jamk

RGCE IoC Palvelu

Samuli Virtapohja

Opinnäytetyö, AMK
Huhtikuu 2021
Tekniikan ala
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Samuli Virtapohja

RGCE IoC Palvelu

Jyväskylä: Jyväskylän ammattikorkeakoulu. Huhtikuu 2021, 42 sivua

Tekniikan ala, tieto- ja viestintätekniikan koulutusohjelma, Opinnäytetyö AMK

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Toimeksiantaja Jyvsectec toimii kyberturvallisuuden alalla tarjoten koulutusta, konsultointia, tutkimusta sekä kehitystä. Soveltavassa tutkimuksessa toteutettiin VirusTotalin kaltainen palvelu RGCE:hen antamaan käyttäjille toiminnallisia ohjeita haitallisten tiedostojen löytyessä.

Työssä tuotiin esille teknologioita ja käyttötapauksia, joilla palvelu on mahdollista toteuttaa. Teoriaosuudessa käytiin läpi teknologioita ja tekniikoita palvelun vaatimusten kannalta. Käytetyt teknologiat ja käyttötapaukset purettiin palasiksi tekstimuodossa kuvien kera.

Työn tuloksena saatiin aikaiseksi Viruscomplete web-sovelluksena, joka on mahdollista käyttöönottaa RGCE:ssä.

Pohdinnassa käydään läpi palvelun jatkokehitysmahdollisuuksia.

Avainsanat (asiasanat)

www-ohjelmointi, tietokantaohjelmat, verkkotietopalvelu, ohjelmistokehitys, soveltava tutkimus

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Virtapohja Samuli

RGCE IoC Palvelu

Jyväskylä: JAMK University of Applied Sciences, April 2021, 42 pages.

Engineering and technology. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

Client company JYVSECTEC, which works in the field of cybersecurity offering education, consulting, research and development, had the need to develop a service similar to VirusTotal into RGCE. The service gives instructions to the user upon finding malicious files.

Thesis brings up technologies and use-cases which allow the possibility of developing the service. In theory, thesis will go through technologies and techniques based on requirements of the service. Used technologies and use-cases have been disassembled in text and graphic formats.

The results of the thesis is a web-application: Viruscomplete, which can be deployed into RGCE.

The deliberation contains thoughts on further development of the service.

Keywords/tags (subjects)

web programming, database programs, online information services, software development, applied research

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

No table of figures entries found.

Lyhenteet

Boolean	Ohjelmointikielessä käytetty tyyppi tosi ja epätosi määrityksille.
CRUD	Sovelluksessa jonkin tietyn resurssin luominen (Create), hakeminen (Read), päivittäminen (Update) tai poistaminen (Delete).
DOM	Document Object Model. Ohjelmointirajapinta html dokumenttien muokkaukseen.
DSL	Domain Specific Language. Tietylle sovellusalueelle erikoistunut kieli esim. typescript, jsx, html-pohjainen
DTO	Data Transfer Object. Nestjs:n käyttämä abstraktio tietotyypistä.
Entity	Luokka joka kertoo ORMille taulurakenteen.
Hash	Tiiviste
Hash Collision	Kahden eri alkuarvon tuottama sama tiiviste.
HTML	Hypertext Markup Language. Pääasiallinen verkkosivujen kirjoituskieli.
HTTP	Hypertext Transfer Protocol. Sovellustason protokolla tiedostojen siirtoon. (Web technology for developers)
IETF	Internet Engineering Task Force. Internetin standardeista vastaava organisaatio.
IoC	Inversion of Control on käsite abstraktiosta ja sen tarkoituksena kyberharjoitusten näkökulmasta IoC on kuvaus jonkun tekemästä asiasta.

JWT	Json Web Token
ORM	Object-Relational-Mapping
MD5	Message-Digest-5. Tiivistefunktio, jota nykypäivänä voidaan käyttää tiedon eheyden tarkastamiseen.
MVC	Model View Controller. Arkkitehtuuri malli, joka erottelee käyttöliittymän muusta sovelluksesta.
NoSQL	Not only SQL. NoSQL-tietokantoihin luetaan yleensä kaikki muut tietokannat paitsi relaatiotietokannat.
ORM	Object-Relational-Mapping
REST	Representational State Transfer. Palvelu, joka jakaa resursseja pyynnöstä.
Repository	ORMin luoma tietolähde.
RFC	Request for Comments. IETF:n julkaisemat standardit.
RGCE	Realistic Global Cyber Environment. Jyvsectecin luoma verkkoympäristö.
SQL	Structured Query Language. Standardoitu kyselykieli tietokantoja varten.

1 Johdanto

Insinöörityön aiheena oli toteuttaa soveltava tutkimus RGCE IoC palvelusta Jyvsectecille. Kyberturvallisuus tulee olemaan tulevaisuudessa, sekä nykyhetkessä, äärimmäisen tärkeässä asemassa yksittäisille henkilöille ja eri organisaatioille. Työkalut tietoturvallisuuden parantamiseksi tapahtuu ohjelmistokehityksen kautta.

Jyvsectec on yritys, joka toimii kyberturvallisuuden sektorilla tarjoten koulutusta, konsultointia, tutkimusta ja testausta. (About us). RGCE on ympäristö, jonka tarkoituksena on emuloida verkkoa. Kyberturvallisuusharjoitukset järjestetään RGCE-ympäristössä ja harjoitusten tarkoituksena on kartoittaa yritysten tai organisaatioiden valmiutta kyberuhkien tai -hyökkäysten tapahtuessa. (Cyber Exercises Overview).

RGCE IoC Palvelu tähtää automatisoimaan kyberharjoitusten aikana tapahtuvan aikaa vievän manuaalisen tiedoston tarkastamisen ja helpottaa siten työntekijöiden taakkaa kyberharjoitusten aikana. Lopullisessa palvelussa on mahdollista ladata kirjautuneena käyttäjänä tiedostoja, ja liittää kyseisen tiedoston tiiviisteeseen raportti, joka palautetaan käyttäjille käyttötapausten perusteella.

Työn teoriaosuudessa käsitellään yleisellä tasolla käytettyjä teknologioita, joilla palvelu on mahdollista toteuttaa. Teoriaosuus on rajattu vastaamaan palveluun käytettäviä teknologioita.

Teoriaosuuden jälkeen käydään läpi RGCE IoC palvelun osuuksia ja sitä kuinka osuuksia voidaan milläkin teknologialla tehdä. Palvelu muodostuu käyttötapauksen ympärille ja palvelun osuudet tukevat käyttötapauksen toteutumista.

Tuloksissa esitellään lopullista palvelun toteutusta kuvakaappauksin käyttäjän näkökulmasta. Lukijalle, komponenteista ja näkymistä, on lisätty kuvia hahmottamaan lopullista palvelua.

Viimeiseksi pohditaan palvelun mahdollista jatkokehitystä ja analysoidaan työtä.

2 Käytetyt teknologiat ja tekniikat

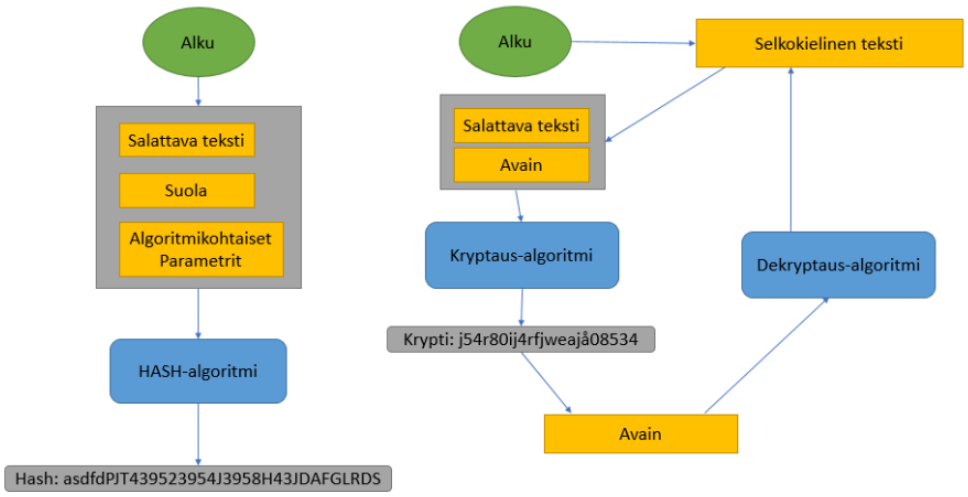
Nykypäivänä sovelluskehityksessä ei tarvitse, eikä edes kannata, kehittää kaikkea itse. Sanonta ”Pyörää ei tarvitse keksiä uudestaan” resonoi erityisesti RGCE IoC palvelun toteutuksessa. Palvelun toteutus nojaa järjestelmän laajuuden ja tekijämäärän takia hyvin moneen ohjelmistokehykseen.

Toimeksiantajalta ei tullut hirveästi vaatimuksia toteutuksen teknologioihin. Ainoa isompi vaatimus oli ulkoisen järjestelmän mahdollisuus muokata järjestelmää. Vaatimuksen palvelinpuolen toteutus oli järkevää tehdä REST rajapintana, jolloin muutosten teko järjestelmään tapahtuu HTTP pyynnöillä.

Teoriaosuuden tiedonhakuun käytettiin teknologioiden tarjoajien sekä kehittäjien omia sivustoja, kyberturvallisuuden alan julkaisuja ja IETF-organisaation julkaisemia RFC-dokumentteja. Suurin osa teoriaosuuden materiaalista on kansainvälistä.

2.1 Tiivisteistä yleisesti

Tietotekniikassa tiivisteiden muodostamiseen käytettävät algoritmit jakautuvat kahteen perheeseen: kaksisuuntaiseen ja yksisuuntaiseen salaukseen. Kaksisuuntaisessa salauksessa alkuperäinen viesti on palautettavissa avaimella, kun taas yksisuuntaisessa salauksessa tiivistettä on melkein mahdoton palauttaa alkuperäiseen muotoonsa. (Niemelä, Jarno. s. 3)

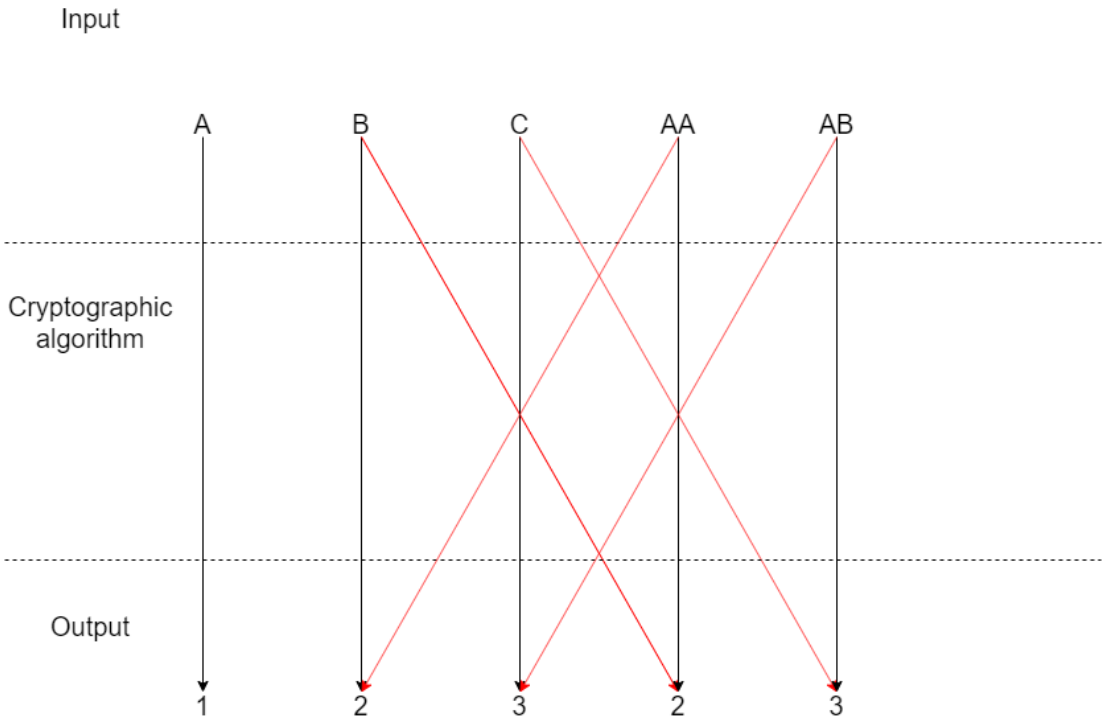


Kuvio 1 Hash-algoritmin ja kryptauksen eroavaisuus. (Pakkanen, A. s. 4)

Tiivisteiden muodostuksessa tarvitaan alkuarvo, josta tiiviste muodostetaan. Tiivistefunktio muodostaa alkuarvosta tiivisteen annettujen parametrien perusteella. Parametreinä tiivistefunktioon voi olla esimerkiksi suola, mikä lisää järjestelmän turvallisuutta tai käytettävän algoritmin kierrosmäärä. Kuvio 1 havainnollistaa tiivistefunktioiden tapahtumia kuvallisesti.

2.1.1 Tiivisteiden törmäykset

Tiivisteiden törmäys tapahtuu algoritmin muodostaessa kahdesta eri alkuarvosta saman sisällön. Kuvio 2 sisältää graafisen esimerkin tapahtumasta.



Kuvio 2 Karkea esimerkki tiivisteiden törmäämisestä

Riippuen sekä tiivisteen osuudesta palvelussa, että palvelun kriittisyydestä tiivisteiden törmäys järjestelmässä voi mahdollistaa mittavat vahingot palvelun omistajalle tai palvelun käyttäjille.

2.2 Tiivisteiden käyttö palvelussa

Tiedostojen tiivisteen muodostus tapahtuu kryptografisella funktiolla, mikä muuntaa funktiolle syötetyn datan merkkijonoksi. Merkkijonojen on tarkoitus olla yhdensuuntaisia eli sama tiedosto tuottaa saman tiivisteen ja yhden bitin muutos tuottaa eri tiivisteen. (Niemelä, Jarno. s. 4).

Tiivistefunktioissa on kolme pääasiaa, mitkä vaikuttavat tarvittavan funktion valintaan. Nopeus, funktio joutuu lukemaan koko tiedoston luodakseen tiivisteen. Luodun tiivisteen pituus vaikuttaa tietokannan viemään tilaan. Turvallisuus, tiivisteiden mahdollisuus törmätä tulisi olla mahdollisimman pieni. (Ramirez, G.).

RGCE IoC palvelun tiivisteiden koko ei tule olemaan relevantti tekijä tiedostoja tarkistaessa. Sen sijaan tiivisteen muodostuksen nopeus sekä tiivisteiden törmäyksien määrä ovat tärkeämpiä tekijöitä. Yksi vaikuttava asia myös tiivisteissä on tiivistefunktion yleisyys.

MD5 suunniteltiin alunperin korvaamaan MD4-algoritmia. Algoritmi on tarkoitettu digitaalisille allekirjoitus sovelluksille. MD5 ottaa syöttönä viestin, jonka pituus on mielivaltainen, ja tuottaa 128-bittisen tiivisteen. (Rivest, R.) MD5-algoritmia ei pitäisi käyttää tapauksissa, missä käsitellään arkaluontoista tietoa. Haavoittuvuudet MD5-algoritmissa on todistettu kerta toisensa jälkeen erilaisilla hyökkäyksillä sekä tiivisteiden törmäyksillä. RGCE IoC palvelun kannalta MD5 algoritmia käytetään tiedostojen tiivisteiden muodostukseen, sekä raporttien hakuun järjestelmästä.

Bcrypt luotiin 1999 ja sen tarkoituksena on salasanojen salaus, joka perustuu Blowfish-lohkosalaukseen. (Abdullah, M. s. 11) Bcryptiä tullaan käyttämään projektissa salasanojen tiivisteiden muodostamiseen.

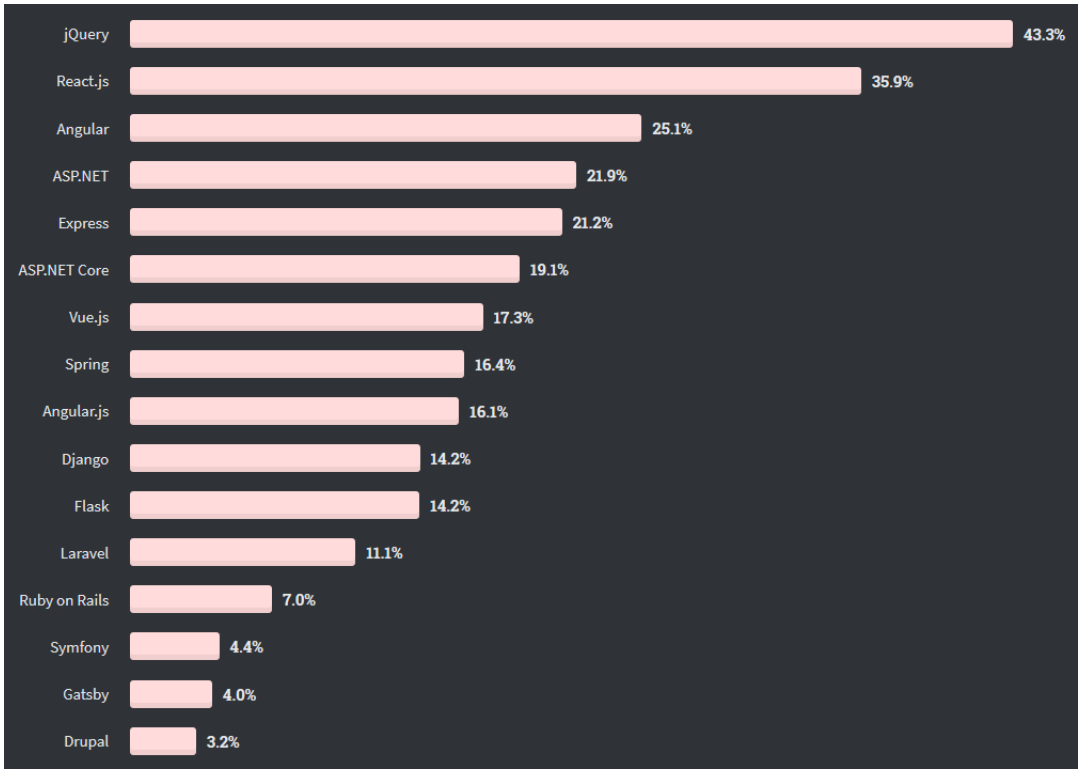
2.3 Käyttöliittymä

Moderneissa web-sovelluksissa käytetään hyvin useasti jotain ohjelmistokehystä ajamaan web-sovelluksen raameja, mutta komponentit sovelluksien käyttötapauksiin luodaan kehitysprosessin aikana. Ohjelmistokehykset web-sovelluksissa toimivat vahvasti oman DSL:n alla ja poikkeaminen ohjelmistokehyksen DSL:stä saattaa jättää kehittäjän ohjelmistokehyksen hyötyjen ulkopuolelle.

Käyttöliittymien toiminta web-sovelluksissa perustuu DOM:in muokkaukseen ja muutoksien renderöimiseen, kun tiedon tila muuttuu. (Introduction to client-side frameworks).

2.3.1 Kehittäjäkysely web-ohjelmistokehyksistä

Stackoverflown kehittäjille tarkoitetussa vuosittaisessa 2020 kyselyyn osallistui melkein 65000 kehittäjää. Kehittäjistä 42279 vastasi käyttävänsä jotain web-ohjelmistokehystä. (2020 Developer Survey).



Kuvio 3 Stackoverflowin kehittäjäkysely 2020, Käytetyimmät web-ohjelmistokehykset

Jollain Stackoverflown kyselyn tuloksiin päätyneillä ohjelmistokehyksillä, kuvio 3, on järkevää toteuttaa RGCE IoC palvelun käyttöliittymä. Syynä on ohjelmistokehyksien yleisyys alalla.

2.3.2 Tyylikirjastot

Tyylikirjastot sisälllyttävät itseensä erilaisia resursseja tarjoten kehittäjälle niin kutsutun yhden totuuden lähteen graafisesta näkökulmasta. (Hacq, A.)

Material on Googlen kehittämä tyylikirjasto, jonka tarkoituksena on auttaa eri ryhmiä luomaan korkeatasoisia digitaalisia kokemuksia. (Material System Introduction.)

2.4 REST

RESTin pääperiaatteina toimivat asiakas-palvelin malli sekä tilaton yhteydenpito käyttöliittymän ja palvelimen välillä. REST-mallissa palvelin jakelee haettua resurssia pyynnön sisältämän tiedon perusteella. (Fielding, R. s. 76-97).

Rest rajapinta jakelee resursseja HTTP-metodien perusteella. Pääasiallisesti käytössä olevat metodit ovat: GET palauttaa tämänhetkisen tilan tavoitellusta resurssista, POST suorittaa resurssikohtaisen prosessin pyynnön sisällön perusteella, PUT ylikirjottaa kaikki tämänhetkisen tilan esitykset pyynnön sisällön perusteella, DELETE poistaa pyydetyn resurssin (Fielding, R. & Reschke, J.).

2.5 Tietokanta

Tietokannat jakautuvat pääasiallisesti NoSQL- ja relaatiotietokantoihin.

Relaatiotietokannat saavat nimensä relaatioista, tarkoittaen tietokannan taulujen välisiä suhteita. Relaatiotietokannan vahvuutena sekä myös heikkoutena on sen painottuminen määritettyyn kaavaan. Taulujen tyypit ja kolumnit joudutaan määrittämään etukäteen, jotta niitä pystytään käyttämään. (What is NoSQL?).

NoSQL tietokannoissa pääasiallisena erona on se, ettei tietokantaa tarvitse päivittää, jos käsiteltävän tiedon rakenne muuttuu. Tämä mahdollistaa sovelluksen nopean kehityksen, mutta aiheuttaa sen, ettei tieto ole välttämättä yhteneväistä. (What is NoSQL?).

2.5.1 ORM

ORMien pääasiallisena tarkoituksena on muuntaa tietoa kahden eri järjestelmän järjestelmän välillä. RGCE IoC palvelun kontekstissa ORM tulee tietokannan ja palvelimen välille.

2.6 Kontitus (Containerization)

Kontitus on tekniikka, jolla sovellus pakataan kaikkine riippuvuuksineen yhteen mahdollistaen sovelluksen ajamisen ympäristöstä riippumattomana. (What is a container?). Kontittaminen siis mahdollistaa järjestelmän kehitysympäristön pystyttämisen vastaavanlaiseksi kuin RGCE ympäristössä.

Sovelluksesta voidaan luoda kuva, joka sisältää tarvittavan konfiguraation sovelluksen ajamiseksi. Kuva voidaan luoda Dockerfile-tiedostolla ja kyseisestä kuvasta ajetaan instanssi kun niin halutaan.

3 Sovelluksen osat

3.1 Ohjelmiston arkkitehtuuri

RGCE IoC Palvelu jakautuu kolmeen osuuteen käyttöliittymään, REST rajapinta palvelimeen sekä tietokantaan. Käyttöliittymään ja palvelimeen tuotuja kirjastoja ja riippuvuuksia hallitaan NPM:n avulla. Liite 1 havainnollistaa kuvallisesti palvelun arkkitehtuuria.

Käyttöliittymän julkisella puolella, jossa kuka tahansa pystyy lähettämään tiedoston. Tiedoston tiiviste tarkistetaan, ja rajapinnasta palautetaan käyttötapausten perusteella raportti käyttäjälle. Raportteja pystyy myös etsimään tiedoston tiivisteen perusteella.

Kirjautuneella käyttäjällä on ohjauspaneeli, millä on tarkoitus hallita raportteja. Käyttötapauksia raportin luontiin annettiin kolme. Raportti on julkinen, raportilla on viive julkaisussa sekä raportti menee julkiseksi tietyn ajan tullessa täyteen.

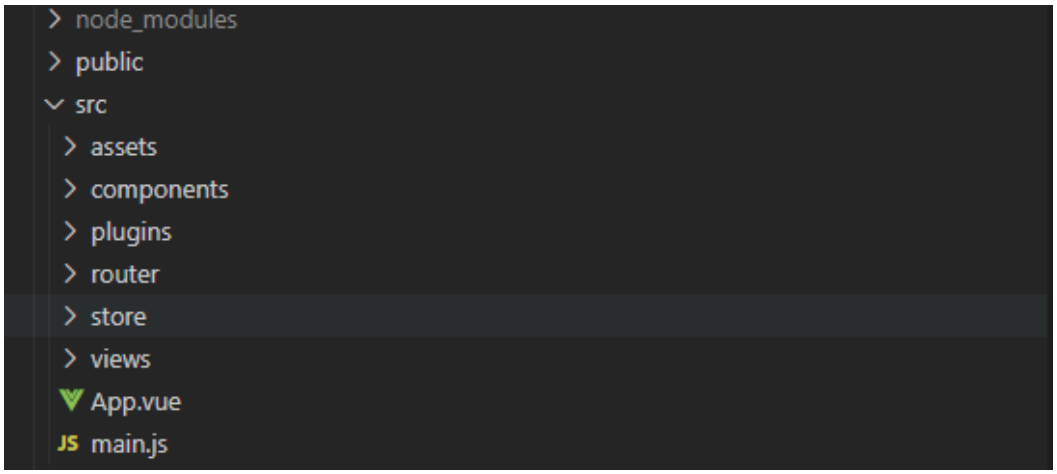
Arkkitehtuurin puolella keskustelimme toimeksiantajan kanssa mikropalveluiden käytöstä palvelussa, mutta päädyimme kuitenkin jakautuvaan monoliittiseen ratkaisuun ylläpitoon aiheutuvan rasituksen vuoksi. Jakautuvassa monoliittisessa ratkaisussa palvelun pääasialliset komponentit jaetaan omiin kontteihinsa.

Mikropalveluissa kontitettuja palveluita tulisi huomattavasti lisää, jonka seurauksesta niin kutsuttu tekninen velka kasvaisi ajan kuluessa liian suureksi. Esimerkiksi tiedostojen tallentamiselle olisi

ollut mahdollista toteuttaa erillinen palvelu, mihin tiedostot tallennettaisiin. Tiedostojen hakemiseen ja luomiseen erillisessä palvelussa tulisi tehdä omat sääntönsä ja palvelun terveyttä sekä vaatimaa tilaa tulisi tarkkailla. Toteutus erilliselle tiedostojen säilytykselle olisi helppoa, mutta palvelun ylläpito vaatisi ylimääräistä työtä.

3.2 Palvelun käyttöliittymä

Vue on progressiivinen ohjelmistokehys web-sovelluksien ja sen ydin keskittyy sovelluksen näkymän luomiseen. (Vue.js Essentials.)



Kuvio 4 Vue sovelluksen kansiorakenne

3.2.1 Vue sovelluksen alkupiste

Vuen lähdekoodin suorittaminen lähtee main.js-tiedostosta. Main sisällyttää itseensä sovelluksen käyttämät resurssit ja määrittää kohdan mihin sovellus renderöidään. Kuvio 5 näyttää esimerkkimallin main.js-tiedostosta, kyseiseen esimerkkiin on tuotu Vuex-tilanhallinta, reititys sekä Vuetify mukaan.

```
import Vue from 'vue'
import App from './App.vue'
import router from './router/index.js'
import store from './store/index.js'
import vuetify from './plugins/vuetify.js'

Vue.config.productionTip = false

new Vue({
    router,
    store,
    vuetify,
    render: (h) => h(App),
}).$mount('#app')

```

Kuvio 5 Vue main.js

3.2.2 Vue-tiedostot

Vue tiedostoissa koodin rakenne jakautuu kolmeen osuuteen: templateen, scriptiin ja styleen. Template sisältää verkkosivulle renderöitävät asiat. Script-osuus kertoo näkymän tai komponentin tiedon. Style-osuus määrittää komponentin tyylitykset. Kuvio 6 havainnollistaa vue-tiedostoa graafisesti.

```
<template>
    <div>
        Rendered text
    </div>
</template>

<script>
export default {
    data() {
        return {
            // component variables
        }
    },
    methods: {
        // component functions
    },
    components: {
        // other components used in template
    }
}
</script>

<style>
/*css styles*/
</style>
```

Kuvio 6 Vue-tiedoston rakenne

3.2.3 Reititys Vuessa

Reititys jakaa käyttöliittymän eri näkymiin. Reititykseen määritetään mihin polkuun haluttu komponentti renderöidään. Itse renderöinti määritellään vue-tiedostojen renderöitävässä osuudessa ’router-view’-komponenteilla. Kuvio 7 visualisoi esimerkkinä vuen käyttämää reititystä.

```
import Vue from 'vue'
import VueRouter from 'vue-router'

Vue.use(VueRouter)

const routes = [
    {
        path: '/',
        name: 'home',
        component: () => import('@/views/Home.vue'),
    },
    {
        path: '/dashboard/',
        name: 'dashboard',
        component: () => import('@/views/Dashboard.vue'),
    },
]

const router = new VueRouter({
    mode: 'history',
    routes,
})

export default router
```

Kuvio 7 Vuen reititys

3.2.4 Vuex

Vuex on vue sovellusten tilan hallintakirjasto. Tilanhallinta tuodaan yleensä sovellukseen siinä vaiheessa, kun komponentteja alkaa olemaan paljon. Web-sovellukset ovat lähtökohtaisesti tilattomia, mikä tarkoittaa sitä etteivät tiedot siirry sovelluksen sisällä. Tilan säilyminen sovelluksessa parantaa käyttäjäkokemusta sekä sovelluksen toimintaa.

Vuex-moduulit sisältävät neljä asiaa: staten eli tiedon sovelluksen tilasta, mutaatiot eli funktiot joilla tilaa muutetaan, getterit eli haku-funktiot jotka palauttavat tilan sekä actions-funktiot eli halutut toiminnot. Kuvio 8 näyttää toteutuksen vuex-moduulista.

```
const state = {
    // tila
}

const mutations = {
    // tilan muuttamisen funktiot
}

const getters = {
    // tilan palauttamisen funktiot
}

const actions = {
    // funktiot
}

export default {
    state,
    mutations,
    getters,
    actions,
}
```

Kuvio 8 Vuex moduulin sisältö

Vuex-moduulit yhdistetään vueen omassa tiedostossaan koodin lukemisen helpottamiseksi, kuvio 9.

```
import Vue from 'vue'
import Vuex from 'vuex'
import createPersistedState from 'vuex-persistedstate'
import exercise from './modules/exercise.js'
import auth from './modules/auth.js'

Vue.use(Vuex)

export default new Vuex.Store({
    modules: {
        auth,
        exercise,
    },
    plugins: [
        createPersistedState({
            paths: ['auth'], // sets auth module to localStorage
        }),
    ],
})
```

Kuvio 9 Vuex moduulien tuominen Vue sovellukseen

Tilan käyttö vue-tiedostoissa tapahtuu tuomalla mapGetters- ja mapActions-funktiot tiedoston skripti osuudessa. MapGetters ja mapActions ottavat parametrinä sisään jonon funktioiden nimistä joita tiedostossa halutaan suorittaa. Kuvio 10 havainnollistaa komponentille tuotuja funktioita.

```
233 <script>
234 import { mapGetters, mapActions } from 'vuex'
235
236 export default {
237     name: 'ReportList',
238     data() {
239         return {
240         }
241     },
242     computed: {
243         ...mapGetters(['stateGetterFunction']),
244     },
245     methods: {
246         ...mapActions(['stateActionFunction']),
247     }
```

Kuvio 10 Vuex tilan funktiot vue-tiedostossa.

3.2.5 Vuetify

Sovelluksen käyttöliittymän tyylikirjastona toimi Vuetify, joka toteuttaa Material Designin määrittelyn. Vuetifyn avulla sovellukseen saadaan tuotua komponentteja valmiilla tyylityksellä, mikä nopeuttaa sovelluksen kehitysprosessia. Kuviossa 11 havainnollistetaan Vuetifyn tuomista projektiin.

```
import '@mdi/font/css/materialdesignicons.css'
import Vue from 'vue'
import Vuetify from 'vuetify/lib'

Vue.use(Vuetify)

export default new Vuetify({
    icons: {
        iconfont: 'mdi',
    },
    theme: {
        dark: true,
    },
})
```

Kuvio 11 Vuetifyn tuominen projektiin

3.3 Nestjs

Nestjs on typescript ohjelmistokehys palvelinpuolen NodeJS applikaatioille. Pääasiallisesti Nestjs:n avulla pyritään tuottamaan niin kutsuttuja ”koodi ensin”-ratkaisuja. (Nestjs Introduction).

Nestjs abstraktoi MVC-arkkitehtuuriin tarvittavia osuuksia, tuoden kirjastoja tarjoten ne oman rajapintansa kautta kehittäjän käytettäväksi.

3.3.1 Nestjs sovelluksen alkupiste

Nestjs sovelluksen käynnistyessä main.js tiedosto ajetaan, jossa sovellukselle kerrotaan suoritettavat asiat sekä sisällytetään sovelluksen konfiguraatio. Kuviossa 12 esitetään graafisesti esimerkkirakennetta.

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { SwaggerModule, DocumentBuilder } from '@nestjs/swagger'

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const options = new DocumentBuilder()
    .setTitle('RGCE IOC Palvelu')
    .setDescription('API description')
    .setVersion('1.0')
    .build()

  const document = SwaggerModule.createDocument(app, options)
  SwaggerModule.setup('api', app, document)

  app.enableCors()

  await app.listen(process.env.PORT);
}
bootstrap();
```

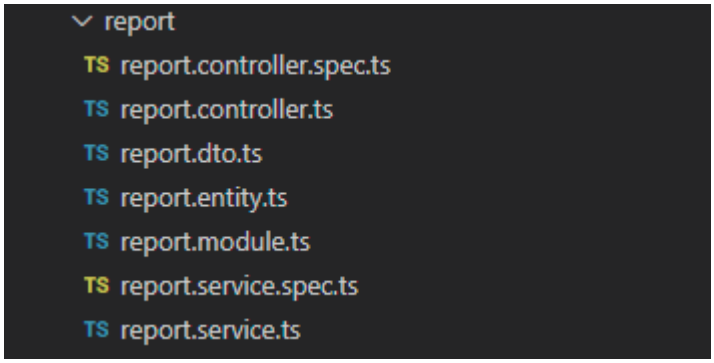
Kuvio 12 Nestjs main.js-tiedosto

Ohjelmistokehyksen rajapinnasta tuodut funktiot kertovat sovellukselle käynnistyksen jälkeen kuunneltavat päätepisteet ja päätepisteiden suoritettavat funktiot.

```
[Nest] 6372   - 2021-04-11 21:58:28   [RoutesResolver] ExerciseController {/exercise}: +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/exercise, GET} route +2ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/exercise/:id, GET} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/exercise, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/exercise/:id, DELETE} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/exercise, PUT} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RoutesResolver] ReportController {/report}: +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/report/:id, GET} route +3ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/report, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/report, PUT} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/report/:id, DELETE} route +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RoutesResolver] UserController {/user}: +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/user, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/user, PUT} route +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/user/logout, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/user/refresh, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RoutesResolver] FileController {/file}: +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/file/:hash, GET} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/file, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RoutesResolver] IocController {/ioc}: +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/ioc, POST} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/ioc, GET} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/ioc/:id, GET} route +0ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/ioc/:id, DELETE} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [RouterExplorer] Mapped {/ioc, PUT} route +1ms
[Nest] 6372   - 2021-04-11 21:58:28   [NestApplication] Nest application successfully started +22ms
```

Kuvio 13 Nestjs käynnistyminen

3.3.2 Nestjs moduulit



Kuvio 14 Esimerkki moduulin sisältävistä tiedostoista

Nestjs sitoo sovellukseensa käytetyt luodut moduulit. Moduulit kertovat sovellukselle jokaisen moduulin tarvitsemat resurssit sekä kontrollerit. Kuvio 15 näyttää esimerkkirakennetta Nestjs moduulin luonnista.

```
@Module({
  imports: [
    TypeOrmModule.forFeature(
      [
        Report,
        ExerciseHasReport,
        Exercise,
        File,
        FileHasHash,
        Ioc,
        User,
        BlackList
      ]
    )
  ],
  exports: [TypeOrmModule],
  controllers: [ReportController],
  providers: [ReportService, FileService, UserService]
})
export class ReportModule {}
```

Kuvio 15 Esimerkki Nestjs moduulista

3.3.3 Nestjs kontrollerit

Kontrollereiden tarkoituksena sovelluksessa on kertoa, mitä pyyntöjä kyseinen kontrolleri vastaanottaa ja käsitellä pyynnöt halutulla tavalla. Kontrollereiden käsittelemät tietotyypit kuvataan pääasiallisesti ’dto’-nimikkeillä.

Kuviossa 16 ReportController luokan konstruktori tuo injektoitavan ReportService-luokan kontrollerin sisälle. ReportServiceen luotuja funktioita on nyt mahdollista kutsua kontrollerin sisältä. @Controller-dekoraattori määrittää kontrollerin päätepisteen, mitä sovellus kuuntelee. @Get-, @Post- ja @Put-dekoraattorit kertovat päätepisteen vastaanottavat HTTP-metodit.

```
@ApiTags('report')
@Controller('report')
export class ReportController {
    constructor(
        private readonly reportService: ReportService
    ){}

    @Get(':id')
    @UseGuards(AuthGuard)
    async getAllReports(@Param('id') exerciseId) {
        return await this.reportService.getReports(exerciseId)
    }

    @Post()
    @UseGuards(AuthGuard)
    @UseInterceptors(FileInterceptor('file', {
        storage: diskStorage({
            destination: './uploads/',
            filename: (request, file, callback) => {
                crypto.pseudoRandomBytes(16, (err, raw) => {
                    if (err) return callback(err)

                    callback(null, raw.toString('hex') + file.originalname)
                })
            }
        })
    }))
    async createReport(@Body() report: ReportDto, @UploadedFile('file') file) {
        return await this.reportService.createReport(report, file)
    }

    @Put()
    @UseGuards(AuthGuard)
    async updateReport(@Body() report: ReportDto) {
        return await this.reportService.updateReport(report)
    }
```

Kuvio 16 Esimerkki kontrollerista

Suoritettava logiikka itsessään sisällytetään niin kutsuttuihin ’service’-tiedostoihin. Jos suoritettavaa koodia tuodaan muualta sovelluksen alueelta, tulee moduulissa kertoa käytetyt luokat, jotta sitä pystytään suorittamaan.

3.3.4 Rajapintakuvaus

REST rajapintakuvaus toteutettiin Swaggerin avulla. Swagger kuvaa rajapinnan pyynnöt ja vaatimukset sekä mahdollistaa rajapinnan dokumentoinnin ohjelmoinnin yhteydessä.

Rajapintakuvaus kertoo kehittäjälle tiedot tyypeistä ja tyyppien ominaisuuksista. Swagger saa kiinni tyyppien ominaisuuksista ’@ApiProperty’-funktion avulla.

```
6   export class ReportDto {
7       @ApiProperty({
8           required: false
9       })
10      id: string
11
12      @ApiProperty({
13          required: false
14      })
15      public: boolean
16
17      @ApiProperty({
18          required: false
19      })
20      releaseTime: number
21
```

Kuvio 17 Esimerkki DTOsta

Palvelimen käynnistyessä tyyppien kentät tuodaan esille, jotta http-pyyntöjä tekevä taho pystyy sisällyttämään oikean tiedon pyyntöön. Kuvio 18 havainnollistaa Swaggerin luoman rajapintakuvauksen palvelusta.

Kuvio 18 Rajapinnan kuvaus Swaggerilla

3.3.5 Tunnistautuminen järjestelmässä

Jwt on avoin standardi allekirjoitetulle tiedolle ja jwt:n pääasialliset käyttötarkoitukset liittyvät tunnistautumiseen sekä tiedon vaihtoon. (Introduction to JSON Web Tokens)

Käyttäjä tunnistautuu järjestelmään tunnus ja salasana yhdistelmällä. Salasanasta muodostetaan tarkistussumma Bcryptin avulla. Jos tunnistautuminen onnistuu, käyttäjälle palautetaan jwt. Liitteessä 2 käydään tunnistautumista läpi sekvenssikaaviona.

3.3.6 Nestjs Guards

Järjestelmän backendiin luodaan middleware eli väliohjelmisto, mikä tarkistaa backendiin tulevien kyselyiden eheyden. Kyselyihin liitetään sisäänkirjautumisen yhteydestä saatu jwt, joka tarkastetaan halutulla tavalla. Jwtn tarkastus määrittää kyselyn pääsemisen läpi.

```
14      async canActivate(context: ExecutionContext): Promise<boolean> {
15        const request = context.switchToHttp().getRequest()
16
17        if (!request.headers.authorization) {
18          return false
19        }
20
21        await this.validateToken(request.headers.authorization)
22
23        return true;
24      }
```

Kuvio 19 Väliohjelmiston toteuttama CanActivate

Väliohjelmiston tulee toteuttaa CanActivate-funktio, mikä ajetaan pyynnön tullessa rajapintaan. CanActivate palauttaa boolean-arvon minkä perusteella sallitaan kontrollerissa suoritettava funktio. Kuvio 19 näyttää esimerkin väliohjelmiston toteuttamasta validoinnista.

```
10  @ApiTags('report')
11  @Controller('report')
12  export class ReportController {
13      constructor(
14          private readonly reportService: ReportService
15      ){}
16
17      @Get(':id')
18      @UseGuards(AuthGuard)
19      async getAllReports(@Param('id') exerciseId) {
20          return await this.reportService.getReports(exerciseId)
21      }
```

Kuvio 20 Kontrollerissa toteutettava väliohjelmisto

Ohjelmistokehyksestä tuodaan kontrollerille ’@UseGuards’-funktio, jolle välitetään parametrinä luotu luokka. Kuvio 20 näyttää esimerkkinä väliohjelmiston tuomisen kontrollerille.

3.3.7 Raportin hakeminen järjestelmästä

Tiedon hakeminen järjestelmästä tapahtuu http-pyynnöllä, johon liitetään tiedosto tai annetaan parametrinä haettava MD5-tiiviste. Jos pyyntö sisältää tiedoston, muodostetaan tiedostosta MD5-tiiviste. Liiteet 3 ja 4 havainnollistavat tapahtumaa sekvenssikaaviona sekä prosessikaaviona.

3.4 Tietokanta

Tietokantarakenteen pääkonsepteina olivat raportti, raporttiin liitetty tiedosto ja sen tiiviste sekä raporttiin liitetyt iocit. Liite 5 sisältää tietokantarakenteen. Raporttien hakuoptimointia lisäämään tuotiin tauluihin harjoitukset ja sisäänkirjautumisen seurauksesta käyttäjätaulu oli tarpeellinen.

3.4.1 TypeORM

Projektin toteutuksen ollessa koodipainotteinen, otettiin mukaan myös ORM eli object-relational mapping. (Nestjs Techniques)

NestJS integroi TypeORMin omaan pakettiinsa, joten se oli luonnollinen tapa sisällyttää se projektiin. TypeORMin avulla tietokanta mallinnetaan osaksi ohjelmistoa. Tietokannan tauluista luodaan luokka, jolle annetaan kirjaston avulla määritelmät tietokannan tauluista.

```
6   @Entity('Report')
7   export class Report {
8       @PrimaryGeneratedColumn('uuid')
9       id: string
10
11      @Column({
12          nullable: true,
13      })
14      public: boolean
15
16      @Column({
17          nullable: true,
18          type: 'timestamp',
19      })
20      releaseTime: Date
21
22      @Column({
23          nullable: true,
24          type: 'json',
25      })
26      releaseDelay: { hours: number, minutes: number }
```

Kuvio 21 Esimerkki entitystä

Jokainen entity paljastaa luokastaan repositorion ja mahdollistaa täten toimintojen toteuttamisen repositorion avulla kyseiselle taululle tietokannassa. Kuvio 22 sisältää esimerkin yksinkertaisesta hausta tietokantaan repositorion avulla.

```
@Injectable()
export class ReportService {
    constructor(
        @InjectRepository(Report)
        private readonly reportRepository: Repository<Report>,
    ){}

    async getReportById(id){
        return await this.reportRepository.find({
            where: {
                id
            }
        })
    }

```

Kuvio 22 Esimerkki repositorion käytöstä

3.5 Kontitus

Palvelu sisältää itsessään kaksi kuvaa, yhden käyttöliittymälle ja toisen palvelimelle.

Kuvat luodaan node:12.16.3 kuvan päälle ja lähdekoodi tuodaan pakettien asennuksen jälkeen kontin sisälle. Käyttöliittymä tarvitsee www-palvelimen jakaakseen tiedostonsa niitä pyytävälle.

frontend > Dockerfile > ...

```
# build stage
FROM node:12.16.3-alpine as build-stage
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
RUN npm run lint
RUN npm run build

# production stage
FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

backend > Dockerfile > ...

```
# build stage
FROM node:12.16.3-alpine AS build-stage
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 3000
RUN npm run build
ENTRYPOINT [ ]
```

Kuvio 23 palvelun Dockerfilet

Sovelluksen yhteneväisyyden takaamiseksi otetaan Docker Compose-työkalu käyttöön. Composen avulla pystytään määrittelemään ja ajamaan monta eri konttia saman aikaisesti.

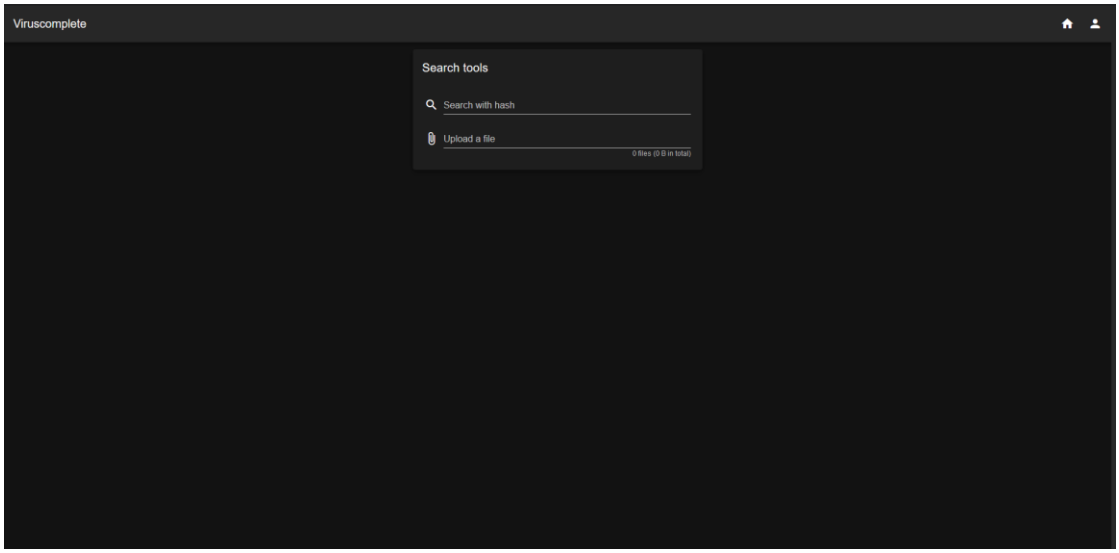
```yaml
version: '3.3'
services:
    viruscomplete_pg:
        image: postgres
        container_name: viruscomplete_pg
        restart: always
        ports:
            - 5432:5432
        environment:
            POSTGRES_PASSWORD: admin
            POSTGRES_USER: admin
            POSTGRES_DB: ioc_palvelu
        volumes:
            - /var/lib/postgresql/data
    viruscomplete_backend:
        image: viruscomplete_backend
        container_name: viruscomplete_backend
        build: backend/.
        command: sh -c "npm run start:prod"
        ports:
            - 3000:3000
        volumes:
            - /app/
    viruscomplete_frontend:
        image: viruscomplete_frontend
        build: frontend/.
        container_name: viruscomplete_frontend
        ports:
            - 8080:80
volumes:
    viruscomplete_pg:
    viruscomplete_backend:
```

Kuvio 24 docker-compose.yml

Sovelluksen ympäristö määräytyy Dockerfilejen perusteella, kun taas Compose määrittää sovelluksen eri osuudet. Compose tiedoston voi ajaa komentolinjalta komennolla ’docker compose up’.

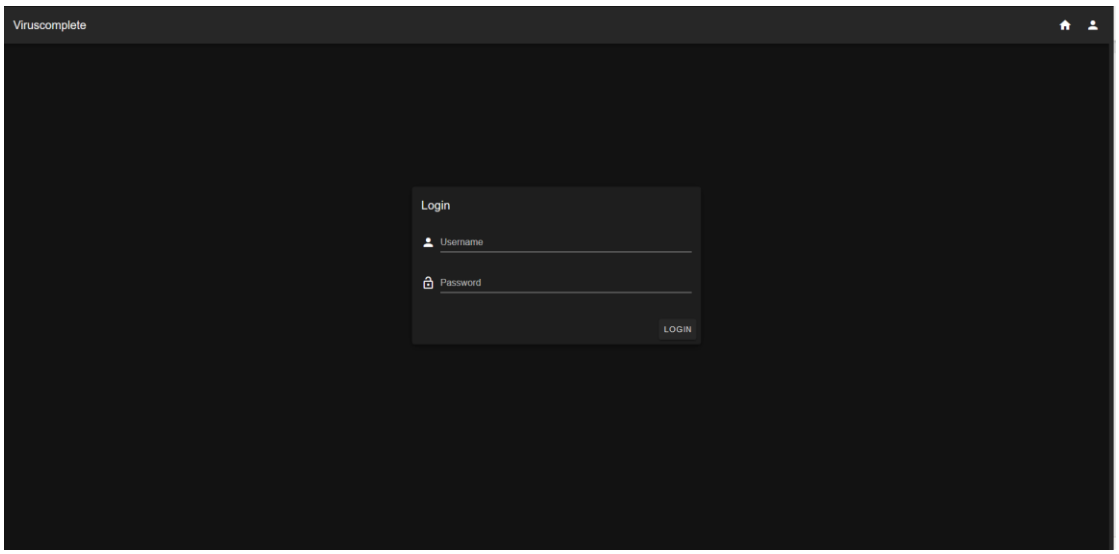
4 Tulos: Viruscomplete

Julkinen puoli sovelluksesta, kuvio 25, kirjautumattomalle käyttäjälle sisältää tiedoston lähettämisen, tiivisteellä hakemisen sekä linkin sisäänkirjautumiseen.



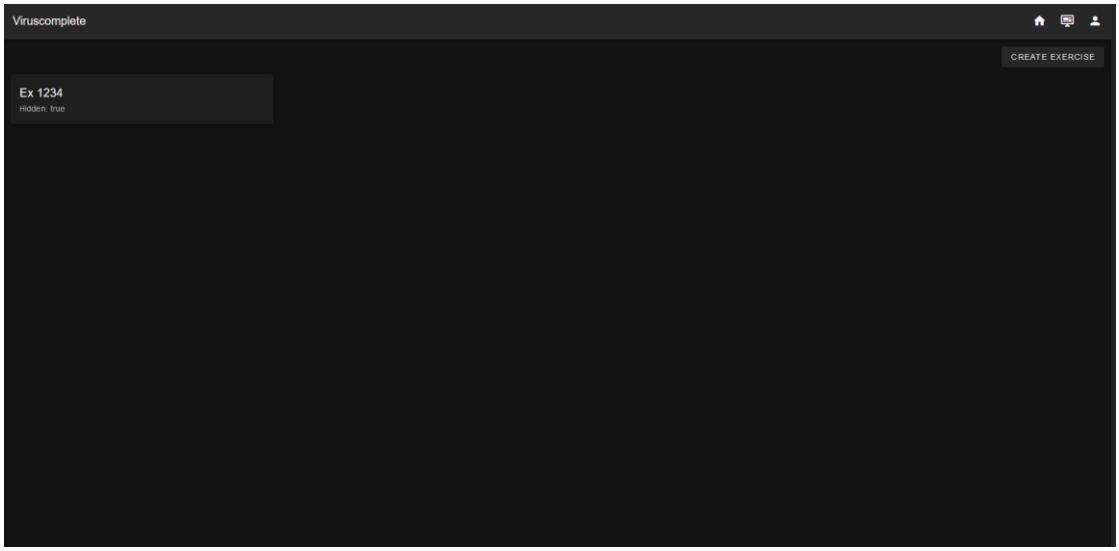
Kuvio 25 Palvelun etusivu

Kuvio 26 näyttää sisäänkirjautumisen näkymän.



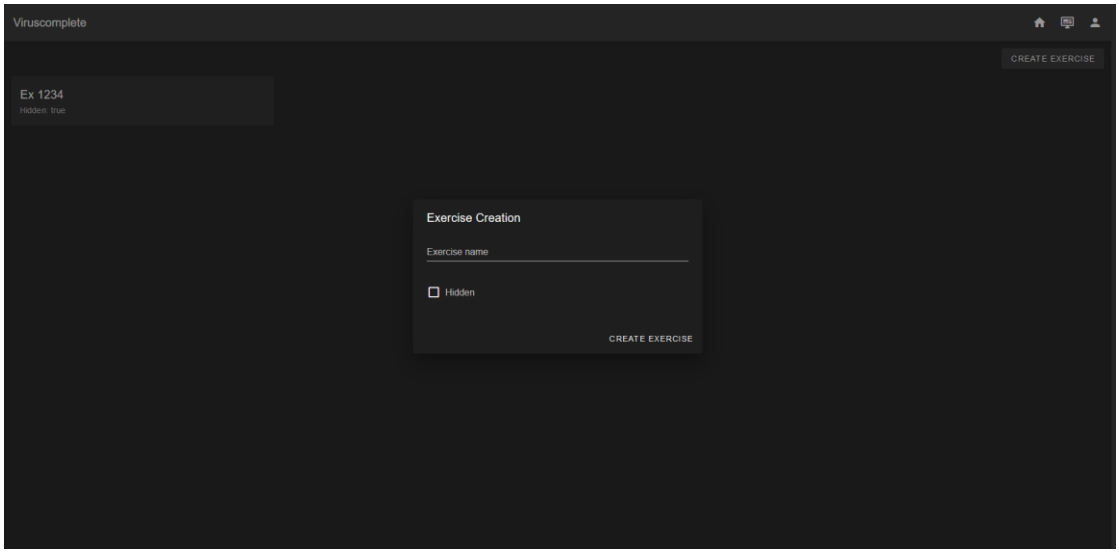
Kuvio 26 Sisäänkirjautumisen näkymä

Sisäänkirjautumisen jälkeen käyttäjälle avautuu kuva ohjauspaneelista, kuvio 27. Ohjauspaneelissa harjoituksia on mahdollista luoda tai käyttäjä voi valita olemassa olevan harjoituksen.



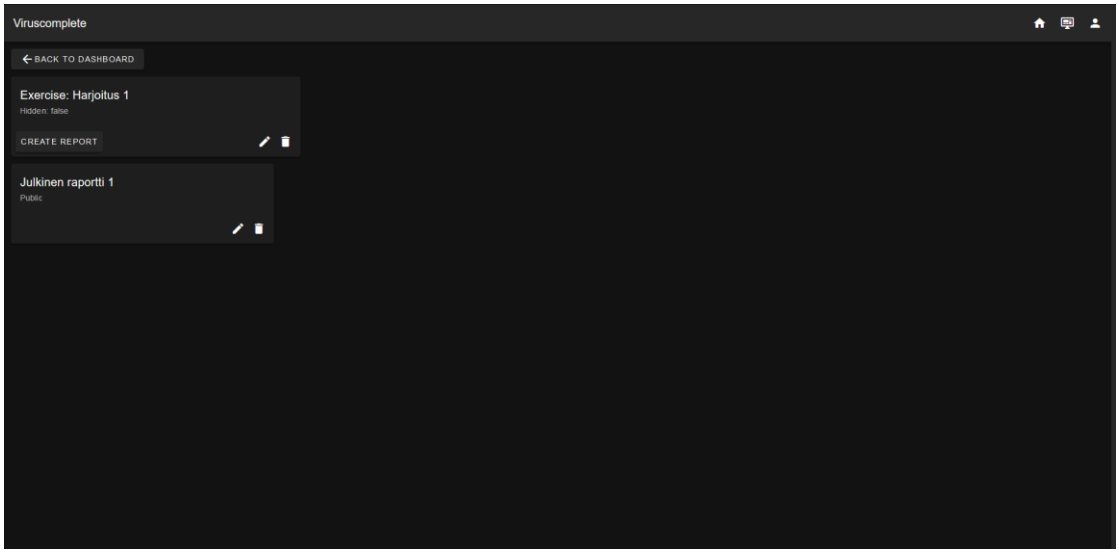
Kuvio 27 Ohjauspaneeli

Harjoituksen luomisessa, kuvio 28, tärkein osuus on harjoituksen asettaminen joko piilotetuksi tai julkiseksi. Jos harjoitus on piilotettu, sen alla olevat raportit eivät tule hakutuloksiin mukaan.



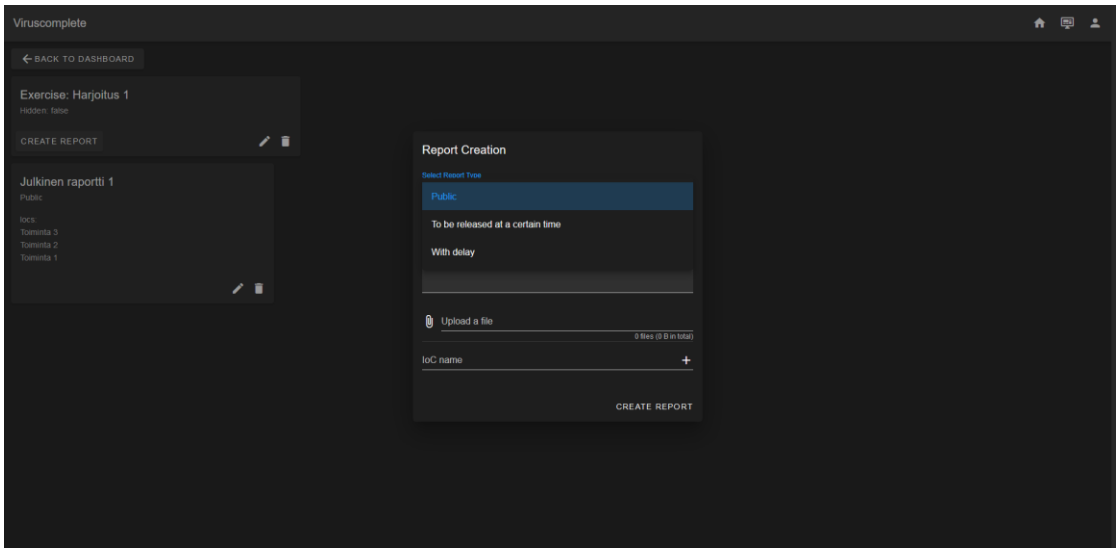
Kuvio 28 Harjoituksen luomisdialogi

Harjoituksen näkymässä, kuvio 29, harjoituksen tietoja voidaan muokata tai poistaa. Raportteja pystyy myös luomaan, muokkaamaan tai poistamaan.



Kuvio 29 Harjoituksen näkymä

Raportin luonnissa, käyttötapausten perusteella on mahdollista luoda julkinen, viivellinen raportti tai vaihtoehtoisesti raportti, joka tulee tietyllä hetkellä julkiseksi. Kuviossa 30 on raportin luomislomake auki.



Kuvio 30 Raportin luonti

Backend toimii järjestelmässä CRUD-osuutena, muuntaen sille tulevien kyselyiden perusteella tietokannassa olevaa tietoa.

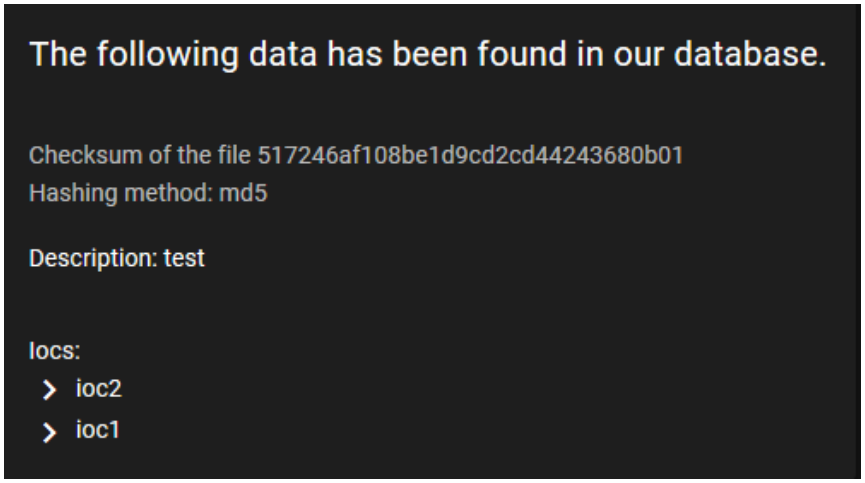
Joka kerta, kun julkinen käyttäjä lähettää järjestelmään tiedoston, se tallennetaan palvelimen yhteyteen. Julkisten käyttäjien lähettämät tiedostot kuitenkin siivotaan pois järjestelmästä Crontab-ajastuksella.

Käyttötapauksien julkisen raportin tietojen palautus tapahtuu yksinkertaisella haulla tietokantaan. Julkinen käyttäjä lähettää tiedoston tai tiivisteen. Tiivistettä verrataan tietokannassa oleviin tiivisteisiin ja tulos palautetaan käyttäjälle.

Raportin teko julkiseksi tiettynä aikana tapahtuu Crontab-ajastuksella. Ajastuksessa tietokannasta haetaan raportit, mitkä eivät ole julkisia, eikä niillä ole viivettä julkaisun asettamiseksi. Raporteista, jotka löytyivät tietokannasta annetuilla hakuehdoilla, tarkastetaan releaseTime-kenttä. Raportin releaseTime kolumnia verrataan nykyhetkeen ja jos kentän arvo on pienempi kuin nykyhetki, asetetaan raportin julkisuus-kenttä todeksi.

Viiveellisen raportin julkistaminen tapahtuu setTimeout funktiolla. SetTimeout ajetaan annettujen millisekunttien jälkeen. Tässä tapauksessa raportin sisältämä viive muutetaan millisekunneiksi, mikä syötetään parametriksi setTimeout-funktiolle. Kuluneen ajan jälkeen raportin julkisuus kolumni asetetaan todeksi.

Haun jälkeen käyttäjä ohjataan uuteen näkymään ja tulokset näytetään käyttäjälle, kuvio 31.



Kuvio 31 Esimerkki Haun tuloksista

5 Pohdinta

Viruscomplete on erittäin käyttötapauskohtainen palvelu ja pienillä muutoksilla palvelusta pystyisi muokkaamaan yleispätevän tuotteen. Raportit palvelussa kuvastavat mahdollisia toimintamalleja käyttäjien toimintaan, jos ei-haluttu tiedosto löytyy järjestelmästä. Viruscompleten ideapohjana toimi VirusTotal, mikä on verkossa toimiva skannausohjelma.

Jatkokehityksenä Viruscompleteen sopisi tietynlainen url-scannaus eli tarkistetaan onko syötetty sivusto pahansuopa vai ei. Tietokantaan lisättäisiin tarvitut taulut ja sivusto käytäisiin tarkistamassa haun yhteydessä.

Heurestiikka-analyysin lisääminen palveluun ei olisi mahdoton idea. Heurestiikka analyysissä ohjelma tai tiedosto suoritetaan eristetyssä ympäristössä ja ohjelman toimintaa tarkkaillaan. Jos epäilyttävää toimintaa löytyy suorittamisessa, palautettaisiin epäilyttävästä toiminnasta tiedot käyttäjälle.

Tietokannan mallissa on otettu huomioon MD5-heikkoudet ja palvelua on mahdollista lähteä kehittämään suuntaan, missä tiivisteitä pystyttäisiin muodostamaan muillakin algoritmeilla.

Työn teoriaosuus on rajattu hyvin rankalla kädellä, sillä jokaisesta osuudesta pystyisi syventymään omaan opinnäytetyöhönsä. Teknologioiden rajauksessa pääasiana toimii käyttötarve sekä sovelluksen jatkokehitykseen liittyvät mahdollisuudet. Teoriaosuus rajattiin sisällyttämään yleisellä tasolla toteutuksen teknologioita ja teknologioista käytettyjä asioita.

Lähteet

Abdullah, M. 2016. SPM System Cybersecurity. Opinnäytetyö, AMK. Vaasan Ammattikorkeakoulu, tekniikan ala. Viitattu 30.3.2021. http://urn.fi/URN:NBN:fi:amk-2016061312783

About Us. N.d. Artikkeli JYVSECTECin www-sivuilla. Viitattu 18.8.2020. https://jyvsectec.fi/about/overview/

Cyber Exercises Overview. N.d. Artikkeli JYVSECTECin www-sivuilla. Viitattu 18.8.2020. https://jyvsectec.fi/services/exercises/overview/

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Tohtorintutkinto. Kalifornian yliopiston kampus Irvine, tieto- ja viestintätekniikka. Viitattu 22.3.2021 https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Fielding, R., Reschke, J. 2014. Http/1.1 Semantics and Content. RFC, IETF. Viitattu 11.4.2021 https://tools.ietf.org/html/rfc7231

Hacq A. 2018. Everything you need to know about Design Systems. Artikkeli uxdesign.cc verkkosivuilla. Viitattu 11.4.2021. https://uxdesign.cc/everything-you-need-to-know-about-design-systems-54b109851969

Introduction to client-side frameworks. N.d. MDN Web Docs. Viitattu 22.3.2021. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction

Introduction to JSON Web Tokens. N.d. Artikkeli jwt.io verkkosivuilla. Viitattu 5.11.2020 https://jwt.io/introduction/

Material System Introduction. N.d. Artikkeli Material Designin www-sivuilta. Viitattu 22.3.2021 https://material.io/design/introduction

Nestjs Introduction. N.d. Nestjs dokumentaatio. Viitattu 21.9.2020 https://docs.nestjs.com/

Nestjs Techniques. N.d. Nestjs dokumentaatio. Viitattu 27.10.2020 https://docs.nestjs.com/techniques

Niemelä, J. 2015 Statistical Analysis Of Malware Defence Methods. Opinnäytetyö, ylempi AMK. Jyväskylän ammattikorkeakoulu, tekniikan ala. Viitattu 30.10.2020 http://urn.fi/URN:NBN:fi:amk-201601071096

Pakkanen, A. 2018. ASP.NET yhteensopivien hash-algoritmien vertailu ja implementaatio: case Visma Tampuuri Oy. Opinnäytetyö, AMK. Turun ammattikorkeakoulu, tietojenkäsittely. Viitattu 30.3.2021 http://urn.fi/URN:NBN:fi:amk-2018121020736

Ramirez, G. 2015. MD5: The broken algorithm. Artikkeli Aviran blogissa. https://www.avira.com/en/blog/md5-the-broken-algorithm

Rivest, R. 1992. The MD5 Message-Digest Algorithm. RFC, IETF. Viitattu 22.3.2021 https://tools.ietf.org/html/rfc1321

Vue.js Essentials. N.d. Vue.js dokumentaatio. Viitattu 22.3.2021. https://vuejs.org/v2/guide/

What is a Container? N.d. Artikkeli dockerin sivuilla. Viitattu 29.10.2020 https://www.docker.com/resources/what-container

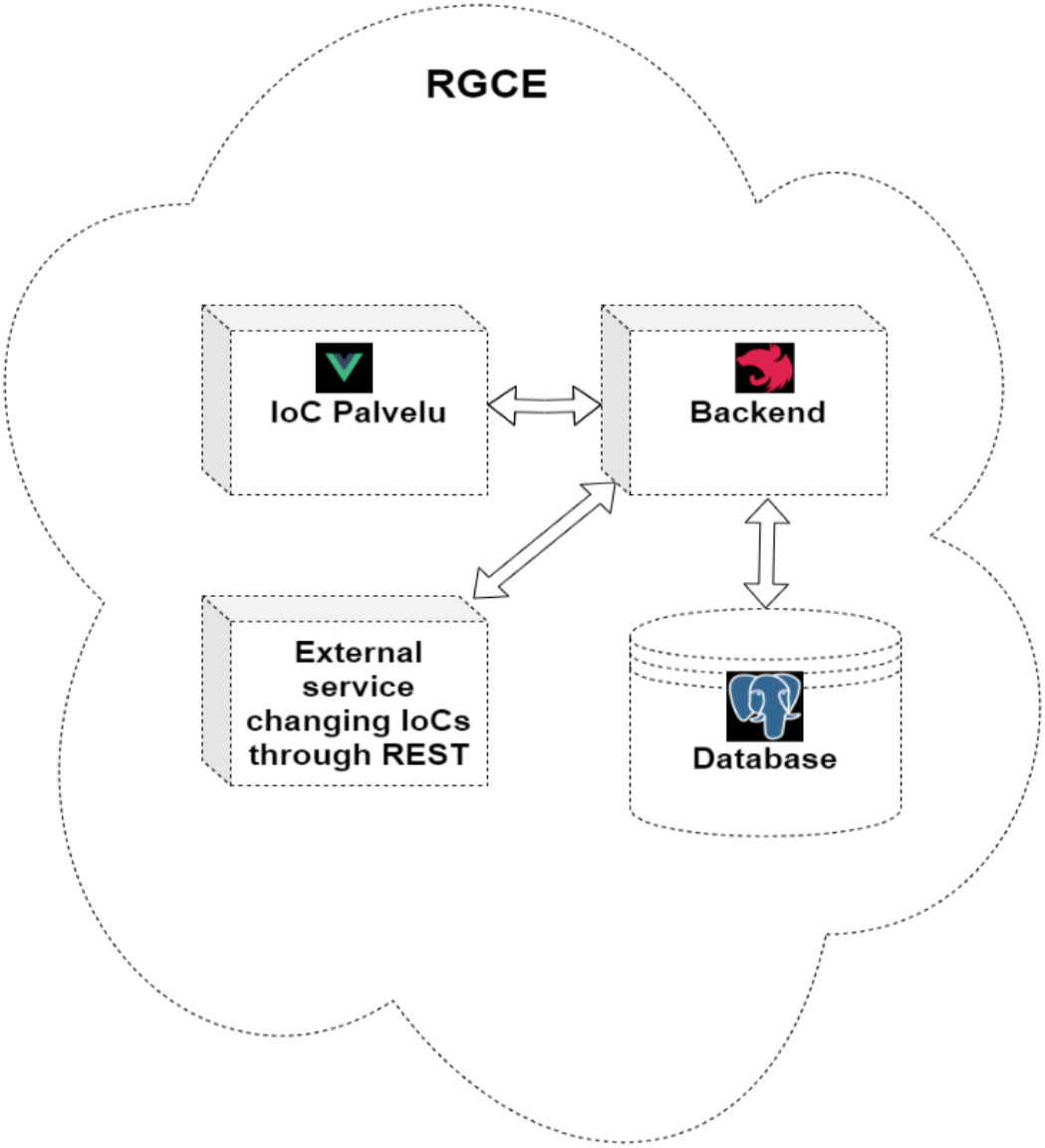
What is NoSQL? N.d. Artikkeli mongodb:n verkkosivuilta. Viitattu 11.4.2021. https://www.mongodb.com/nosql-explained

Web technology for developers. 2021. MDN Web Docs. Viitattu 22.3.2021 https://developer.mozilla.org/en-US/docs/Web/HTTP

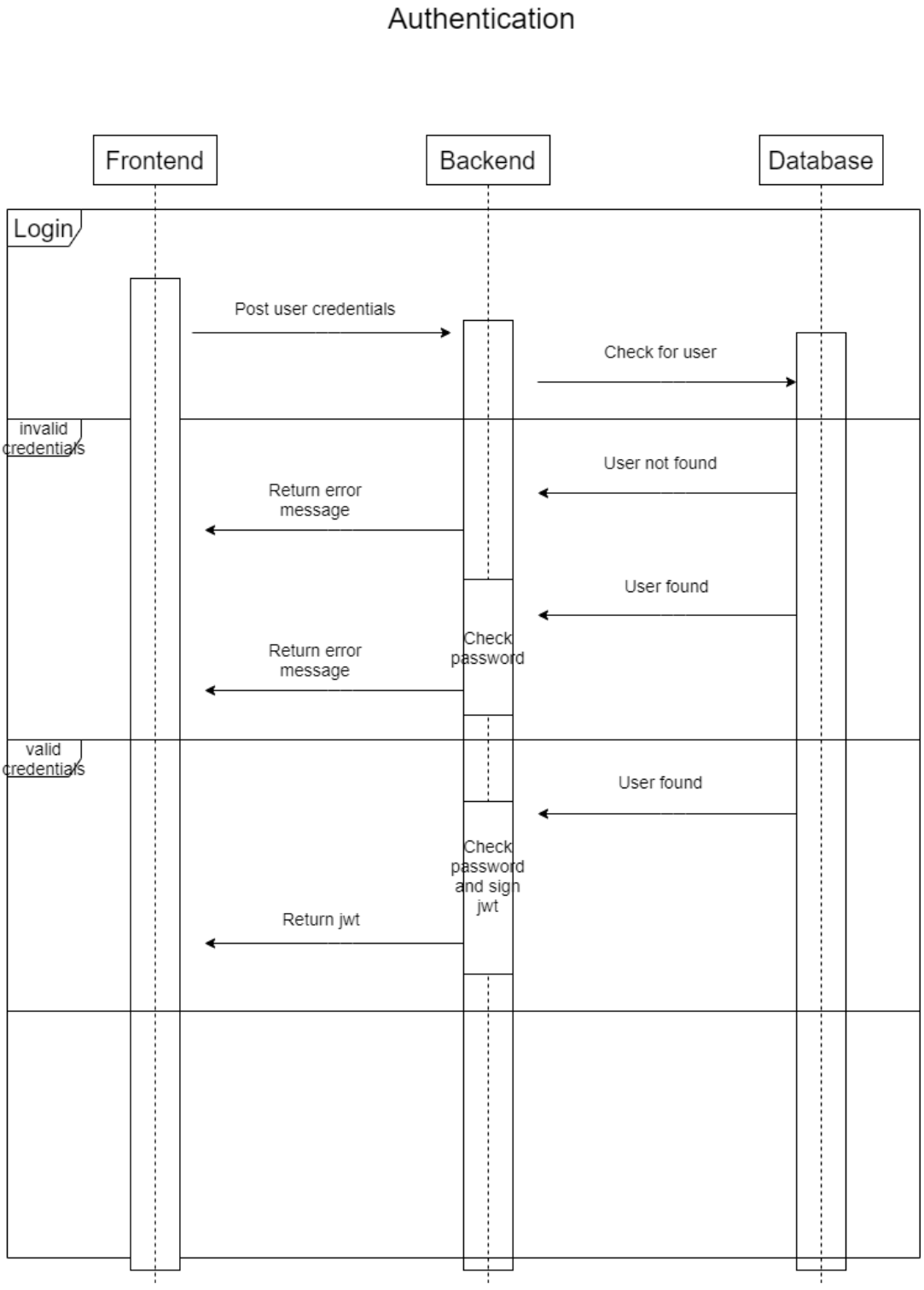
2020 Developer Survey. 2020. Stackoverflown vuosittainen kehittäjille tarkoitettu kysely. Viitattu 22.3.2021. https://insights.stackoverflow.com/survey/2020#technology-web-frameworks

Liitteet

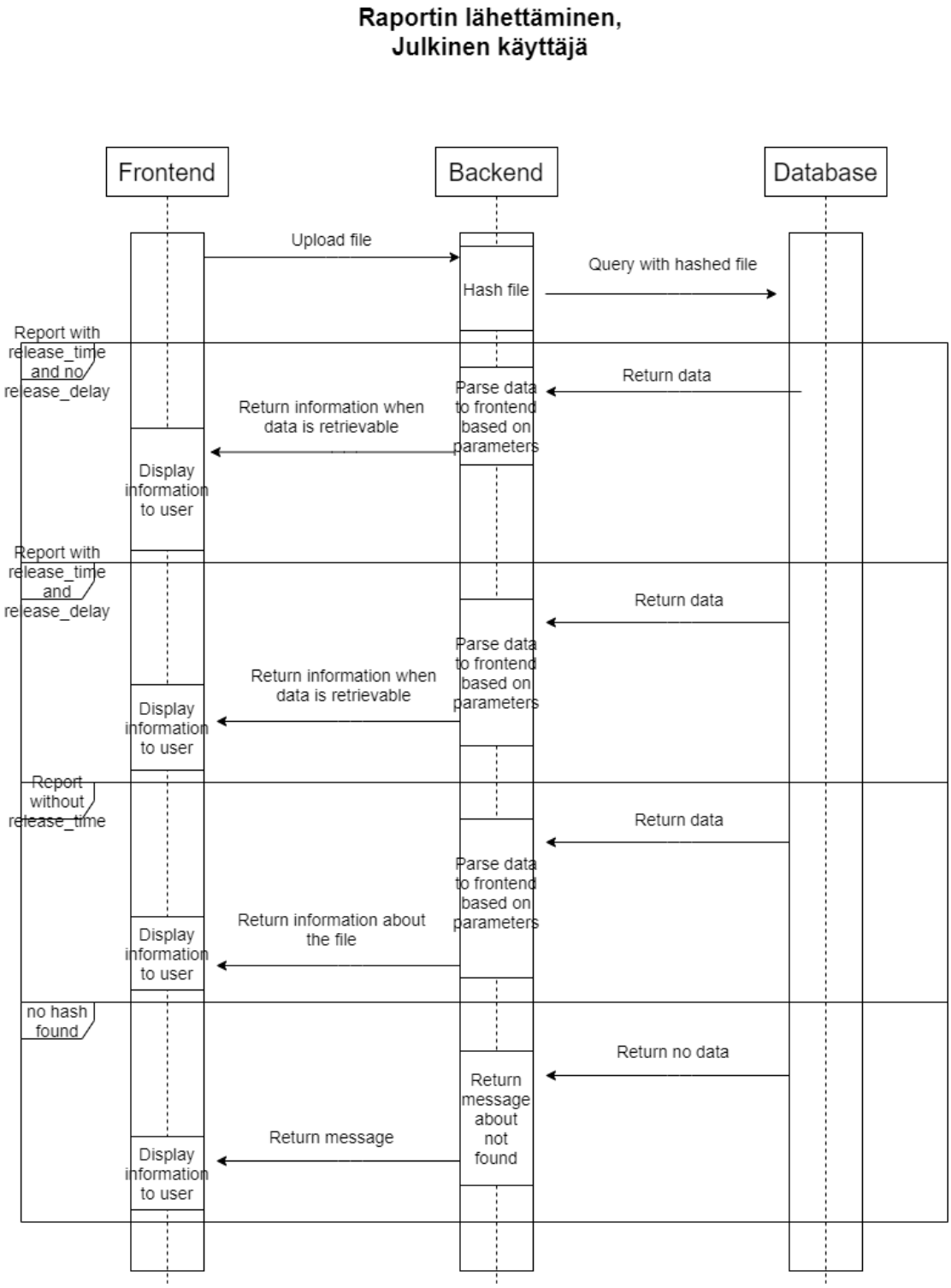
Liite 1 RGCE IoC palvelun arkkitehtuuri



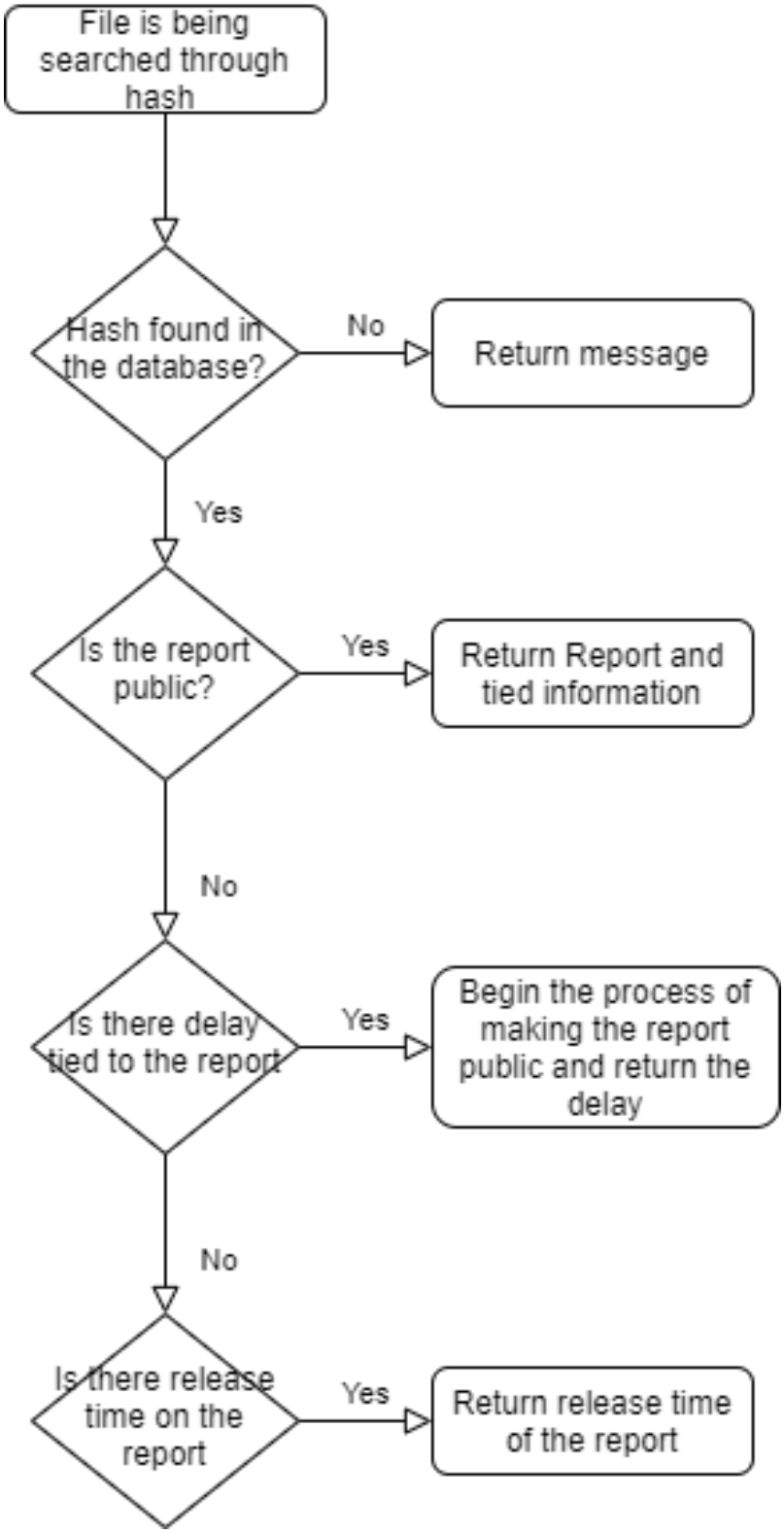
Liite 2 Tunnistautumisen sekvenssikaavio



Liite 3 Tiedoston lähettäminen palveluun sekvenssikaaviona



Liite 4 Tiedostojen hakemisen prosessikaavio



Liite 5 Tietokantarakenne