostgreSQL:ssä on myös sisäänrakennettu Unicode-tuki, sekvenssit (sequences), taulun periytyminen (table inheritance) ja SQL:n lisävalinnat (subselects) ja se on yksi markkinoiden eniten ANSI SQL-sopivimmista relaatiotietokannoista. Se on nopea, luotettava, pystyy käsittelemään teratavuja dataa ja sitä hyödynnetään projekteissa kuten Skype, Ranskan Caisse Nationale d'Allocations Familiales (CNAF) ja Yhdysvaltojen Federal Aviation Administation (FAA). (Redmond & Wilson, 2012.) PgAdmin on avoimen lähdekoodin graafinen käyttöliittymä PostgreSQL:n tietokannan hallinnointiin. PgAdmin voidaan suorittaa joko web-sovelluksena tai työpöytäsovelluksena. (FAQ.) Palvelintietokoneella suoritettava mm. testattavien laitteiden tunnuskoodeja (identifier) ylläpitävä tietokanta on toteutettu PostgreSQL:llä.

Nginx on HTTP-palvelin ja käänteinen välityspalvelin, sähköpostivälityspalvelin ja tyypillinen TCP/UDP-välityspalvelin. Sitä käytetään esim. venäläisillä sivustoilla kuten Yandex, Mail.Ru ja VK. Sen ominaisuuksiin kuuluvat mm. tiedostojen jakaminen palvelimelta, kiihdytetty käänteinen välityspalimena toimiminen välimuistin kera, datakuorman tasapainottaminen ja virhekestoisuus sekä SSL- ja TLS SNI-tuet. (Nginx.)

Nginx on alunperin kotoisin Venäjältä ja se lausutaan "engine X". Projektin aloitti vuonna 2002 Igor Sysoev. Hänen tarkoituksenaan oli luoda palvelin venäläiselle paljon käytetylle sivustolle nimeltä Rambler, jolle saapui syyskuussa 2008 yli 500 miljoonaa HTTP-pyyntöä päivässä. Nykyisin Nginx:ää käytetään mm. suosituimpien internetsivujen tarjoamiseen, kuten esim. Reddit, Wikipedia, WordPress ja Dropbox. Nginx:n on todistettu olevan erittäin tehokas ja kevyt mutta samalla myös voimakas webpalvelin. Nykyään monet yritykset ovat siirtyneet käyttämään Nginx:ää Apachen sijaan.
Nginx on monella tapaa kilpailijoitaan tehokkaampi webpalvelin. Yksi sen tärkeimmistä ominaisuuksista on nopeus. Nginx hyödyntää asynkronisia pistokkeita (asynchronous sockets), joten Nginx ei luo prosesseja saatujen pyyntöjen määrän mukaan. Yksi prosessi ydintä kohden riittää tuhansien yhteyksien käsittelemiseen. Tämä mahdollistaa pienemmän prosessorin kuormituksen ja muistinkäytön. Toinen tärkeä ominaisuus on helppokäyttöisyys. Määritystiedostot (configuration files) ovat helpommin luettavissa ja muokattavissa esim. Apacheen verrattuna ja muutama rivi riittää kokonaisen virtuaalisännän määrittämiseen. Kolmas Nginx:n tärkeä ominaisuus on muutettavuus. Nginx on BSD-lisenssin alainen avoimen lähdekoodin ohjelmisto ja se sisältää vahvan kirjaston laajennuksia (plug-in), joita kutsutaan moduuleiksi (modules). Nginx:n jakeluversiot sisältävät paljon moduuleita ja monia kolmannen osapuolen moduuleja voidaan myös ladata internetistä. Kaiken kaikkiaan nopeus, tehokkuus ja

suorituskykyisyys ovat Nginx:n tärkeimpiä ominaisuuksia. (Fjordvald & Nedelcu, 2018.) Nginx on vastuussa mm. pyyntöjen edelleenohjauksesta taustan ohjelmistoille.

Docker on avoimen lähdekoodin alusta, jota käytetään eristetyn ympäristön sovellusten rakentamiseen, ajamiseen ja hallinnoimiseen. Eristetyt ympäristöt ovat nimeltään kontteja (container). Kontit ovat suoritettavia komponentteja, jotka yhdistävät sovelluksen lähdekoodin, käyttöjärjestelmän kirjastot ja riippuuvuusohjelmistot (dependencies), joita tarvitaan ohjelman suorittamiseen. Dockerin etuja ovat ohjelmistojen keveys virtuaalikoneisiin verrattuna, nopeampi ohjelmiston kehitys ja resurssien käyttö. Docker Compose on Dockerin työkalu, jolla otetaan käyttöön useampia kontteja samalla isäntätietokoneella (host). Docker Compose luo YAML-tiedoston, joka määrittää, mitä palveluita käytetään sovelluksen kanssa. Työkalu pystyy alustamaan ja suorittamaan monia kontteja yhdellä komennolla. Lukuisat ohjelmointikielet voivat hyödyntää YAML-syntaksia käyttäviä tiedostoja. Kehittäjät voivat käyttää Docker Composea myös esimerkiksi pysyvän, ennaltamääritellyn kokoisen tallennustilan määrittämiseen ja palvelujen riippuvuuksien (dependency) dokumentointiin ja määrittämiseen. (What is Docker?) Palvelintietokoneessa on käynnissä sovelluksia monissa Dockerin konteissa. Jokainen sovellus toimii omassa kontissaan. Konteissa olevia sovelluksia ovat Node.js-palvelin, PostgreSQL ja Nginx. Dockerin konteissa suoritettavat sovellukset ovat välttämättömiä palvelintietokoneen toiminnan kannalta.

2.3 Palvelimen hyödyntämät kirjastot

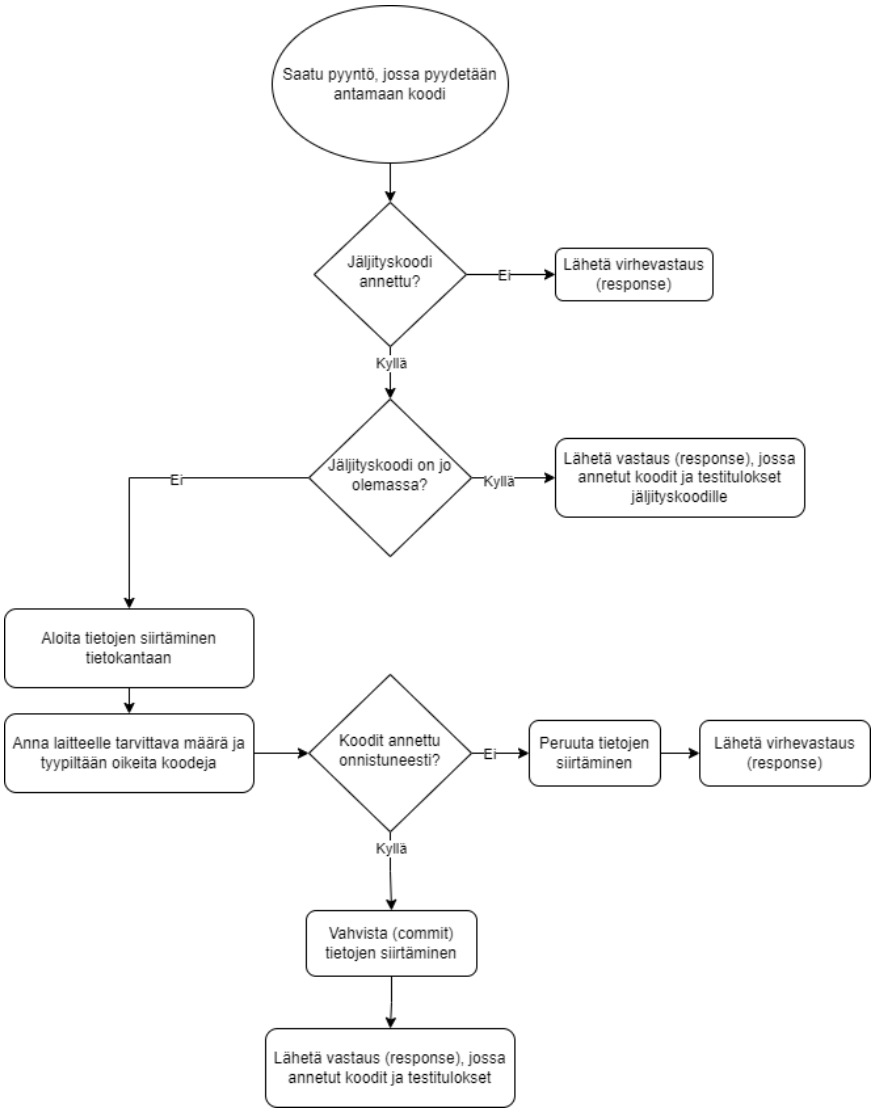
Palvelintietokone hyödyntää monia kirjastoja, jotka mahdollistavat palvelimen oikeanlaisen toiminnan. Nämä kirjastot ovat ilmaiseksi saatavilla. Yksi kirjastoista on Express.js. Express.js on nopea ja minimalistinen websovellusten viitekehys (framework) Node.js:lle. Sen ominaisuuksiin kuuluvat mm. kestävä reititys, hyvä suorituskyky ja HTTP-pyyntöjen käsittely. (Express.) Express-fileupload on yksinkertainen Expressin väliohjelmisto (middleware). Sitä käytetään tiedostojen lähettämiseen. (Express-fileupload.)

PostgreSQL:n käyttöä helpottavat monet kirjastot. Node-postgres on kokoelma node.js-moduuleja, joiden avulla ollaan yhteydessä PostgreSQL-tietokantaan. Se sisältää tuen vastakutsuille (callbacks), lupauksille (promises), asynkronisille toiminnoille (async/await), yhteyksien yhdistämiselle (connection pooling), valmistelluille käskyille (prepared statements), kursoreille (cursors), suoratoistetuille tuloksille (streaming results), sidokset C-kieliin (C/C++ bindings), laajan tyypin jäsentämiseen (rich type parsing) ja moneen muuhun. (Node-postgres.) Tätä kirjastoa käytetään palvelimella silloin, kun tarvitaan tietoja koodeja sisältävästä tietokannasta. Node-pg-format on Node.js-toteutus PostgreSQL:n format()-funktiosta. Sitä käytetään turvallisesti dynaamisten SQL-pyyntöjen luomiseen. SQL-tunnisteiden ja -literaalien (identifiers and literals) käyttö estetään SQL-injektion (SQL injection) torjumisen auttamiseksi. Toiminta on samankaltaista PostgreSQL:n format()-funktion kanssa. Moduuli myös tukee Noden puskureita (Node buffers), taulukoita (arrays) ja objekteja (objects). (Node-pg-format.) Pg-copy-streams-kirjasto antaa luettavan (readable) tai kirjoitettavan (writable) tietovirran (stream) suoraan tietokannan tauluun tai taulusta ulos. Tällainen yhteys tauluun on erittäin nopea ja läpitunkeva. (Pg-copy-streams.) Suoratoisto-ominaisuutta käytetään palvelimella koodien lataamisessa tietokannasta ja lähettämisessä tietokantaan.

Lisäksi palvelintietokoneella on käytössä myös muita hyödyllisiä kirjastoja. Morgan on HTTP-pyyntöjen kirjaustyökalu (HTTP request logger) (Morgan). Validator on kirjasto, joka sisältää merkkijonojen vahvistajia (validators) ja siistijöitä (sanitizers) (Validator). Material UI on komponenttikirjasto, joka sisältää oman toteutuksen Googlen Material Design-järjestelmästä (MUI Core). Käyttäjäpuolella (client-side) käytetään Reactia. React on kirjasto, jota käytetään web- ja natiivikäyttöliittymien (native user interface) rakentamiseen (React).

2.4 REST API-rajapinta

REST API on ohjelmointirajapinta (Application Programming Interface, API). API on kokoelma sääntöjä, jotka määrittävät sen, miten sovellukset tai laitteet voivat yhdistää toisiinsa ja kommunikoida toistensa kanssa. Se on mekanismi, jonka avulla sovellus tai palvelu saa yhteyden toisen sovelluksen tai palvelun resurssiin. REST API on ohjelmointirajapinta, joka mukautuu REST:in suunnitteluperiaatteisiin. REST tulee englannin kielen sanoista *representational state transfer*. REST on arkkitehtuurin tyyli. REST API tunnetaan joskus myös nimellä RESTful API. REST määriteltiin ensimmäisen kerran vuonna 2000 ja se mahdollistaa hyvän joustavuuden ja vapauden kehittäjille. (What is a REST API?) REST API-rajapintaa käytetään palvelimella tietokannan ja testilaitteiden väliseen kommunikaatioon (KUVIO 1). Testilaitteet lähettävät tietoja rajapintaan (API) ja PostgreSQL:ään lähetetään edelleen SQL-pyyntöjä, jotka sisältävät rajapintaan lähetettyjä tietoja. Kommunikointiin käytetään HTTPS-protokollaa ja lähetetyn datan formaatti on JSON. Esimerkkinä on tilanne, jossa testatulle laitteelle asetetaan tietokannasta löytyvä koodi. Silloin palvelin saa laitteen jäljityskoodin (tracecode), laitteen tyypin (esim. LAITE1) ja tarvittavien koodien (identifier) määrän. Tämän jälkeen testattavan laitteen tiedot tallennetaan tauluun ja palvelin antaa tietyn määrän koodeja ja merkitsee ne käytetyksi tietokantaan. Prosessi ilmenee kuviosta 3.



KUVIO 3. Koodien jakeluprosessi testattavalle laitteelle.

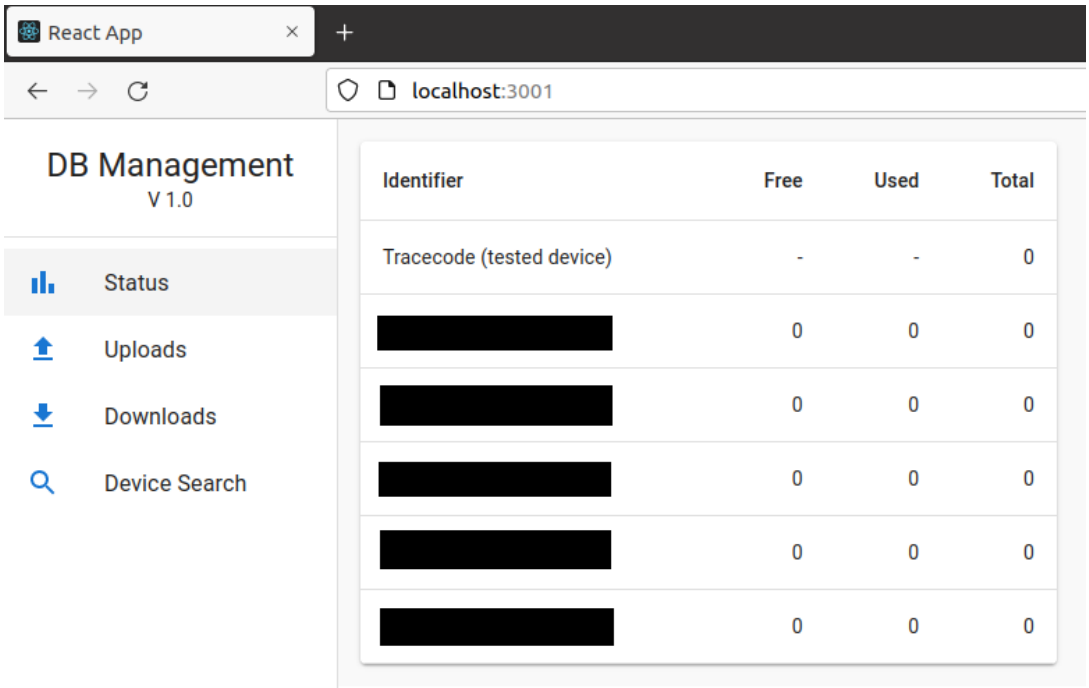
Kuviossa 3 ensin rajapintaan saapuu pyyntö, jossa pyydetään järjestelmää antamaan testattavalle laitteelle yksilöllinen tunnuskoodi (identifier). Mikäli testattavalle laitteelle ei ole annettu jäljityskoodia (tracecode), lähetetään virhevastaus (response), jonka avulla käyttäjä tietää prosessin epäonnistuneen. Mikäli jäljityskoodi on annettu, mutta testattavalla laitteella on jo sellainen olemassa, lähetetään vastaus, joka sisältää testatulle laitteelle aiemmin annetut koodit sekä jäljityskoodia vastaavan laitteen testitulokset. Näin ollen testattava laite ei saa uudestaan yksilöllisiä koodeja, sillä ne on jo annettu aiemmin jakeluprosessin seurauksena ja testit laitteelle on jo suoritettu (KUVIO 3).

Mikäli testattavalla laitteella ei ole jäljityskoodia, silloin tietokantaan tullaan siirtämään testattavan laitteen tyyppi ja testitulokset. Nämä tallentuvat boards-tauluun. Tämän jälkeen testatuille laitteille py-

ritään antamaan laitteiden määrää vastaava määrä tunnuskoodeja (identifier). Mikäli testatuille laitteille ei voida antaa tunnuskoodeja, testitulosten ja laitteen tyypin siirtäminen tietokantaan peruutetaan ja rajapinnasta lähetetään virhevastaus. Käyttäjä näkee käyttöliittymään ilmestyvän viestin perusteella, että toiminto epäonnistui. Mikäli koodit pystytään antamaan, silloin vahvistetaan testitulosten ja laitteen tyyppitiedon siirtäminen tietokantaan ja lähetetään vastaus (response), jossa lukee juuri testatuille laitteille annetut tunnuskoodit ja laitteiden testitulokset.

2.5 Graafinen käyttöliittymä (Admin UI)

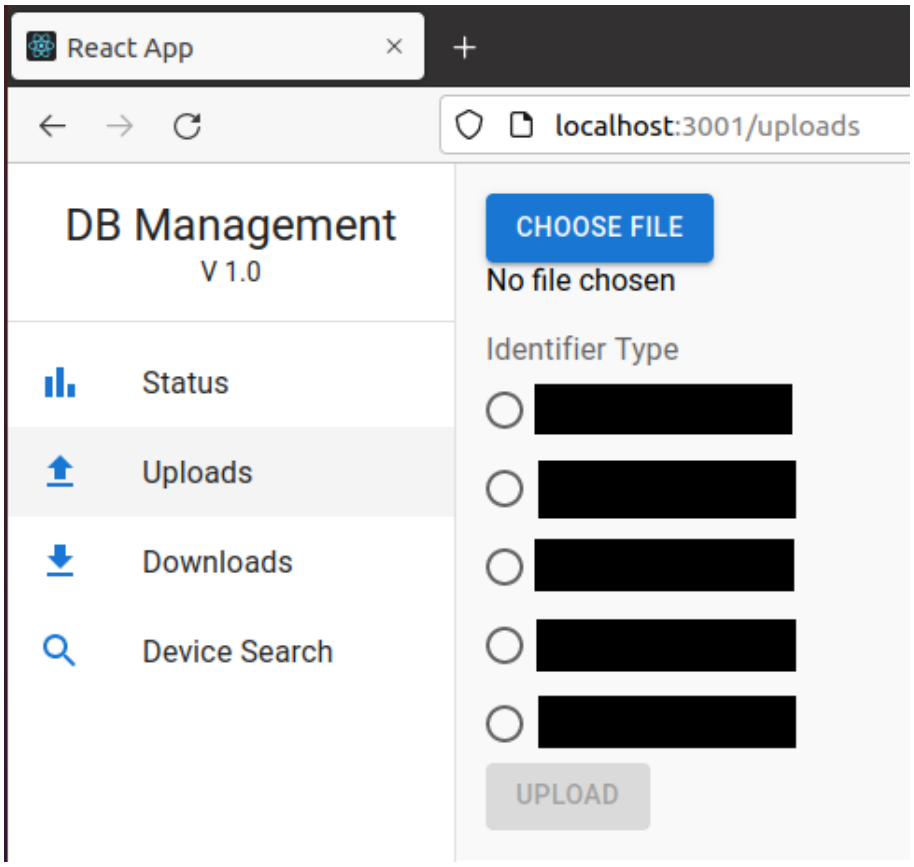
Graafinen käyttöliittymä eli Admin UI on sivusto, josta hallinnoidaan tietokantaa ja sen sisältämiä koodeja. Status-sivulta (KUVA 1) nähdään sillä hetkellä tietokannan sisältämien yksittäisen koodityypin käyttämättömien koodien lukumäärä, käytettyjen koodien lukumäärä sekä tyypin koodien kokonaismäärä.



KUVA 1. Edellisen tietokannan Status-sivu.

2.5.1 Uploads

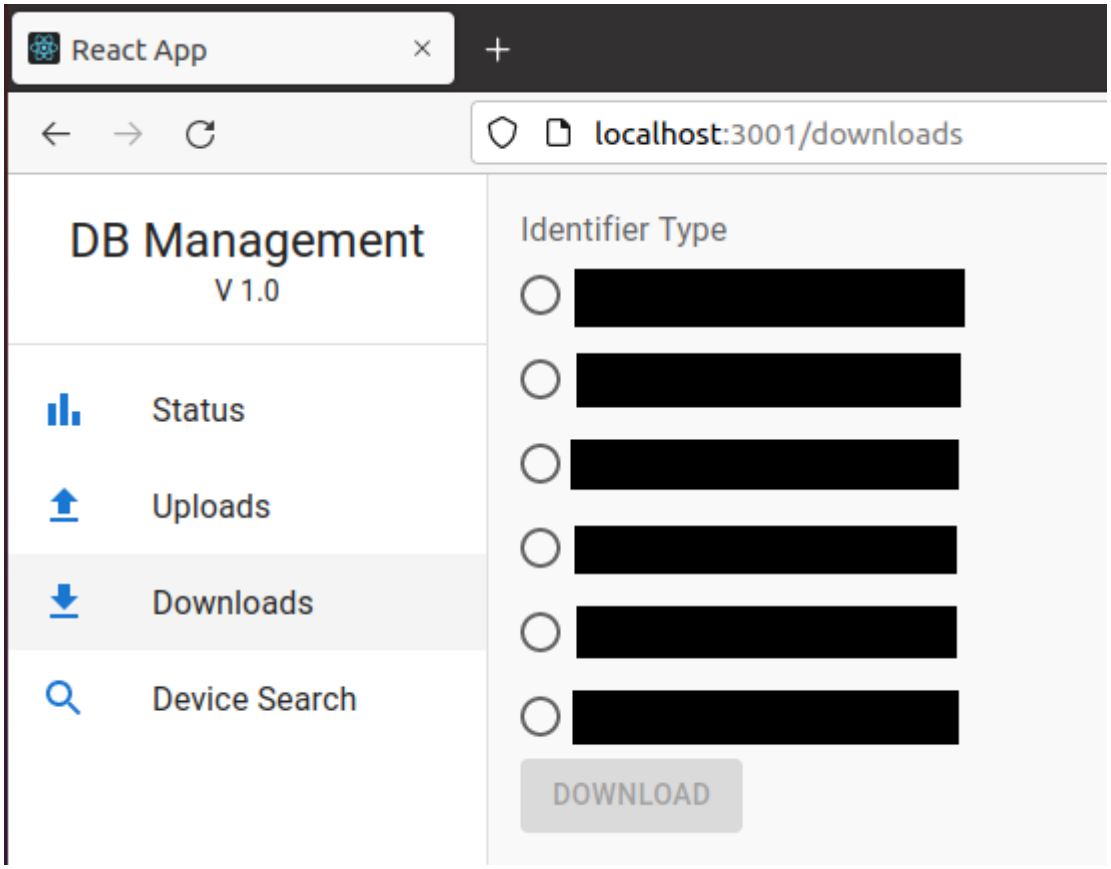
Uploads-sivulta (KUVA 2) tietokantaan voidaan ladata yksittäisen koodityypin altaaseen lisää koodeja. Käyttäjä voi valita koodeja sisältävän tiedoston tietokoneeltaan ja sen jälkeen valita yhden listan koodityypeistä. Sen jälkeen koodit voidaan ladata painamalla Upload-nappia. Mikäli koodeja ladattaessa tietokantaan ilmenee ongelmia, ne ilmoitetaan Upload-napin alapuolella sijaitsevassa viestissä.



KUVA 2. Edellisen tietokannan Uploads-sivu. Koodien lähettäminen palvelimelle tiedostosta oli mahdollista.

2.5.2 Downloads

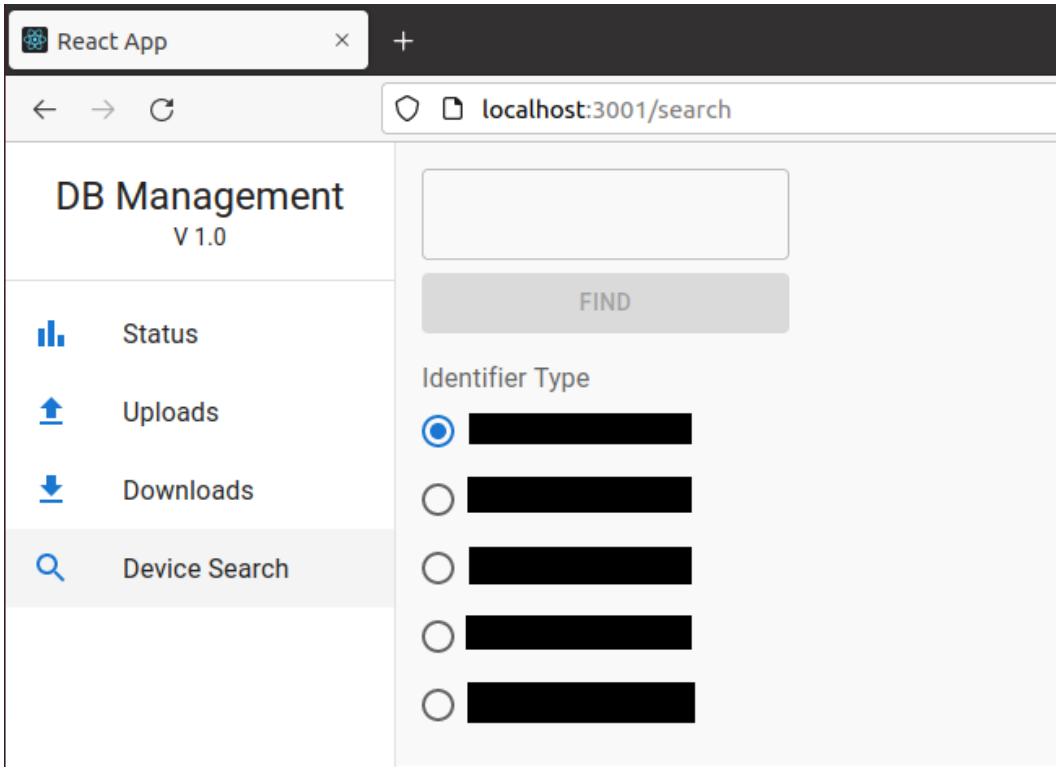
Downloads-sivulta (KUVA 3) voidaan ladata yksittäisen koodityypin koodit tietokannasta tietokoneelle. Palvelimen ohjelmisto koostaa koodit .csv-tiedostomuotoon, jota voidaan lukea esim. taulukkolaskentasovelluksella.



KUVA 3. Edellisen tietokannan Downloads-sivu. Koodialtaiden sisältämien koodien lataaminen oli mahdollista.

2.5.3 Device Search

Device Search-sivulta voidaan etsiä tekstikenttään kirjoitettua nimeä vastaavien laitteiden tiedot. Tässä opinnäytetyössä Device Search-sivua ei ole kehitetty eteenpäin.



KUVA 4. Edellisen tietokannan Device Search-sivu. Tähän ei tehty muutoksia tässä opinnäytetyössä.

3 PROSESSIN KUVAUS

Ennen kehitystyön aloittamista minun täytyi asentaa virtualisoidulle tietokoneelle eri sovelluksia, jotka mahdollistivat palvelimen toiminnan ja kehittämisen. Alkuperäiset palvelimen tiedostot sain ladattua Sasken Finlandin GitLab-palvelimelta. Samalle GitLab-sivustolle päivitin myös edistystä työssäni. Jotta sain ladattua uusia versioita kehittämästäni ohjelmistosta GitLabiin, minun täytyi asentaa GitHub Desktop-sovellus. Päädyin GitHub Desktopiin Gitin sijasta, sillä koin työpöytäversion olevan helppokäyttöisempi perinteiseen komentorivillä toimivaan Gitiin verrattuna. Saskenin GitLabiin pääsy vaati lisäksi myös VPN-yhteyttä, joka mahdollistettiin GlobalProtect-ohjelmistolla. GitHub Desktop-sovellusta ei ollut saatavilla Ubuntu-käyttöjärjestelmälle, joten päädyin asentamaan sen isäntätietokoneelle eli koneelle, jossa virtuaaliympäristö oli käynnissä. GitHub Desktop oli kohtuullisen yksinkertainen sovellus, mutta se ei kuitenkaan ratkaissut täysin tiedostojen siirtoon liittynyttä ongelmaa. Monia tiedostoja oli vaikea siirtää samanaikaisesti virtuaaliympäristöstä isäntäkoneelle. Ongelma ratkesi kuitenkin molempien käyttöjärjestelmien jaetulla kansiolla, jota molemmat järjestelmät pystyivät käyttämään. Jaettu kansio oli mahdollista ottaa käyttöön VMWare Workstation Player-sovelluksella. Jaetun kansion avulla pystyin siirtämään muutetut tiedostot virtuaalikoneelta isäntäkoneelle ja lopulta päivittämään ne GitLabiin.

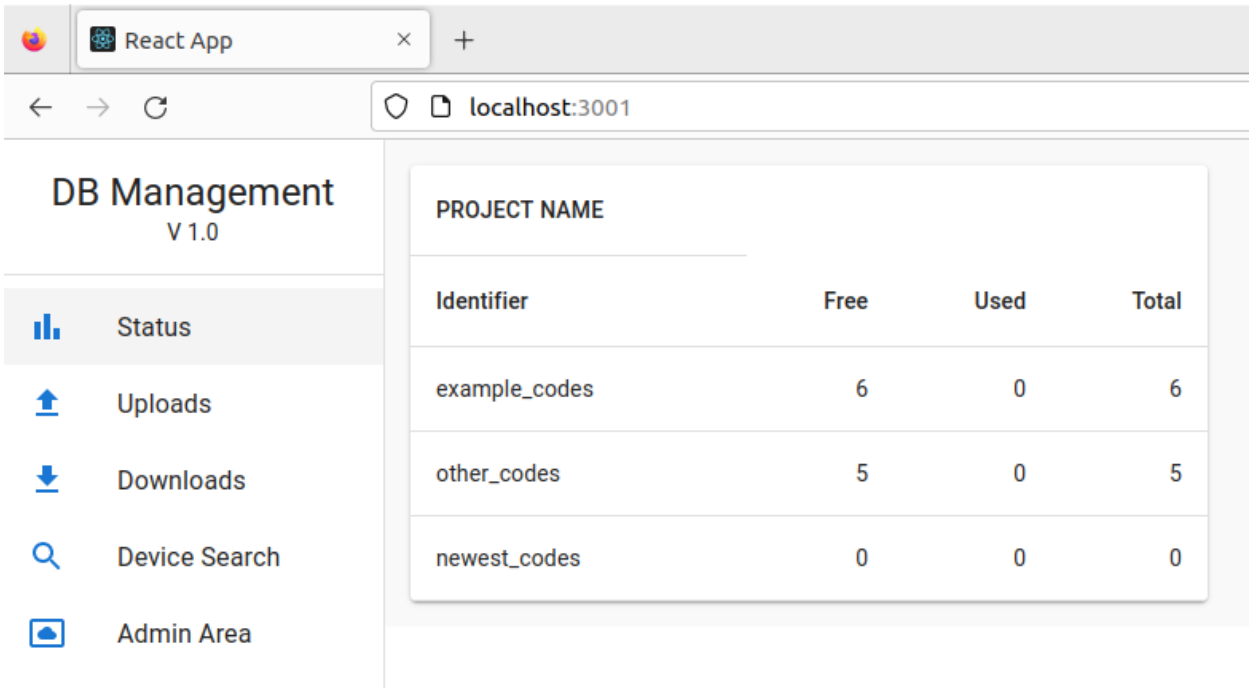
Kehitin dynaamista tietokantaa virtuaalitietokoneella, joka luotiin VMWare Workstation 17 Player-sovelluksella. Saskenin ohjeistuksen mukaan virtuaalitietokoneella käytettävän käyttöjärjestelmä tuli olla Ubuntu, jonka versio oli 20.04 LTS. Lähdin aluksi kuitenkin kehittämään tietokantajärjestelmää Ubuntun versiolla 22.04.1, joka oli senhetkinen Ubuntun uusin versio. Lopulta ensimmäinen virtuaalitietokone lakkasi toimimasta tallennustilan puutteesta johtuen. Tämä ei kuitenkaan johtunut Ubuntun versiosta. Kun ongelma ilmeni, tein tämän jälkeen uuden virtuaalitietokoneen, johon asensin Saskenin ohjeistuksen mukaisen version 20.04 LTS. Käytännön eroja Ubuntun versioissa ei ollut, mutta asentamalla oikean version lopulta toiseen virtuaalitietokoneeseen varmistuin siitä, että tietokantaratkaisu toimi oikein kummallakin Ubuntun versiolla. Käyttöjärjestelmien näköistiedostot sain ladattua Ubuntun viralliselta internetsivustolta.

Palvelimen toimintaa varten virtuaalitietokoneelle tuli asentaa yrityksen toimittaman ohjedokumentaation mukaisesti Docker, jonka versio tuli olla 20.10.17 (build 100c701) sekä Python, jonka version täytyi olla 3.10.4. Ennen näiden asentamista Ubuntun Terminal-sovelluksella tuli asentaa Node Package Manager eli NPM. Sovellukset asennettiin Ubuntun Terminal-sovelluksen avulla. Nämä sovellukset

täytyi asentaa ennen tietokannan käynnistämistä. Virtualisoidussa palvelintietokoneessa suoriutui monia palvelimen toiminnan kannalta välttämättömiä sovelluksia Docker-konteissa. Näitä sovelluksia ei tarvinut manuaalisesti asentaa vaan ne suoriutuivat Docker-kontteja luotaessa. Nämä sovellukset olivat Nginx (versio 1.23.1), PostgreSQL (versio 14) ja pgAdmin4 (versio 6.13). Lisäksi palvelinkoneessa suoriutui OpenSSH. Kehitystyötä helpottamaan asensin virtuaalikoneelle myös Visual Studio Code-ohjelmiston, jonka avulla lähdekoodin kokonaisvaltainen muokkaus oli helpompaa. Käytin alkuvaiheessa nopeiden muokkausten tekemiseen myös Notepad++-sovellusta, jonka latasin heidän sivustoltaan. Lisäksi virtuaalitietokoneelle tuli asentaa Node Version Manager eli NVM, jolla päivitettiin Node.js versioon 16.17.0. Kehitystyöhön käytetty tietokone oli Lenovo ThinkPad T14 -kannettava tietokone.

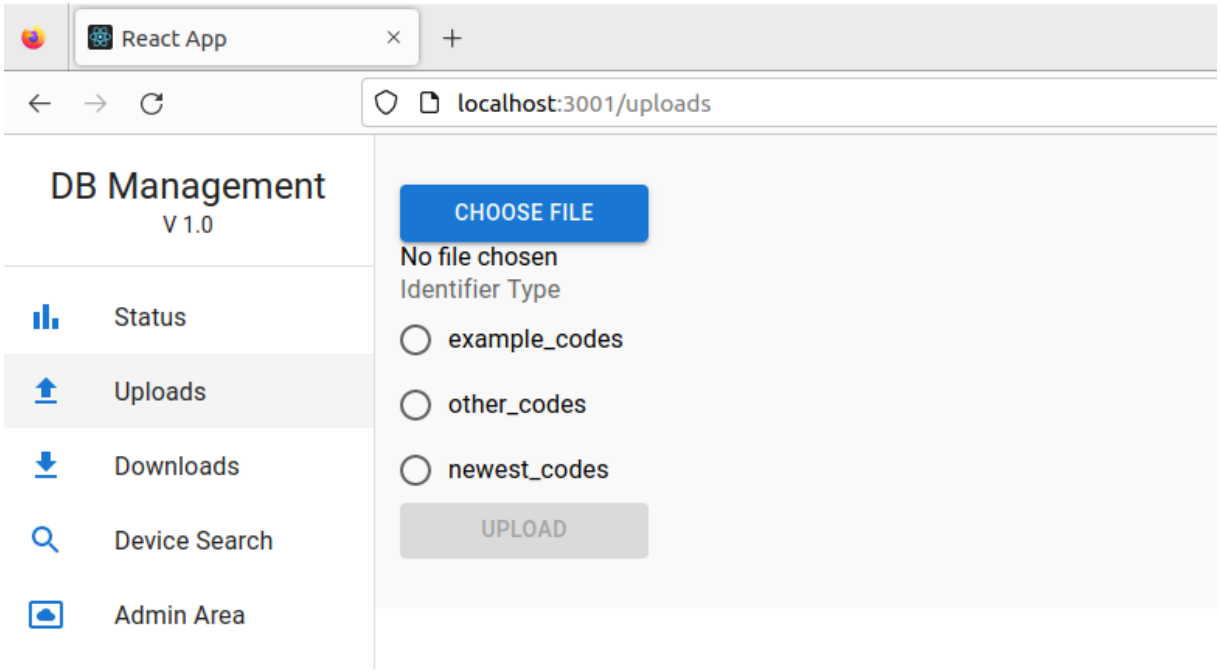
Aloitin dynaamisen tietokannan kehittämisen Saskenin tuottaman vanhan tietokannan pohjalta. Uuden ratkaisun tavoin myös edellinen tietokanta oli ollut myös Saskenin testilaitteiden käytössä. Nyt tarkoituksena oli kehittää tätä ratkaisua eteenpäin, jotta sama tietokantaratkaisu olisi käytännöllinen myös tulevaisuuden testilaitteille. Lähdekoodiin tehtiin muutoksia yrityksen vaatimusten perusteella.

Status-sivulla näkyvä muutos oli dynaaminen koodialtaiden listaus. Alkuperäisessä tietokannassa käytettyjen testilaitteiden koodialtaat oli ohjelmoitu näytettäväksi sivun tauluun, mutta lista altaista ei ollut dynaaminen, sillä se ei suoraan perustunut tietokannan koodialtaisiin. Lista koodialtaista oli kirjoitettu suoraan lähdekoodin taulukoihin (array). Sivun vapaiden koodien, käytettyjen koodien ja koodien kokonaismäärät perustuivat tietokannan tietoihin. Määrät saatiin laskettua tietokannasta SQL-kyselyiden palauttamista tuloksista. SQL-kyselyiden suorittamiseen käytettiin Axios-komponenttia, jolla lähetettiin kyselyitä rajapintaan. Rajapintaan palautui SQL-dataa. Jotta Status-sivu saatiin dynaamiseksi, päädyin muuttamaan Axios-komentoa siten, että getStats-niminen funktio palauttaa koodien määrien lisäksi rajapintaan myös koodialtaiden nimet. Muutin hieman siis rajapinnan funktion sisältämää SQL-kyselyä. Tehdyn muutoksen lopputulos näkyy kuvassa 5, jossa muokatun käyttöliittymän kautta luotujen koodialtaiden nimet ja niihin ladatut koodit näkyvät.

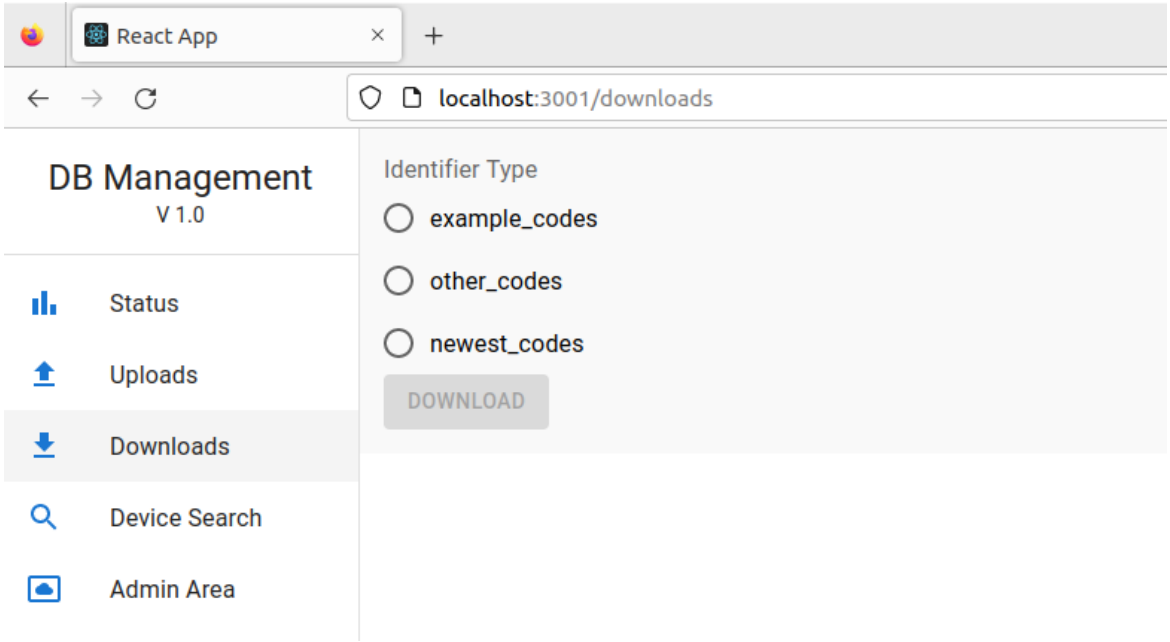


KUVA 5. Kuvakaappaus uudelta Status-sivulta.

Uploads- ja Downloads-sivuille tehdyt muutokset olivat samankaltaisia Status-sivulle tehtyjen muutoksien kanssa. Alunperin sivujen näyttämät listat koodialtaista olivat suoraan lähdekoodiin kirjoitettuja eikä niitä pystynyt hyvin muokkaamaan dynaamiseksi. Koodien lähettämistä ja lataamista varten päädyin muokkaamaan Uploads- ja Downloads-sivujen Axios-käskyjä siten, että lataaminen ja lähettäminen uusiin käyttöliittymällä luotuihin koodialtaisiin onnistui. Lähdekoodi siis luki valitun koodialtaan nimen käyttäjän valinnan perusteella ja latasi koodit oikeaan altaaseen. Uploads- ja Downloads-sivuilla dynaaminen listaus koodialtaista saatiin näkyviin Status-sivun tavoin getStats-funktion avulla. Käytin samaa funktiota siksi, jotta samaa pyyntöä ei olisi tarvinut tehdä jokaisella eri sivulla uudestaan. Tehdyt muutokset ilmenevät kuvista 6 ja 7.



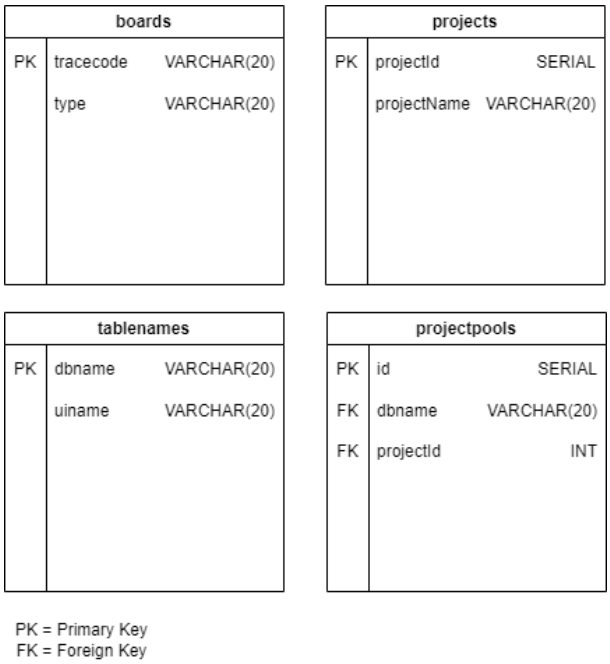
KUVA 6. Kuvakaappaus uudelta Uploads-sivulta.



KUVA 7. Kuvakaappaus uudelta Downloads-sivulta.

3.1 Uuden tietokannan rakenne lähtötilanteessa

Uuden, dynaamisen tietokannan lähtötilanteessa tietokanta muodostuu ainoastaan välttämättömistä tauluista, joita tarvitaan palvelimen alustavaan toimintaan. Päädyin tilanteeseen, jossa näitä tauluja ovat *boards*, *projects*, *tablenames* ja *projectpools*. Uuden tietokannan lähtötilanne ilmenee kuviosta 4.



KUVIO 4. Uuden tietokannan rakenne lähtötilanteessa.

3.1.1 Boards

Boards-taulu pitää sisällään lähtötilanteessa sarakkeet (columns), jotka ovat jäljityskoodi (*tracecode*) ja testatun laitteen tyyppi (*type*). Jäljityskoodi on ensisijainen avain (primary key, PK). Graafisesta käyttöliittymästä voidaan lisätä avaimia (key) ja arvoja (value) tarvittaessa. Esimerkiksi mikäli testitilanteen tulos täytyy tallentaa tietokantaan, sitä varten voitaisiin luoda tähän tauluun esim. sarake nimeltä *test_passed*, jonka arvo olisi TRUE tai FALSE. Jäljityskoodilla (*tracecode*) testattu laite sidotaan omaan yksilölliseen koodiin.

- tracecode VARCHAR(20) PRIMARY KEY
- type VARCHAR(20)

3.1.2 Projects

Yrityksessä käynnissä oleville ja tietokantaa hyödyntäville projekteille on tarkoituksena olla omat tiedot *projects*-taulussa. *Projects*-taulun yksi rivi sisältää projektin id:n (*projectId*), joka on ensisijainen avain ja automaattisesti lisääntyvä numero (SERIAL), sekä projektin nimen (*projectName*). *Projects*-taulua ei hyödynnetä palvelimen nykyisessä versiossa, mutta sen olemassaolosta on hyötyä tulevaisuutta ajatellen.

- projectId SERIAL PRIMARY KEY
- projectName VARCHAR(20)

3.1.3 Tablenames

Tablenames-taulu pitää sisällään koodialtaista käytettävät nimet. Selkeyden vuoksi ajattelin tietokannassa ja käyttöliittymässä käytettävien nimien olevan hieman erilaisia. Esimerkiksi erään koodialtaan nimi voi tietokannassa olla nimeltään *test_codes*, mutta käyttöliittymässä tämä näyttäisi paremmalta muodossa *Test Codes*. *Tablenames*-taulua ei hyödynnetä palvelimen nykyisessä versiossa, mutta myös

sen olemassaolosta on hyötyä tulevaisuutta ajatellen. *Tablenames*-taulu sisältää avaimet *dbname*, joka on ensisijainen avain, ja *uiname*.

- dbname VARCHAR(20) PRIMARY KEY
- uiname VARCHAR(20)

3.1.4 Projectpools

Projectpools-taulussa koodialtaisiin yhdistetään projekti-id:t. Projektit ovat kokoelma koodialtaita (pool) ja yhteen projektiin voi kuulua yksi tai useampi koodeja sisältävä allas. Yksi koodiallas, joka tunnetaan nimellä (*dbname*) yhdistetään projekti-id:seen (*projectID*). Kun projektin ID yhdistetään projektin nimeen *projects*-taulun avulla, voidaan yhdistää tietyt koodialtaat tiettyyn projektiin. Tätä ominaisuutta voidaan hyödyntää tulevaisuudessa, jos tietyt koodialtaat halutaan näkymään tietyn projektin alaisuudessa Status-sivulla. *Projectpools*-taulua ei hyödynnetä palvelimen nykyisessä versiossa, mutta myös sen olemassaolosta on hyötyä tulevaisuutta ajatellen. *Projectpools* sisältää ID:n (id), joka on automaattisesti lisääntyvä numero sekä ensisijainen avain. Taulu sisältää myös kaksi vierasta avainta (foreign key, FK). Nämä ovat *dbname*, joka vastaa tiettyä nimeä *tablenames*-taulussa sekä *projectId*, joka vastaa tiettyä projekti-id:tä *projects*-taulussa.

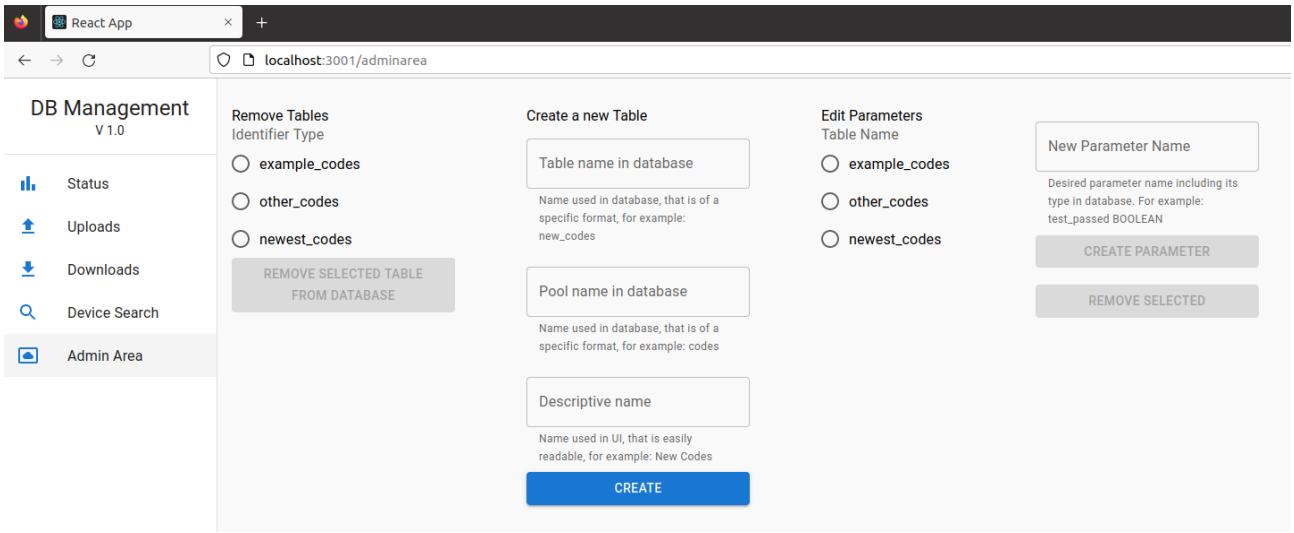
- id SERIAL PRIMARY KEY
- dbname VARCHAR(20) FOREIGN KEY
- projectId INT FOREIGN KEY

3.2 Admin Area

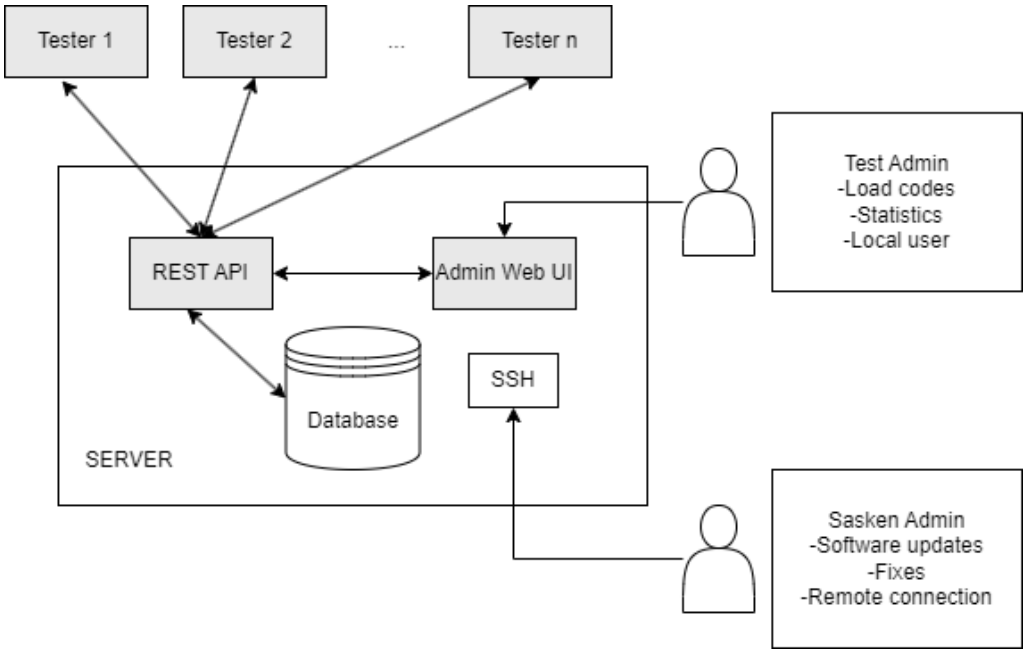
Admin Area-sivun (KUVA 8) luominen palvelimelle oli yksi yrityksen vaatimuksista ja tekemistäni muutoksista. *Admin Area*-sivulta voidaan hallinnoida kokonaisvaltaisesti tietokannan tauluja, sarakkeita ja datatyyppejä. Tulevaisuuden tarkoituksena Admin Area-sivu edellyttää erityistä kirjautumista, mutta kehitystilanteessa yksi kirjautuminen sivustolle riittää. Admin Area-sivulla käyttäjä voi poistaa valitsemansa koodeja sisältävän taulun tietokannasta, luoda uusia tauluja sekä muokata luotujen taulujen parametrejä eli sarakkeita (column). Jotta tietokanta ei hajoaisi, ensisijainen avain-tyyppisiä (primary key) eikä tracecode-nimisiä sarakkeita voida tauluista poistaa. Tämä mahdollistetaan tekemällä

Remove Selected-nappi harmaaksi silloin, kun sarake jota ei voida poistaa on valittu parametrien listasta. Parametrien lista ilmestyy näkyviin sivustolle silloin, kun jokin Edit Parameters-kohdan alaisista tauluista valitaan. Nappien tekeminen harmaaksi on muutenkin yleisesti tekemäni ominaisuus sivulla, sillä se estää mahdollisesti tietokannan hajottavan toiminnan.

Uudessa tietokannassa kokonaisuuden arkkitehtuuri muuttui siis siten, että tietyn testilaitteiden määrän sijaan sivusto tukee nyt useita testilaitteita, sillä testilaitteita vastaavia koodialtaita on määrä n. Koodialtaiden määrää voidaan muuttaa lisäämällä ja poistamalla tauluja Admin Area-sivun kautta. Myös vanhassa tietokannassa hyödynnetyllä *db-init.sql*-tiedostolla voidaan määrittää koodialtaita. Tämä tiedosto luetaan palvelimen alustamisvaiheessa ja palvelimeen luodaan altaita tiedoston perusteella. Palvelimen uusi arkkitehtuuri ilmenee kuviosta 5.



KUVA 8. Kuvakaappaus uudelta Admin Area-sivulta.



KUVIO 5. Uuden kokonaisuuden arkkitehtuuri.

4 ARVIOINTI JA POHDINTA

Tässä opinnäytetyössä oli tavoitteena rakentaa dynaaminen tietokanta edellisen epädynaamisen tietokannan pohjalta. Tavoitteena oli kehittää tietokannan sisältävä ratkaisu, jota yrityksen tulevaisuuden testilaitteet hyödyntäisivät. Uusi versio tietokannasta kehitettiin virtualisoidussa ympäristössä lähdekoodia muokkaamalla. Väliversiot tietokantasovelluksesta tallennettiin Saskenin GitLab-palvelimelle. Tuloksena syntyi aiempaa tietokantaratkaisua dynaamisempi tuotos, jota voidaan käyttää tulevaisuudessa edellisen ratkaisun tavoin Saskenin palvelintietokoneella. Yrityksen asettama tavoite täyttyi, sillä dynaamisen tietokannan luominen onnistui. Mielestäni opinnäytetyö kokonaisuudessaan oli minulle mielekäs ja olen tyytyväinen lopputulokseen. Opinnäytetyön tekemisen mielekkyys edesauttoi hieman työn saattamista valmiiksi hyvässä tahdissa.

Kehitystyötäni aloittaessa uusien ohjelmistojen vieraus minulle aiheutti ongelmia. Ennen kuin pystyin aloittamaan kehitystyön kunnolla, minun täytyi oppia jotain palvelimen komponenttien toiminnasta. Internetistä löysin hyvin tietoa komponenteista ja niiden toiminnasta. Myös Saskenin toimittama ohjeistus auttoi palvelimen toiminnan käynnistämisessä ja kokonaisvaltaisesti palvelimen toiminnan hallitsemisessa.

Ongelmia tuotti kehitystyön aikana myös vähäinen virtuaalikoneen tallennustila. Tästä johtuen en pystynyt esimerkiksi välillä rakentamaan Docker-kontteja uudestaan lähdekoodin muutosten jälkeen. Tallennustilan vähäisyyden lisäksi kehittämistä vaikeutti ajoittainen virtuaalijärjestelmän jumiutuminen, joka osakseen saattoi johtua vähäisestä tallennustilasta. Vähäisen tallennustilan syy oli lopulta hyvin pitkälti vanhojen Docker-konttien jäännöstiedostot, joita Docker-järjestelmä ei automaattisesti poistanut. Internetistä löysin tietoa siitä, miten Docker-järjestelmä voitiin puhdistaa. Tämän jälkeen konttien uudelleenrakentaminen sujui hieman paremmin. Kehitystyön loppuvaiheessa vähäinen tallennustila ei kuitenkaan enää johtunut Docker-järjestelmän jäännöstiedostoista ja jouduin luomaan uuden virtuaalikoneen, sillä entinen virtuaalitietokone jumiutui niin, että sitä ei voinut enää käyttää. Uuden virtuaalitietokoneen avulla sain vietyä kehitystyötä eteenpäin.

Opinnäytetyön tekemiseen varattu aikataulu aiheutti myös hieman ongelmia. Alunperin tarkoitukseni oli aloittaa työn tekeminen vuoden 2022 lopulla, mutta keskeneräiset kurssini koulussa ja tehtävät yrityksessä hidastivat opinnäytetyön etenemistä. Mielestäni sain kuitenkin tiukasta aikataulusta huolimatta kehitettyä työni kohtuullisessa ajassa.

Tulevaisuutta ajatellen kehittämäni sovellus on hyvä lähtökohta jatkokehitykselle, sillä sovelluksessa on paljon ominaisuuksia, mitä voi vielä parantaa ja ominaisuuksia, mitä ei ole kehitetty loppuun saakka. Esimerkiksi projektien numeroiden (id) käyttö tietokannassa, projektien käyttöliittymänimien ja tietokantanimien (*uiname* ja *dbname*) hyödyntäminen ja koodialtaiden jakaminen useampiin käyttöliittymän tauluihin pääsivulla projektien mukaan voisivat olla seuraavia kehityksen kohteita. Nämä asiat tekisivät sivustosta selkeämmän ja käyttäjäystävällisemmän. Käyttäjäystävällisyys olisi vielä jatkokehitettävissä monelta kannalta, sillä esimerkiksi tällä hetkellä kun tauluja poistetaan tietokannasta, käyttäjälle ei anneta varoituksia ennen poistoa poistonapin klikkaamisen jälkeen. Tauluja ja niiden parametrejä poistettaessa ja lisätessä sivustolla voisi näyttää esimerkiksi Uploads-sivun tiedoston lataamistilanteen mukaisesti tilaviestejä, joista ilmenee, onnistuiko jokin tietty operaatio. Tämä helpottaisi toiminnan ymmärtämistä. Yrityksen tarkoituksena oli myös saada tulevaisuudessa erillinen kirjautuminen Admin Area-sivulle. Tällä hetkellä ennen kuin käyttäjä edes näkee Status-sivua, vaaditaan kirjautumista. Status-, Uploads- ja Downloads-sivujen käyttö on yrityksen mukaan päivittäistä työtä, joten useammat kirjautumiskerrat hidastavat toimintaa. Erillinen kirjautuminen olisi myös turvallisuussyistä oikea ratkaisu.

Opinnäytetyöni kirjallisen osuuden kirjoittaminen sujui melko helposti. Kirjoitusosuuden tekemisen haastavuutta lisäsi hieman se, että yrityksen tietokantaratkaisun koodialtaiden nimet tuli pitää salassa eikä mitään yrityksen projektien tietoja saanut tulla kuviin tai kuvioihin näkyviin. Tästä johtuen muutin kuvien ja kuvioiden sisältöä siten, että luottamukselliset tiedot pysyvät salassa. Saskenin sisäiseen dokumentaatioon pohjautuvassa kuviossa testilaitteiden nimet on muutettu ja käyttöliittymää kuvaavissa kuvissa testattavien laitteiden koodialtaiden nimet ovat piilotettuja. Kirjoitusosuuteen vaadittuja lähteitä löytyi vähän, mutta onnistuin löytämään mielestäni laadukkaita ja luotettavia lähteitä sekä kirjoista että internetistä. Saskenin toimittama dokumentaatio auttoi opinnäytetyön teossa ja sen sisältämät kuviot ja tiedot olivat hyödyllisiä lähes suoraan tätä opinnäytetyötä ajatellen.

5 YHTEENVETO JA JOHTOPÄÄTÖKSET

Tämän opinnäytetyön tuloksena syntyi yrityksen tulevaisuuden tarpeisiin mukautuva dynaaminen tietokanta, joka on hyvin jatkokehitettävissä. Valmista ratkaisua voidaan muuttaa uusia testilaitteita kehitettäessä ja se soveltuu hyvin erilaisiin tilanteisiin erityisesti jatkokehityksen kautta. Uutta ratkaisua voidaan käyttää myös edellisten testilaitteiden kanssa ja sitä voidaan käyttää tulevaisuudessa korvaamaan vanhoilla testilaitteilla käytössä ollut epädynaaminen tietokanta. Tästä kehitetystä työstä on hyötyä yritykselle, sillä nyt uutta tietokantapalvelinta ei tarvitse alustaa vanhalla epädynaamisella tietokannalla, vaan dynaamista ratkaisua voidaan käyttää myös lähtötilanteessa ja sitä voidaan helposti muokata käyttöliittymän kautta yrityksen tarpeita vastaavaksi. Tämä säästää yritykseltä aikaa ja rahaa. Minulle tämän projektin kehittämisestä on myös hyötyä, sillä nyt minulla on enemmän kokemusta sekä taustan (back-end) että käyttöliittymän (front-end) kehittämisestä sekä useista siihen liittyvistä järjestelmistä, kuten esim. Postgres, Node.js ja React. Opinnäytetyöstä saadun kokemuksen avulla minulla on enemmän mahdollisuuksia saada alan työpaikkoja, joissa hyödynnetään edellämainittuja järjestelmiä.

Opinnäytetyöprojekti oli kokonaisuutena mielestäni onnistunut. Mielekäs aihe vaikutti siihen, että sain tehtyä projektin hyvässä tahdissa ja minulle tuli myös yrityksen mielestä hyviä ideoita kehitystyön aikana. Tavoitteena ollut dynaaminen tietokanta, joka oli muokattavissa käyttöliittymän kautta onnistuttiin saavuttamaan ja toisena tavoitteena ollut tietokannan alustamisen mahdollistava lähtötilanteen rakentaminen myös onnistui. Projektissa on myös mahdollisuuksia jatkokehittämiseen, sillä esimerkiksi käyttäjäystävällisyyttä voidaan vielä parantaa. Myös helppokäyttöisyyttä voidaan parantaa, mikäli kehittää käyttöliittymää siten, että koodien lataaminen, lähettäminen ja Status-sivun katseleminen eivät vaadi kirjautumista. Kirjautumattomuus näille sivuille oli yrityksen tulevaisuuden toive, sillä se nopeuttaa päivittäistä työtä. Kuitenkin Admin Area-sivulla kirjautumista vaadittaisiin edelleen turvallisuussyistä.

Tämän opinnäytetyön lopputuloksena syntynyt tietokantaratkaisu on mielestäni hyvä lähtökohta yrityksen tulevaisuuden projekteja ajatellen ja jatkokehittämällä siitä saa vielä toimivamman kokonaisuuden, joka palvelee yritystä mahdollisesti pitkän aikaa. Työn kehittäminen oli kiinnostava projekti, josta on sekä minulle että yritykselle hyötyä.

LÄHTEET

Express-fileupload. GitHub. Saatavissa: https://github.com/richardgirges/express-fileupload. Viitattu 20.4.2023.

Express. GitHub. Saatavissa: https://github.com/expressjs/express. Viitattu 20.4.2023.

FAQ. PgAdmin. Saatavissa: https://www.pgadmin.org/faq/#1. Viitattu 5.4.2023.

Fjordvald, M. & Nedelcu, C., 2018. Nginx HTTP Server. Harness the power of Nginx to make the most of your infrastructure and serve pages faster than ever before. 4. painos. Birmingham, Iso-Britannia: Packt Publishing.

Introduction to Node.js. OpenJS Foundation. Saatavissa: https://nodejs.dev/en/learn/. Viitattu 5.4.2023.

Johanan, J., Khan, T. & Zea, R. 2016. Web Developer's Reference Guide. A one-stop guide to the essentials of web development including popular frameworks such as jQuery, Bootstrap, AngularJS, and Node.js. Milton Keynes, Iso-Britannia: Lightning Source UK Ltd.

Morgan. NPM. Saatavissa: https://www.npmjs.com/package/morgan. Viitattu 26.4.2023.

MUI Core. GitHub. Saatavissa: https://github.com/mui/material-ui. Viitattu 26.4.2023.

Nginx. Nginx.org. Saatavissa: https://nginx.org/en/. Viitattu 5.4.2023.

Node-pg-format. NPM. Saatavissa: https://www.npmjs.com/package/pg-format. Viitattu 26.4.2023.

Node-postgres. Saatavissa: https://node-postgres.com/. Viitattu 26.4.2023.

Pg-copy-streams. NPM. Saatavissa: https://www.npmjs.com/package/pg-copy-streams. Viitattu 26.4.2023.

React. Meta Open Source. Saatavissa: https://react.dev/. Viitattu 24.4.2023.

Redmond, E. & Wilson, J. R. 2012. Seven Databases in Seven Weeks. A Guide to Modern Databases and the NoSQL Movement. 2. painos. USA.

Validator.js. NPM. Saatavissa: https://www.npmjs.com/package/validator. Viitattu 26.4.2023.

What is a REST API? IBM. Saatavissa: https://www.ibm.com/topics/rest-apis. Viitattu 20.4.2023

What is Docker? IBM. Saatavissa: https://www.ibm.com/topics/docker. Viitattu 5.4.2023.

What is PostgreSQL? The PostgreSQL Global Development Group. Saatavissa: https://www.postgresql.org/about/. Viitattu 5.4.2023.