OULUN AMMATTIKORKEAKOULU

Marno Kulmala

CV-PANKKI

CV-PANKKI

Marno Kulmala
Opinnäytetyö
Kevät 2019
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma

Tekijä: Marno Kulmala
Opinnäytetyön nimi: CV-pankki
Työn ohjaaja: Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: kevät 2019
Sivumäärä: 58

Työssä pyrittiin toteuttamaan CV-pankki, jonka avulla konsulttien CV:itä voidaan hallinnoida helposti. Tavoitteena oli saada aikaan toimiva runko sovellukselle, jota voitaisiin myöhemmin kehittää eteenpäin tarpeen näin vaatiessa.

Työ tehtiin Kotlin-ohjelmointikielellä Spring Boot viitekehystä hyväksikäyttäen. Tämän lisäksi käytettiin GraphQL-protokollaa tiedon hakemiseen ja muuttamiseen. Tieto tallennetaan dokumentteina MongoDB-tietokantaan. Näiden lisäksi käytettiin HackMyResume-nimistä node.js-sovellusta CV:eiden muuntamiseen JSON:sta PDF:ksi. Sovellus suunniteltiin toimivaksi Docker-kontin sisällä, jolloin se voidaan asentaa helposti mihin tahansa ympäristöön. Sovelluksen suunnitteluvaihe eteni nopeasti ja sovelluksen runko saatiin aikaiseksi parissa viikossa. Sitä seurannut toteutusvaihe vaati runsaan kuukauden. Testausvaihe toteutui limittäin toteutusvaiheen kanssa. Kaiken kaikkiaan aikataulu oli tiukka, mutta työ onnistui alun kriittisen poissulkemisen ansiosta.

Sovellus toimii ja se voidaan asentaa mille tahansa alustalle. Tästä huolimatta testausvaiheen tulokset olivat selkeät: sovellusta tulee muuttaa tulevaisuudessa yksinkertaisemmaksi ja osaa toiminnollisuuksista tulee muokata erilaisiksi. Sovelluksen jatkokehittely jatkuu ensi vuonna, koska alustavat testitulokset olivat positiivisia ja mahdollisten muutosten määrä oli pieni.

Asiasanat: GraphQL, JSON, Kotlin, Spring Boot

ABSTRACT

Oulu University of Applied Sciences
Information Technology

Author: Marno Kulmala
Title of thesis: CV-bank
Supervisor: Pekka Alaluukas
Term and year when the thesis was submitted: kevät 2019
Pages: 58

Idea from CV-bank was born from need to send CV of consultant faster to would be customer. Need was also for application that even little experienced consultant could work with it and learn Java based technology by doing so.
Technology in CV-bank is based on Spring Boot and programming language is Kotlin, querying and communications is handled by GraphQL and the data is stored in MongoDB. In addition HackMyResume node.js application was used to do transformation to CV formats form JSON to PDF.

Design phase of application was short and was done in couple of weeks. The implementation phase was month or so and the testing phase again only couple of weeks. Timetable was strict, but succeeded because of heavy exclusion during design phase: only the bare minimum of features was chosen to be implemented.

Application works and can be installed in everywhere. Having said that there are still some problems found during testing and these are to be fixed next year. Also some other features are to be added and old ones to be changed. But all in all application was successfull.

Keywords: GraphQL, JSON, Kotlin, Spring Boot

ALKULAUSE

Olen opiskellut vuodesta 1998 lähtien aktiivisen epäaktiivisesti – minua on aina vienyt työt koulujen edelle. Vuosia olin yksi sadoista suomalaisista ict-alan ”college dropouteista”, jotka tällä hetkellä tekevät paljon erilaisia sovelluksia monissa eri yrityksissä. Tämä asia muuttui, kun sain toisen lapseni ja halusin olla lapsilleni esimerkkinä: en halunnut olla se isä, joka huutaa, että tehkää niin kuin minä sanon, ei niin kuin minä teen! Tämän ajatuksen johdattelemana hain aikuiskoulutukseen Metropoliaan lukemaan itseäni insinööriksi, mutta huomasin pian sen olevan erittäin raskasta pienen lapsen isälle, jolla on iltaisin jotain muutakin tekemistä kuin algebra ja muut tekniset oppiaineet. Ajatus jäi hautomaan mielessäni ja vuonna 2013 löysin Oulun ammattikorkeakoulusta aikuiskoulutuslinjan, jossa voisi opiskella koko tutkinnon etänä – olin löytänyt oppilaitokseni!

Opiskeluaikani OAMK:ssa on ollut opettavaista ja hauskaa. Vaikka olenkin ohjelmoinut vuosia työkseni, koen saaneeni kouluni opettajilta paljon lisäoppia matkaani ja toivon myös antaneeni takaisin jotakin, edes pieniä tiedonjyväsiä siitä, miten kentällä näitä asioita sovelletaan. Olen myös aina pyrkinyt kiittämään OAMK:n opettajistoa heidän työstään ja tuestaan (ja myös heidän pettämättömästä jaksamisestaan kaltaiseni pilvilinnojen rakentajan kanssa). Heidän myötävaikutuksensa oli suuri tämänkin työn loppuun saamiseksi. Joten lopetan tämänkin tekstin sanoihin: kiitos OAMK:n opettajat!

Helsinki 14.6.2019

Marno Kulmala

SISÄLLYS

SANASTO

Docker

Docker on eräs kevyt virtualisointitapa ja se tarkoittaa sovelluksen ja sovellusalustan paketointia tavalla, jolla niitä voidaan ajaa kevyissä konteissa käyttöjärjestelmätason virtuaalipalvelimissa.

GraphQL

Aluperin Facebookin kehittämä teknologia omien mobiilisovellustensa tarpeisiin. Julkaistu vuonna 2015 avoimena koodina. Tämän jälkeen GraphQL:ää ovat lähteneet hyödyntämään monet tahot, jotka haluavat tarjota perinteisempää RESTiä monipuolisemman rajapintakerroksen.

Kotlin

Androidin uusi virallinen ohjelmointikieli, joka sisältää paljon uusia ja ohjelmointia helpottavia ominaisuuksia. Sitä voidaan käyttää helposti Javan kanssa, tai sen sijaan.

Node.js

Avoimen lähdekoodin alustariippumaton JavaScript-ajoympäristö JavaScript-koodin suorittamiseen palvelimella.

PDF

Lyhenne sanoista Portable Document Format. Postscript-kieleen perustuva siirrettävissä oleva tiedostomuoto, jota käytetään sähköiseen julkaisemiseen, tulostamiseen ja painamiseen.

Spring

Suosittu Open Source -kehys Java-ohjelmointiin. Sitä voidaan käyttää Java EE:n sijaan. Spring Boot mikropalveluarkkitehtuurin pohjana on entisestään nostanut Springin suosiota. Spring on kokoelma teknologioita.

1 JOHDANTO

Työn tavoitteena oli toteuttaa CV:iden hallintaan soveltuva sovellus. Alkuperäinen ajatus lähti työn alkuperäisen tilaajan myyntipäällikön puheista konsulttiosaamisen hallinnoinnin vaikeuksista. Myös työntekijänä ja pitkän linjan konsulttina huomasin tekeväni vuosittain useita CV:itä – en pelkästään omalle yritykselleni, vaan myös asiakkaille ja mahdollisesti myös heidän asiakkailleen. Tajusin, että tähän olisi pakko olla jokin yksinkertaisempikin tapa.

Ohjelmistokonsultointiyritykset perustuvat karrikoidusti ansioluetteloiden lähettelyyn asiakasyrityksiin (toki ne tekevät myös lupaamansa asiat, mutta näin yksinkertaistaen ohjelmistokonsulttiyritykset siirtävät konsulttejaan asiakkaalta toiselle). Nämä ansioluettelot eli CV:tä pitävät sisällään asiakkaalle tarjottavan konsultin osaamisen kirjallisessa muodossa ja ne muuttavat usein konsulttien vaihtaessa asiakkaalta toiselle. Monesti tämä on raskasta käsityötä, johon kallis konsultti joutuu käyttämään aikaansa: konsultin CV muunnetaan usein eri asiakkaalle sopiviksi ja siinä olevaa tietoa päivitetään jatkuvasti. Tämän tehostaminen ja automatisointi säästää sekä aikaa että rahaa. Tämä kaikki kiteytyy sanoihin: ketä on vapaa tietynlaisella osaamisella?

Ratkaisun pohjana toimii nykyaikaan soveltuva tiedonvälityksen formaatti JSON, joka kuvaa tiedon tarkasti ja josta voidaan nykyaikaisilla kehitysmenetelmillä muuntaa halutun muotoinen ansioluettelo automaattisesti. Kun tähän vielä lisätään tunnettu Java-viitekehys ja moderni tietoliikenneprotokolla, jota voidaan käyttää graafimaisten kyselyiden tekemiseen, saadaan yksinkertaisuudessaan tässä työssä kuvattu CV-pankkiratkaisu.

Vaikka sovelluksen idea onkin yksinkertainen, se kätkee taakseen monimutkaisen rakennelman, jota käytetään yksinkertaisella tavalla graafisen integroidun kehittimen kautta. Tieto pyritään kuvaamaan mahdollisimman vapaasti, jottei sovellus joudu ottamaan liikaa kantaan CV: lopulliseen muotoon. Javan GraphQL-kirjasto tarjoaa monet ratkaisut valmiina, mutta ratkaisuun tarvittiin muitakin osasia, jotta kokonaisuus toimisi. Näin päädyttiin luomaan myös CV:iden muunto PDF:ksi, koska ei haluttu mahdollisen CV:n saajan pystyvän helposti muuttamaan CV:tä, ja tämän muunnetun tiedon lähettäminen sähköpostitse.

CV-pankin tilasi PHZ Fullstack, mutta kesken työn tapahtui työpaikan muutos, joka otettiin huomioon alkuperäisen ratkaisun pienentämisellä ja sen muuttamisella avoimen lähdekoodin projektiksi. Tämä vaikutti muun muassa tietoturvaan, joka määriteltiin käyttäjän päävaivaksi, samoin kuin muutkin autentikoimismenetelmät. Lisäksi sovelluksen käyttöliittymäksi päätyi valmiina GraphQL:n mukana tuleva selaimessa toimiva GraphiQL-kehitysympäristö. Tämän avulla voitiin helposti testata kaikki toiminallisuudet ja täten todentaa sovellus toimivaksi.

Kaikkia ideoita ei saatu muodostettua tarpeeksi koherentisti, jotta ne olisivat päässeet suunnitelmaan saakka, eikä sitä kautta toteutukseen, mutta kaikki ideat kirjattiin ja CV-pankki varmasti jatkaa elämäänsä sovelluksena.

2 CV-PANKIN TOTEUTUKSEN MÄÄRITTELY

CV-pankin vaatimukset olivat melko yksinkertaiset: toimeksianto oli saada yksinkertainen CV:iden tallennusjärjestelmä, jota olisi helppo käyttää.

Teknisiä vaatimuksia oli vähän ja ne olivat lähinnä periaatteiden tasolla. Mitään teknologia-alustaa ei määritelty tarkkaan. Alkuperäinen tarkoitus oli, että CV-pankki toimisi opetuksellisena alustana nuoremmille ohjelmoijilla ja siihen ajateltiin Spring-viitekehystä Java-ohjelmointikielen kanssa. Näiden yhdistelmälle pystyttytäisiin luomaan, päivittämään ja poistamaan tietovarastosta CV:itä. Lyhyen testaus- ja tutkimusvaiheen jälkeen päädyttiin valitsemaan kuitenkin ohjelmointikieleksi sovellukselle uudempi Kotlin-ohjelmointikieli, viitekehyksen säilyessä Springinä. Tämän lisäksi valittiin GraphQL-viestintäprotokolla, koska haluttiin kokeilla miten sen kanssa voisi tehdä perinteisesti REST-rajapintaa hyödyntävä sovellus. Tietojenvarastointi alustaksi valittiin MongoDB dokumenttitietokanta. Tämä valinta oli sinänsä selvää, koska siitä oli aikaisempaa kokemusta ja CV-mallina käytettiin ilmaista ja avoimen lähdekoodin mallia nimeltään FRESH, joka pohjautuu ohjelmoijien itsensä suunnittelemaan JSON-skeemaan.

Näiden teknologisten vaatimusten lisäksi oli myyntiorganisaatiolta saadut vaatimukset: CV pohjien muuttaminen mahdollisten asiakkaiden CV:iden muotoiseksi, mahdollisuus kysellä henkilön CV:tä tämän osaamien teknologioiden ja käytettävyyden mukaan. Lisäksi haluttiin ominaisuus, jolla CV voitaisiin lähettää valitulle asiakkaalle.

Nämä ominaisuudet toteutettiin, mutta vain palvelin sovelluksena. Käyttöliittymänä toimii GraphiQL-sovellus, jota voi käyttää graafisena sovelluskehittiminenä CV:iden käsittelemiseen. Tosin palvelimen rajapintoja hyödyntämällä mikä tahansa käyttöliittymä on helppo toteuttaa.

2.1 CV-pankin arkkitehtuuri

Suunnittelun ohessa CV-pankista piirrettiin monia kuvia ja kaavioita. Suurin osa näistä oli lähinnä jonkin periaatteen tai operaation näyttämiseksi ja todistamiseksi. Myöhemmin näistä koottiin kolme järjestelmälle tärkeää kaaviota: kokonaisarkkitehtuuri kaavio, josta näkee järjestelmän kokonaisuudessaan. Tämän lisäksi on sekvenssikaavio, jolla

kuvataan CV-pankissa yksinkertaisesti datan siirtymistä vastuualueelta toiseen ja toinen kaavio kuvaa tärkeimmät käyttötapaukset.

CV-pankin arkkitehtuuri ratkaisu perustuu mikropalveluiden arkkitehtuuriin, joka voidaan kuvata hyvin seuraavalla lauseella:

”Mikropalveluiden idea on tiivistettävissä muutamaan sanaan. Perinteisessä ohjelmoinnissa kirjoitetaan yksi monoliittinen sovellus, joka hoitaa kaiken tarvittavan. Mikropalveluohjelmoinnissa kirjoitetaan joukko omina prosesseinaan toimivia itsenäisiä ja tilattomia palveluita. Ne käyttävät keskinäiseen kommunikointiinsa keveitä mekanismeja kuten rest-rajapintoja ja toteuttavat yhdessä sovellukselta halutut toiminnot.” (1, ensimmäinen kappale)

Mikropalvelut ovat syntyneet ketterän kehityksen aikaansaannoksena: suuret internetjätit käyttivät niitä web-palveluissaan. Tästä johtuen niiden piti skaalautua hyvin ja niiden kehittämisen piti olla helppoa. Tämän usein ajatellaan tarkoittavan, että mikropalvelu hoitaa yhden rajatun asian hyvin. Ja se voidaan tuottaa kulloinkin parhaiten tehtävään soveltuvalla ohjelmointikielellä, tai teknologialla. Mikropalveluja on myös helppo siirtää tuotantoon, kun siirto voidaan suorittaa pieninä ryhminä tai yksitellen – ongelmia kohdatessa takaisin siirtyminen edelliseen toimivaan versioon on helppoa. Mikropalvelujen etuna on myös se, että pullonkaulana toimivat palvelut on helppo monistaa kuormantasaajan taakse.

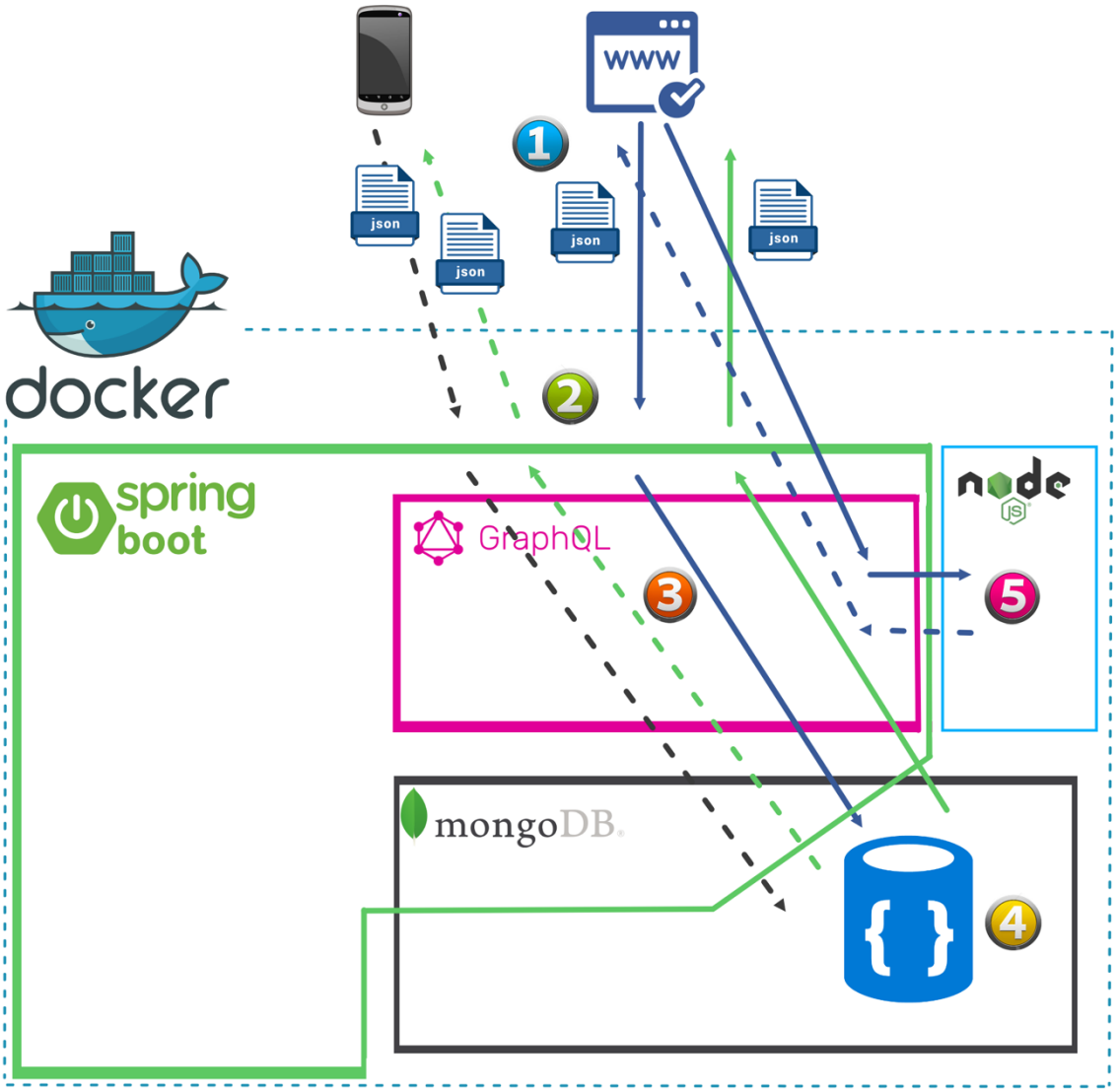
Seuraavaksi käydään tarkemmin läpi CV-pankin sisäinen arkkitehtuuri. Järjestelmän arkkitehtuuri on tarkoituksella mahdollisimman minimaalinen, jotta myöhemmin järjestelmän skaalaaminen tai sen muuttaminen olisi helpompaa. Ei myöskään haluttu luoda liian vanhanaikaiseen arkkitehtuuriin pohjautuvaa ratkaisua.

Kaiken perustana toimii Spring Boot, jota ajetaan Docker kontissa Debianin (Linux-käyttöjärjestelmä) päällä. Viestintäprotokollana toimii GraphQL ja tiedonsiirtotyyppinä JSON.

CV-pankki periaatteeltaan yksinkertainen mikropalveluarkkitehtuurin toteuttava sovellus. Ainut erottava tekijä moniin muihin moderneihin Java-teknologioilla rakennettuihin

sovellukseen on se, että käytetään ohjelmointikielenä Kotlinia ja kyselyissä ja viestintäprotokollana GraphQL:ää.

Kuva 1 näyttää kokonaisarkkitehtuuri kuvan CV-pankista. Tästä huomaa, että sovellus voidaan jakaa kuvassa havaittaviin erilaisiin osiin.



KUVA 1. CV-pankin kokonaisarkkitehtuuri

Ensimmäinen jakaja on eri väriset laatikot, jotka kertovat tietyn teknologian rajat ja niiden suhteet. Vihreällä katkoviivalla on jaettu Docker kontissa ajettavat teknologiat ja CV-pankki karkeasti. Tämän ulkopuolella on eri asiakassovellukset, kuten mahdollinen mobiilisovellus ja nettiselain, joista molemmista voi kutsua CV-pankkia. Paksu vihreä viiva kertoo Spring Boot viitekehyksen sisällä ajettavat teknologiat. Tämä periaatteessa

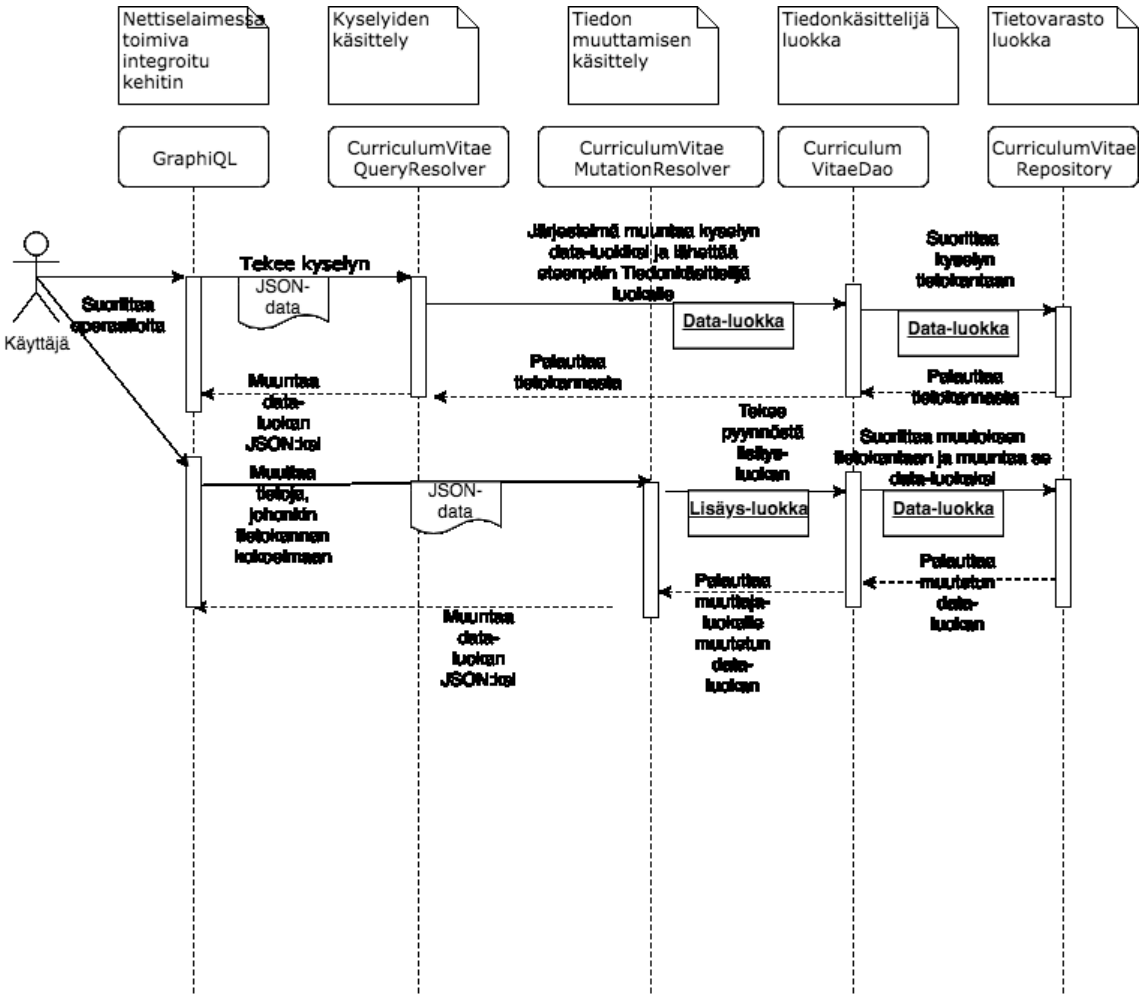
tarkoittaa MongoDB-kirjastoa ja Graphql-kirjastoa, joista molemmista on implementaatio Spring viitekehyksen sisäisissä riippuvaisuuksissa (dependency). Nämä sisältävät toteutuksen, jotta voidaan käyttää kyseessä olevaa teknologiaa. Punainen laatikko kertoo ,missä vaiheessa CV-pankkia kutsuttaessa siirrytään Graphql-viestintäprotokollan sisälle ,ja tumman harmaa kertoo, milloin kutsu on saavuttanut MongoDB dokumenttikannan. Itse MongoDB on asennettu Dockeriin, samoin kuin muutkin teknologiat, joita CV-pankissa tarvitaan.

GraphQL:n ja MongoDB:n lisäksi CV-pankissa hyödynnetään Hackmyresume nimistä node.js-sovellusta, jota myös ajetaan Dockerin sisällä ja jonka tehtävänä on muuntaa JSON PDF:ksi. Tämän lisäksi sillä on monia muita ominaisuuksia, joita esitellään myöhemmin luvussa 2.6.

Kutsu kuvataan kuvassa 1 numeroiden avulla. Tehdessään kutsun CV-pankkiin asiakas ensimmäisenä kutsuu Docker-kontin sisällä ajettavaa Spring Boot-sovellusta GraphQL-viestintäprotokollan läpi. Kutsu tehdään JSON muotoisena ja sen tulee olla FRESH JSON-mallin mukainen.

Kutsu CV-pankille alkaa numerosta 1, jolloin käyttäjä tekee tarvittavan JSON mallin GraphQL:n nettiselaimessa pyörivässä integroidussa ohjelmointiympäristössä GraphiQL:ssä ja lähettää sen avulla kutsun CV-pankille. Seuraavaksi kutsun nappaa kiinni Docker-kontissa ajettava Spring Boot ohjelmistokehyksellä tehty CV-pankki sovellus (numero 2), jonka GraphQL-kirjasto osaa käsitellä erilaisilla sillä lähetettyjä kyselyjä (queries) ja muutoksia (mutation). Tämä tapahtuu kohdassa 3. Tästä ohjelmistokehys osaa tallentaa MongoDB dokumenttikantaan oikeanlaisen CV:n FRESH JSON mallin muodossa (kohta 4). Kohdassa 5 voi kutsu CV-pankin kautta Docker-kontissa olevaa Hackmyresum nodejs-sovellusta, joka osaa muodostaa pdf-muotoisia asiakirjoja annetusta JSON-muotoisesta tiedosta.

CV-pankin sekvenssikaaviosta (kuva 2) selviää helposti, miten GraphQL-kutsut tulevat selainpohjaisesta integroidusta kehittimestä CV-pankkiin. Kutsut voidaan jakaa kahteen eri tyyppiin: tietoja kyseleviin kutsuihin ja tietoja muuttaviin kyselyihin ja kutsuihin.



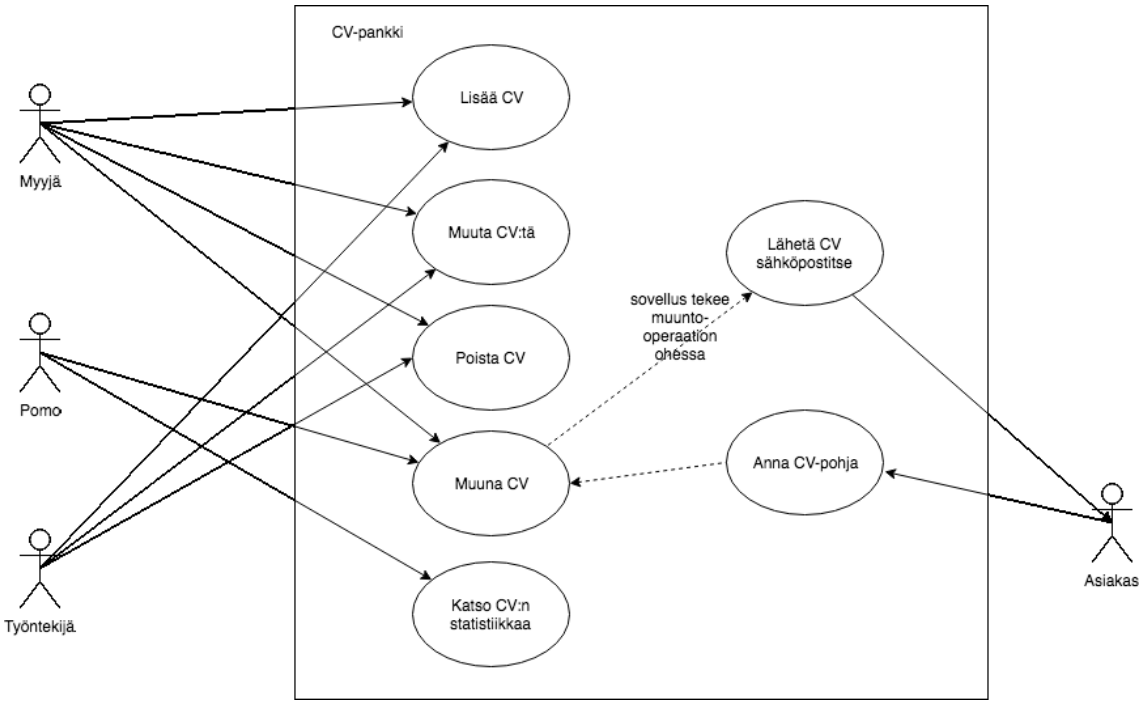
KUVA 2. CV-pankin sekvenssikaavio

Tietoja kyselevät kutsut hoitaa CurriculumVitaeQueryResolver-luokka ja tietoja muuttavia CurriculumVitaeMutationResolver-luokka. Rakenne haluttiin pitää mahdollisimma yksinkertaisena, joten kaikki operaatiot tulevat näiden kahden luokan läpi. GraphQL-skeematiedostoon on merkitty, mitä osia CV-pankki tunnistaa ja miten niitä tulee käsitellä. Tieto liikkuu kolmessa eri muodossa CV-pankissa: JSON-muodossa, data-luokkina ja Lisäys-luokkina (input class). Data-luokat ovat muuttumattoa luokkia, joissa data esiintyy dokumenttitietokannassa. Lisäys-luokkia käytetään tiedon muuttamiseen: poisto, lisäys ja muuttaminen.

Tietoja käsittelee CurriculumVitaeDAO-luokka, joka toteuttaa CurriculumVitaeRepository-rajapinnan. Nämä yhdessä muodostavat tietokannan käsittelyyn tarvittavat operaatiot. Mitään erillistä ORM-kerrosta ei siis tarvita, vaan

MongoDB:n omat metodit riittävät ja ne tulevatkin valmiina Spring viitekehyksen MongoDB riippuvuuksista.

Kuvasta 3 voidaan huomioida, että CV-pankille ei suunniteltu montakaan käyttötapausta, jotta sovellus saataisiin nopeasti toteutettua. Toisaalta ajatus oli myös, että käyttötapauksella pitäisi pystyä tekemään ja ylläpitämään CV:itä.



KUVA 3. CV-pankin käyttötapaukset

Ensimmäinen käyttötapaus on myyjän käyttötapaus: hänen ajateltiin pystyvän lisäämään CV:itä, muuttamaan niitä, poistamaan niitä ja muuntamaan CV:itä pdf:ksi ja lähettämään niitä asiakkaille.

Asiakkaille annettiin käyttötapaukseksi vain CV-pohjan antaminen, joka lisätään järjestelmään ja tähän pohjaan voidaan muuntaa järjestelmän CV:itä.

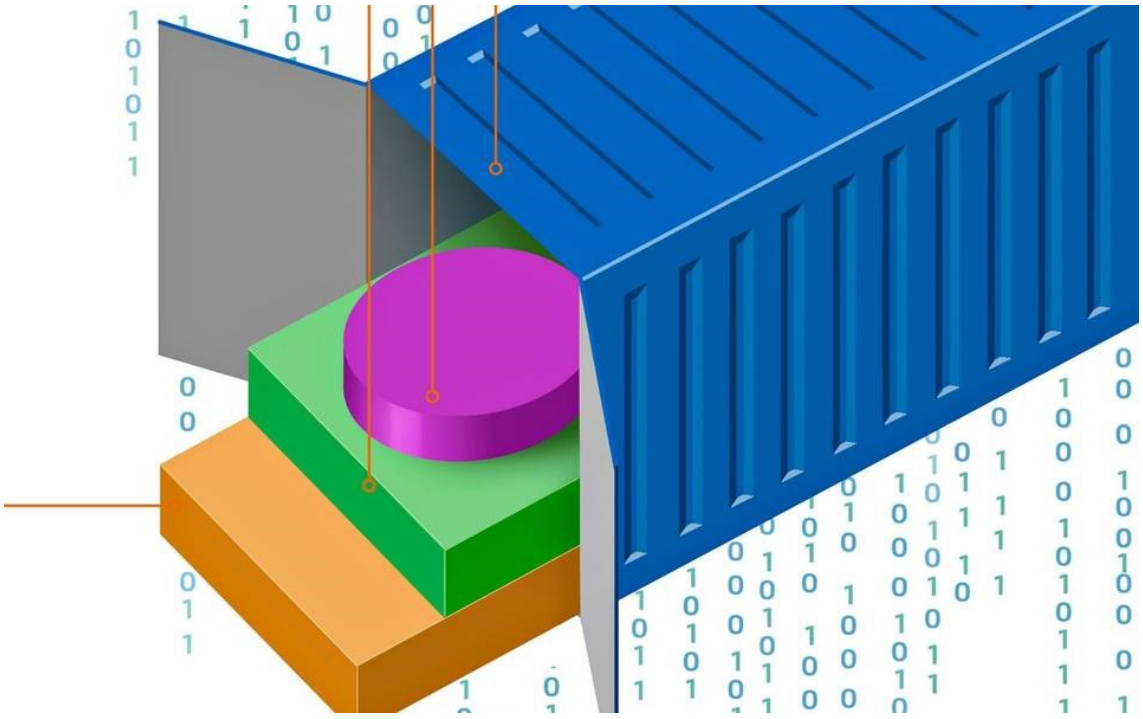
Pomon käyttötapauksia olivat CV:iden statistiikan katselu ja muuntamisoperaatiot. Analyysitarvetta ei suunniteltu mitenkään suuremmin, mutta todettiin, että HackMyResume sovelluksen työkalut CV:iden analysointiin olivat aluksi riittävät: niistä näkyy poikkeamat työkokemuksessa ja eri teknologioiden kokonaismäärät vuosina.

Työntekijälle suunniteltiin muuten samat käyttötapaukset kuin myyjälle, paitsi työntekijän ei ajateltu tarvitsevan muunnosoperaatioita.

2.2 Docker konttiteknologia

Docker on suosittu moderni konttiteknologia. Dockerin voi ajatella olevan hieman samanlainen, kuin tavaraliikenteen mullistaneen rahtikontin: kerran pakattu ja minuuteissa siirrettävä kontti tehosti kuljetuksia ympäri maailmaa. Se teki suuren mullistuksen kuljetusteknologiassa ja samalla muutti koko maailman talousjärjestelmän.

Ihan samanlaiseen pyrkii Docker. Se on ohjelmallinen kontti, jossa sovelluksia ajetaan helposti. Enää ei tarvita erillisia käyttöjärjestelmiä, vaan kaiken voi pakata näppärästi ohjelmakontin sisään. Näin sovellusten jakelu, kehittäminen ja uudistaminen nopeutuvat. Konttiteknologian ajatellaankin olevan yksi tärkeimmistä mikropalvelujen teknologioista.



KUVA 2. Ohjelmallinen kontti

Dockerin tärkeimmät hyödyt CV-pankille ovat tietenkin se, että sovellus voidaan helposti siirtää haluttuun ympäristöön ajoon. Tämän lisäksi sovelluksen skaalaus ja muutostyöt

ovat helppoja, kun kaikki tarvittava sovellukseen on samassa kontissa eikä mitään erillisiä asennuksia Docker-komentojen lisäksi tarvita.

Seuraavat hyvät puolet ja hyödyt on (2, Mitä ongelmia Dockerilla voidaan ratkaista ja mitä hyötyä siitä on?):

- ”Helppo siirrettävyys
  Kontit ovat helppoja siirrettäviä pienen kokonsa ja standardisoidun yksikkönsä vuoksi eikä siirtäminen häiritse konttien suorituskykyä tai itse konttia millään tavalla.”
- ”Ripeä jakelu
  Dockerilla on standardisoitu konttiformaatti, joten ohjelmistotiimien ei tarvitse murehtia toistensa työtehtävistä, vaan kehittäjät voivat keskittyä konttien sisällä oleviin sovelluksiin ja niiden vaatimiin muutoksiin ja operaattorit puolestaan voivat keskittyä asennuksiin toimiviksi todetuilla konteilla. Jos kontit ja niiden sisältämät paketit on testattu ja ne sisältävät kaikki tarvittavat riippuvuudet, ne toimivat kaikilla Dockerin alustoilla.”
- ”Skaalautuvuus
  Käyttäjät voivat siirrellä kontteja helposti paikasta toiseen, eli esimerkiksi pilvestä koneelle ja takaisin pilveen ilman ongelmia. Tämän lisäksi skaalaus ylös- ja alaspäin on mahdollista jopa yhdestä tuhansiin ja takaisin tuhansista yhteen.”
- “Nopeammat rakennusajat
  Kontit ovat luonnostaan erittäin pieniä ja siten rakentuvat hyvin nopeasti. Tämä mahdollistaa nopeamman testauksen, kehityksen ja käyttöönoton. Kontin rakentamisen jälkeen se voidaan lähettää suoraan testattavaksi ja sen jälkeen suoraan kaikille muille kohteille.”
- “Tiheämpi ja tehokkaampi
  Koska Dockerin kontit eivät käytä tavanomaisen virtualisoinnin hypervisoria, voidaan saatavilla olevat resurssit käyttää entistä tehokkaammin. Tämä tarkoittaa yksinkertaisesti sitä, että yhdellä koneella voi olla useampi kontti ja vaikka kone täyttyisi konteista, kevyet Docker kontit ovat silti suorituskyvyltään vähintäänkin yhdenvertaisia ja yleensä parempia kuin tavanomainen virtualisointi.”

2.3 Spring Boot mikropalveluarkkitehtuuri

Spring Boot perustuu Spring viitekehykseen, joka ei ole vain yksi teknologia, vaan pitää sisällään useamman teknologian: komponenttimalli, tietokannankäsittely ja moni muu ohjelmointiteknologia löytyy Springin sisältä.

Spring Boot sai alkunsa vuonna 2012, kun taas Spring itsessään sai alkunsa vuonna 2002, jolloin Rod Johnson kirjoitti kirjan ”Expert One-on-One J2EE Design and Development”, jossa hän kuvaili yksinkertaisemman tavan tehdä silloisin trendin mukaisia Enterprise-tason Java-sovelluksia käyttäen hyväkseen POJO-mallia (Plain Old Java mikä tarkoittaa lähinnä perinteisiä Java-luokkia), hän kuvasi muun muassa IoC-mallin (Inversion of Control) parantaakseen edellä mainittujen POJO:jen käyttöä. Syy Spring Bootin suunnittelemiseksi oli, että haluttiin arkkitehtuurimalli Springin komponenteille, jonka avulla voitaisiin konfiguroida sovellukset ylhäältä alaspäin tämä tarkoitti, että konfiguraatiot, joita oli jo olemassa Springissä, yhdistettiin ja ne muutettiin sellaisiksi, että Spring-pohjainen sovellus pystyttiin käynnistämään yksinkertaisella main()-metodilla. Tämä johti Spring Bootin suureen suosioon Java-kehittäjien joukossa ja siitä tulikin nopeimpia tapoja kehittää REST-pohjaisia mikropalveluja Javalla. Suosiota lisäsi hyvä tuki Docker konteille.

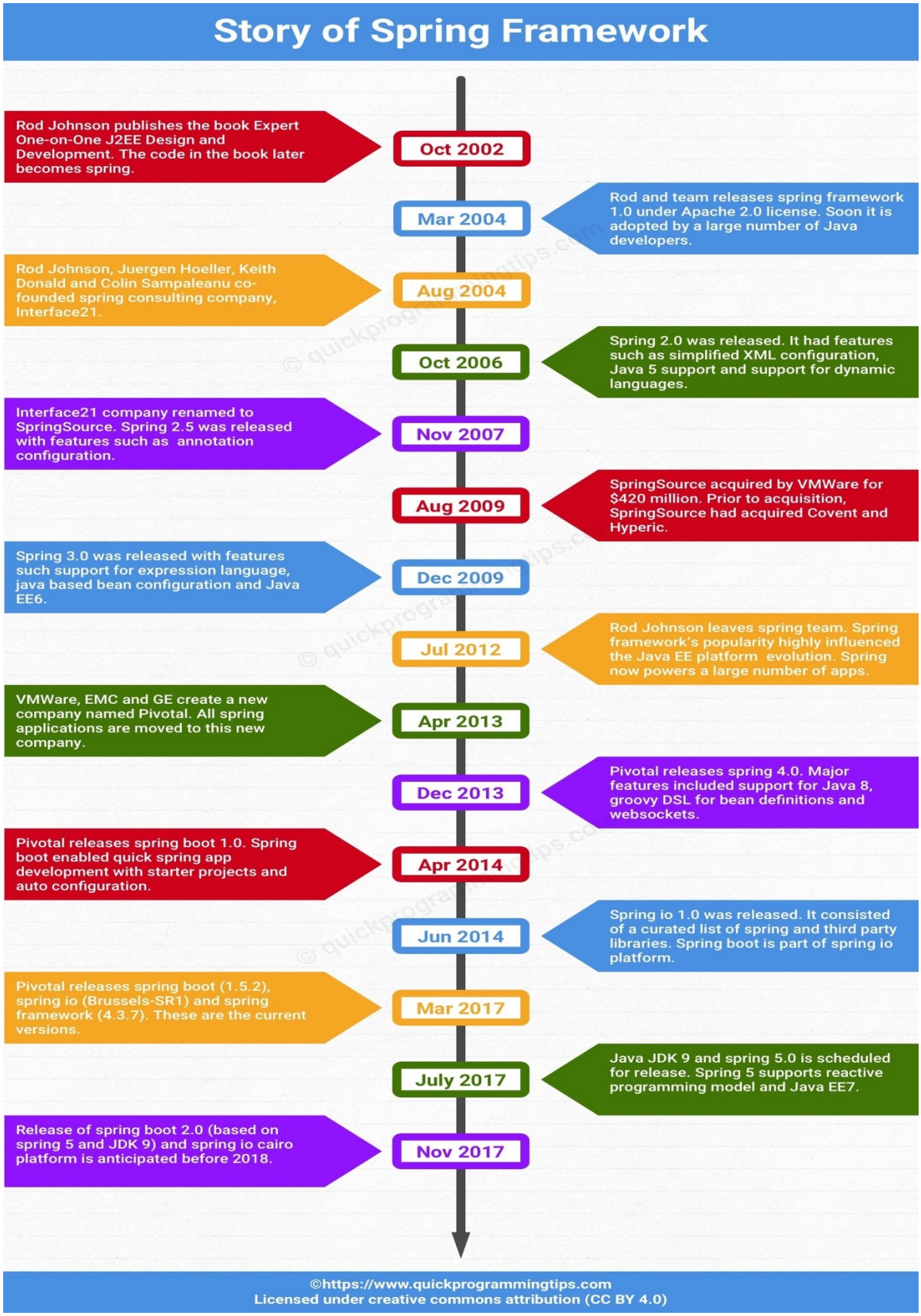
Spring Bootin ensimmäisessä versiossa oli jo kehittynyt tuki useille erilaisille Springistä eroaville teknologioille, kuten automaattinen konfigurointi Elasticsearch- ja Apache Solr-kirjastoille. Myöhemmissä versiossa teknologiatukea jatkettiin moniin nykyaikaisiin teknologioihin, kuten kakutukseen (caching). Näiden avulla pystytään rakentamaan tehokkaita moderneja mikroarkkitehtuuriin pohjautuvia sovelluksia.

Seuraava iso muutos Spring Bootiin oli Spring.io-tuen rakentaminen vuonna 2014. Tämän avulla voitiin helposti valmiiksi määriteltyjä riippuvuuksia useiden eri kirjastojen välille (mukaan lukien Springin omien kirjastojen ja kolmanten osapuolten kirjastojen välille). Tämä tarkoitti, ettei kehittäjän enää tarvinnut kuin määrittää tietty Spring io versio ja hän saisi käyttöönsä useat valmiiksi määritellyt kirjastot käyttöönsä ilman erillistä versio määrittelyä – ominaisuus, joka helpotti paljon isojen Springiin perustuvien sovellusten määrittelyä ja kehitystyön aloittamista.

Viimeisin suuri muutos Spring Bootiin on version 2.0 julkaisu vuonna 2017. Tämän suurimpia eroja edelliseen versioon oli, että 2.0 versio tarvitsi toimiakseen Javasta version 8. Muita suuria muutoksia olivat riippuvuuksien hallinta erilaisten aloittelija pakettien muodossa (starters), jotka integroivat sisäänsä pienempiä kokonaisuuksia. Näin saatiin aikaiseksi pienempi määrä riippuvuuksia, jotka tosin olivat aikaisempia suurempia. Tämän lisäksi parannettiin automaattista konfigurointia, jotta minimoitaisiin sovellukseen ajoon tarvittavat konfiguraatiot. Myös tuotantovalmiuttu parannettiin erilaisten ominaisuuksien muodossa: parempi loggaus, valvonta, metriikka ja erilaiset PaaS (Platform as a Service) integroinnit. Näiden parannusten lisäksi parannettiin itse kehitys kokemusta Spring Bootilla parempien testaustyökalujen ja palautemekanismien muodossa.

Kuvassa 3 näkyy Springin eri versioiden julkaisu ajankohta ja mitä ominaisuuksia ne toivat mukanaan.

Springin omistaa tällä hetkellä yritys nimeltään Pivotal, joka panostaa paljon Springin kehittämiseen ja ylläpitämiseen. (3, History of Spring Framework and Spring Boot)



KUVA 3. Springin historia (3)

Spring Bootin valintaan oli selkeät syyt: minä kehittäjänä osasin sitä ennestään hyvin ja se on markkinoilla edelleen erittäin kysyttyä teknologiaa. Tämän lisäksi oli vielä ajatuksena, että CV-pankki voisi toimia nuoremmille kehittäjille eräänlaisena harjoitusalustana Java-teknologiaan perehdyttäessä.

Spring Boot 2.0 on hyvä väline mikroarkkitehtuuria toteuttavan sovelluksen kehykseksi. Siinä on tarpeeksi ominaisuuksia ilman suurta määrää ylimääräisiä riippuvuuksia. Sen riippuvuuksista löytyi helposti muut sovelluksessa tarvittavat kirjastot, kuten tässä tapauksessa GraphQL ja tiedon tallentamiseen tarvittavatt MongoDB kirjastot. Lisäksi siinä oli hyvä tuki valitulla ohjelmointikielelle Kotlinilla.

2.5 Kotlin ohjelmointikieli

Kotlin ohjelmointikieli on tsekkiläisen ohjelmistoalan yrityksen Jetbrainsin kehittämä kieli ja se sai alkunsa vuonna 2011 Dmitry Jemerovin todetessa, että monissa ohjelmointikielissä ei ole hänen haluamiaan ominaisuuksia Scalaa lukuunottamatta. Hän myöhemmin totesi myös Scalan omaavan ongelmia ja rupesi kehittelemään omaan ohjelmointikieltä.

Kotlinin tavoitteina oli, että sen tuli kääntyä yhtä nopeasti kuin Java ja sen tulisi olla yhteensopiva Javan bittikoodin kanssa. Lisäksi sen haluttiin olevan olio-ohjelmointikieli, mutta sitä pitäisi voida ohjelmoida myös proseduraalisesti ja sillä olisi oltava funktionaalisen ohjelmointikielten ominaisuuksia. Näiden ominaisuuksien lisäksi Kotlinin muuttujat ja muuttujalistat muutettiin enemmän Pascal-maisiksi, siten että muuttujan nimi tulee ensin ja sitten vasta sen tyyppi. Kotlinissa voi kahdenlaisia muuttujia: muuttumattomia ja muuttuvia (immutable/mutable). Nämä erotetaan toisistaan määritteillä val (muuttumaton) ja var (muuttuva). Käytännössä tämä tarkoittaa siis sitä, että muuttuvan muuttujan arvo voi muuttua sen luomisen jälkeen, kun taas muuttumattoman arvo pysyy samana. Jälkimmäisen etuna on se, että ne ovat luonnostaan säieturvallisia (thread-safe).

Muita Kotlinin ominaisuuksia on esimerkiksi metodien jatkaminen, mikä tarkoittaa sitä että, käyttäjä pystyy lisäämään luokkaan metodeja ilman, että hänen tarvitsisi luoda uutta periytyvää luokkaa saadakseen käyttöönsä uudet metodit. Ominaisuuslista on kattava, mutta Kotlinista löytyy muun muassa seuraavat ominaisuudet: sisäkkäiset metodit

(nested methods) ja dekonstruktorimetodit, luokat ovat oletuksena viimeisteltyjä (final), akstraktit luokat ovat oletuksena taas avoimia (open), tavalliset luokat ovat oletuksena julkisia näkyvyydeltään (public).

Kotlinin etuina Javaan verrattuna on sen helppokäyttöisyys ja monipuolisempi käytettävyys, koska Kotlinia voi käyttää myös skriptikielenä. Myös monet sen ominaisuuksista ovat helpompia käyttää kuin Javan vastaavat. Esimerkiksi Android-sovellusten tuottamiseen se on jo suositumpi kuin Java.

Esimerkkinä alapuolella on yksinkertainen Kotlin ohjelma, joka tulostaa merkkijonon: ”Hei, Maailma!”.

```
// Hei, Maailma! Kotlin ohjelma

fun main (args: Array<String>) {

    val scope = ”Maailma”

    println(”Hei, $scope!”)

}
```

Esimerkki siis on vain main()-metodi, joka voi saada syötteekseen listan parametrejä (tässä esimerkissä niitä ei käytetä). Tämän metodin sisällä on muuttumaton muuttuja scope, joka saa arvokseen ”Maailma” tämä sitten riviä alempana tulostetaan merkkijonon ”Hei,” ja merkin ”!” kera näin muodostaen merkkijonon ”Hei, Maailma!”. Symboli ”$” muuttujan scope edessä kertoo Kotlinille, että viitataan johonkin muuttujaan merkkijonon sisällä.

Kotlin ohjelmointikielellä on paljon ominaisuuksia, joita Javalla ei ole. Tärkeimpiä tässä opinnäytetyössä käytettyjä ominaisuuksia on ovat data-luokat, lambdat ja annotaatiot.

Data-luokat ovat luokkia, joiden tarkoituksena on tiedon pitäminen sisällään. Kääntäjä tunnistaa data-merkkijonon luokan määrityksen edessä ja osaa antaa sille seuraavia ominaisuuksia automaattisesti:

- Equals()-ja hashCode()-funktiot

- toString()-funktio
- componentN()-funktiot
- copy()-funktio.

Equals() ja hashCode() toimivat kuten Javassa antaen luokalle olion vertailuominaisuuden (equals) ja tarkisteominaisuuden (hashCode). Näiden avulla voidaan olion tietoa vertailla tai rakentaa tarkiste sille. Funktio toString() palauttaa merkkijonona olion tiedot, esimerkki: ”Olio(ominaisuus1=1, ominaisuus2=2)”. ComponentN()-funktiot ovat Kotlinin purkumääritys funktioita (destructuring declarations), joilla olio voidaan purkaa muuttujiksi. Copy()-funktiolla oliosta luodaan kopio.

Lambdat ovat funktionaalisen ohjelmoinnin paradigmoja, jotka on lainattu lambda-kalkyylistä. Toinen nimitys näille on funktioliteraali.

Annotaatiot ovat Javasta tuttuja ja tarkoittavat metadatan liittämistä ohjelmakoodiin. Myös annotaatioiden käyttötapa on samanlainen, kuin Javassa eli nimetyn annotaation eteen lisätään merkki ”@” esimerkiksi: @Inject.

2.6 GraphQL-viestintäprotokolla

GraphQL on Facebookin suunnittelema viestintäprotokolla, joka on nykyään myös avoin standardi. GraphQL on selkeästi määritelty, joten se voi helposti keskustella minkä tahansa sitä toteuttavan sovelluksen kanssa. Tämän lisäksi GraphQL on etu perinteisempään REST:iin nähden langattomissa verkoissa, joissa jokaisella kyselyllä on merkitystä verkkoliikenteen nopeuden kanssa: GraphQL luo vain yhden ison kyselyn. Lisäksi GraphQL on järjestelmäriippumaton.

Vaikka GraphQL määritelläänkin viestintäprotokollaksi, sen avulla voidaan tehdä kyselyjä sovellukseen siten, että edustasovellus lähettää kyselyn palvelinsovellukselle, joka käsittelee ne ja palauttaa määrittelyn muotoisen tiedon takaisin kyselijälle. GraphQL:ää ei ole sidottu mihinkään tietokantajärjestelmään tai tyyppijärjestelmään, vaan se sidotaan olemassaolevaan koodiin ja dataan.

GraphQL-palvelun toteuttaminen aloitetaan määrittelemällä tyypit ja kentät näille, sitten määritellään halutut funktiot kentille. Esimerkkinä alla käyttäjä-palvelu, joka kertoo sisäänkirjautuneen käyttäjän ”minä” voidaan määritellä seuraavanlaisesti:

```
Type Query {
   Minä: Käyttäjä
}
type Käyttäjä {
  id: ID
  nimi: String
}
```

Esimerkissä on yksinkertainen kysely (Query) ja tyyppimääritys Käyttäjä. Tyyppi Käyttäjä koostuu tunnuksesta ID ja nimi-kentästä, joka on merkkijono tyyppinen (String).

Jotta Käyttäjää voidaan kysellä, tarvitaan pari funktiota, joiden avulla tämä onnistuu. Esimerkki näistä seuraavalla sivulla. Esimerkki on pseudo-JavaScriptiä, jossa osa toiminnallisuuksista on käännetty suomeksi, joten funktiot eivät sellaisenaan välttämättä toimi.

```
function Query_minä(request) {
  return request.auth.käyttäjä;
}
function Käyttäjän_nimi(käyttäjä) {
  return käyttäjä.haeKäyttäjäNimi();
}
```

Näiden funktioiden avulla saadaan Käyttäjä tyypistä itse tyyppi ja sen arvo.

GraphQL:ssä on monenlaisia tyyppejä, joiden avulla kuvataan data, kysellään sitä ja muokataan sitä. Objektityyppi kertoo minkälaisesta datasta on kyse ja mitä kenttiä sillä on. Tämän lisäksi on kysely (Query) ja mutaatio (mutation) tyypit. Kyselytyypin avulla dataan tehdään kyselyjä ja mutaatiotyypin avulla dataa muutetaan, muokataan tai poistetaan. Näiden lisäksi on skaalaarityyppi, joka kuvaa kyselyn lehtisolmut (viitataan puurakenteisen tietorakenteen alirakenteisiin solmuihin taikka lehtiin, koska tietorakenne näyttää puulta) ja enumeraatiotyyppi. Tämä viimeinen on myös skaalaarityyppi, mutta sille asetaan rajoitteeksi tietty objektityyppi. Tämän ominaisuuden avulla sillä voidaan validoida, että vain tietyn tyyppisiä objekteja sallitaan ja että kommunikaatio on aina tietyn tyyppistä. On myös olemassa listoja ja non-null tyyppejä. Listojen avulla kuvataan, että objektityyppejä voi olla useita, ja non-null kertoo, että kyseinen objektityyppi on pakko antaa. GraphQL:n tyyppijärjestelmän avulla voidaan luoda myös liittymiä (interfaces) hieman samaan tapaan kuin monessa ohjelmointikielessä, kuten Java. Tämä merkitään yleensä objektityypin type merkinnän sijaan sanalla interface. Jos halutaan toteuttaa jokin liittymä, niin se onnistuu helposti sanalla ”implements” ja mikä liittymä toteutetaan.

GraphQL:n valintaan vaikutti sen moderni tapa käsitellä tietoa ja sen helppokäyttöisyys edustajakoodaajalle (frontend), koska tarvitaan vain yksi rajapinta. Myös järjestelmäriippumattomuus oli tärkeä tekijä sen avulla voitiin tyyppimääritykset säilyttää ja haluttaessa toteuttaa järjestelmän jollain toisella ohjelmointikielellä ja viitekehyksellä, jos tälläisiä tarpeita järjestelmälle tulevaisuudessa tulee eteen.

2.7 MongoDB dokumenttikanta

MongoDB on dokumenttitietokanta, joka on skaalautuva ja joustava, mikä tarkoittaa sitä että dataa voi kysellä haluamallaan tavalla. Tämä tekee myös indeksoinnin helpoksi.

MongoDB tallentaa datan JSON-dokumenteiksi, mikä tarkoittaen sitä, että jokainen dokumentti tietokannassa voi olla erilainen ja niiden rakenne saa muuttua. Dokumenttimalli itsessään on helppo muokata sovellusten koodiin tietotyypeiksi, tehden niiden käsittelyn helpoksi. MongoDB tarjoaa indeksoinnin lisäksi myös ad hoc kyselyt, jolla dataa on helppo analysoida. MongoDB on suunniteltu hajautetuksi järjestelmäksi ja on siten helposti skaalattavissa, jonka lisäksi MongoDB omaa korkean saavutettavuuden (high availability) ja maantieteellisen siirrettävyyden. Näiden ominaisuiden lisäksi

MongoDB on avoimen lähdekoodin järjestelmä ja on siten käytännössä ilmainen käyttäjälleen, jos ei valita jotain ylläpidettyä MongoDB:tä, vaan tehdään kaikki itse.

MongoDB:n valinta CV-pankin tietokannaksi oli selkeää, koska CV:t itsessään ovat dokumentteja. Haluttiin myös tarjota mikropalveluarkkitehtuurissa määritelty atomisuuden periaatteen mukaisesti jokin sovellus, jolla dataa eli dokumentteja voidaan varastoida. MongoDB:n omaisuudet riittivät tähän upeasti ja takaavat myös mahdollisuuden skaalata CV-pankkia tarpeen tullen.

2.8 HackMyResume-sovellus

HackMyResume on node.js:llä rakennettu komentoriviltä ajettava ohjelma, joka ymmärtää FRESH-tyyppisiä CV:itä. Sovelluksen avulla voidaan luoda, muokata, analysoida, validoida ja muuntaa CV:itä. CV-pankissa sovellusta suunniteltiin käytettäväksi lähinnä muuntoon JSON:sta PDF:ksi, joka voitaisiin sitten lähettää asiakkaalle. Toki alustavia suunnitelmia oli myös analyysiominaisuuksia hyödyntämiseksi työntekijöiden koulutuksen suunnittelemisen avuksi. Tämä ratkaisu säästi aikaa, kun näitä ominaisuuksia ei tarvinnut rakentaa CV-pankkiin, vaan pystyttiin kutsumaan hackMyResumea ja annetaan sille vastuu hoitaa nämä toiminnallisuudet.

HackMyResumessa on myös ansioluettelon analysointi ominaisuuksia, joilla voidaan huomata esimerkiksi pitkät poissaolot työelämästä, tiettyjen teknologioiden osaaminen vuosina tai tiettyjen teknologiatermien esiintyminen. Analysointiominaisuuksia työn teettäjät toivoivat useissa suunnittelupalavereissa ja nämä saatiin HackMyResumessa ilman ylimääräistä ohjelmointityötä.

HackMyResumen suurin ongelma on sovelluksen rakenne: se toimii ainoastaan komentoriviltä. Toki sovellus on helppo asentaa Docker-konttiin, mutta sen käyttö sieltä käsin vaatisi hieman erilaisia toimenpiteitä, jotka päätettiin rajata pois tässä vaiheessa. Myöhemmin HackMyResumen päälle voitaisiin rakentaa oma mikropalvelumaisempi webbisovellus kloonaamalla sovellus versiohallinnasta ja aloittamalla siitä uusi avoimen lähdekoodin projekti.

2.9 FRESH-ansioluettelomalli

FRESH on avoimen lähdekoodin standardiystävällinen JSON:iin perustuva ansioluettelomalli.

Mallin lähtökohtana on ihmisen luettava tietokoneystävällinen esitysmuoto ansioluettelosta. Se pitää sisällään tiedot käyttäjän urasta, taidoista ja ansioista. Mukaan on saatu myös versiointi, jonka avulla mallin muutoksia on helppo seurata. Tietokoneystävällisyys oli tärkeää, jotta mallia olisi helppo jäsennellä erilaisilla ohjelmilla koneellisesti. Avoimuus takaa sen, että malli olisi vapaa tietyn yksinoikeudella toimivan tahon määräysvallasta – voit käyttää mitä tahansa mallia missä tahansa sovelluksessa käsitelläksesi sitä.

Malli sopi täydellisesti CV-pankkiin, koska se oli kattava ja siihen oli hyvä tuki HackMyResumessa, jotta ansioluettelon tulostus ja muunto eri formaatteihin oli helppo toteuttaa.

Kuvassa 4 on FRESH ansioluettelomallin skeema.

```
// Pared-down FRESH/FRESCA resume representation (JSON)
{
  "name": "Jane Doe",
  "info": { /* Basic info */ },
  "contact": { /* Contact information */ },
  "location": { /* Location / address */ },
  "meta": { /* Resume metadata */ },
  "employment": { /* Employment history */ },
  "projects": [ /* Project history */ ],
  "skills": [ /* Skills and technologies */ ],
  "education": { /* Schools, training, certifications */ },
  "affiliation": { /* Clubs, groups, and associations */ },
  "service": { /* Volunteer, military, civilian service */ },
  "disposition": { /* Disposition towards work, relocation, schedule */ },
  "writing": [ /* Writing, blogging, and publications */ ],
  "reading": [ /* Books and publication a la StackOverflow Careers */ ],
  "speaking": [ /* Writing, blogging, and publications */ ],
  "governance": [ /* Board memberships, committees, standards groups */ ],
  "recognition": [ /* Awards and commendations */ ],
  "samples": [ /* Work samples and portfolio pieces */ ],
  "social": [ /* Social networking & engagement */ ],
  "references": [ /* Candidate references */ ],
  "testimonials": [ /* Public candidate testimonials */ ],
  "extracurricular": [ /* Interests & hobbies */ ],
  "interests": [ /* Interests & hobbies */ ],
  "languages": [ /* languages spoken */ ]
}
```

KUVA 4. FRESH malli (4)

3 CV-PANKIN TOTEUTUS

CV-pankin toteutus alkoi erilaisilla prototyypeillä, joilla testasin eri komponenttien yhteensopivuutta ja yleensäkin niiden mahdollista käyttöä projektissa.

3.1 Esivalmistelut ja testaaminen

Ensimmäisenä ajatuksena oli tuottaa myös jonkinlainen mobiilisovellus, josta CV-pankkia voisi helposti kutsua. Tätä varten käytettiin jonkin verran aikaa Googlen flutter.io viitekehykseen tutustumiseen. Viitekehys itsessään oli mielenkiintoinen, mutta pian erilaisten testien jälkeen huomattiin sen vievän liian paljon aikaa, jotta mobiilisovellus saataisiin tehtyä järkevällä aikataululla. Lisäksi huomasin, että monet tarvittavat riippuvuudet olivat vasta kehitteillä ja voisivat toimia huonosti, mikä voisi johtaa pitkiin testaus –ja korjausvaiheisiin. Tämä puolestaan aiheutti sen, että päätettiin käyttää ainoastaan GraphQL:n mukana tulevaa webbipohjaista kehitintä (GraphiQL) käyttöliittymänä. Päätös nopeutti itse sovelluksen toteutusta paljon, kun saatiin yksi toteutettava komponentti vähemmän.

3.2 Ohjelmointikielen valinta

Seuraavaksi testattiin ja mietitiin kumpaa ohjelmointikieltä CV-pankissa tultaisiin käyttämään: Javaa vai uudempaan Javan moottorin (JVM) päällä ajettavaa Kotlinia. Ohjelmointikielen valintaa varten tehtiin selvitystyönä yksinkertainen muunnostyökalu CV-pankkiin Javalla ja se käännettiin JetBrainsin IntelliJ kehittimen työkalujen avulla Kotliniksi – operaatio, jossa ei kestänyt montaakaan minuuttia. Tämä tuotos ei tietenkään ollut kovinkaan hyvää Kotlin koodia, joten virheiden korjaamisen jälkeen käyttäen hyväksi kehittimen antamia vihjeitä ja internettiä saatiin toimivaa Kotlin koodia nopeasti. Johtopäätös oli, että kyseessä olevan uuden ohjelmointikielen oppisi tarpeeksi nopeasti, jotta sen avulla saisi määrättyyn valmistumispäivään mennessä kehitettyä sovelluksen. Prototyyppien ohella huomattiin, että Kotlin toimi myös monessa operaatiossa paremmin kuin Java: sitä oli nopeampi kirjoittaa, kuin Javaa ja se myös korjasi joitain Javan puutteita, kuten luokkamuunnokset ja helpotti funktionaalista ohjelmointia. Molemmat toiminnallisuudet olivat Kotlinissa huomattavasti selkeämmin suunniteltuja, joten ne oli helpommin omaksuttavissa.

Kieliongelman ratkaiseminen siirsi toteutukseen eri Spring versioiden valintaan ja siihen valikoitui uusin versio Spring Bootista. Kyseessä oli selkeästi aikaisempaa kehittyneempi ohjelmointikehys, jota olisi helppo ajaa Docker kontin sisällä. Spring Bootin avulla oli myös helppo rakentaa mikropalveluarkkitehtuurin toteuttava sovellus, koska kaikki määritellyt ominaisuudet voitaisiin toteuttaa sen avulla.

3.3 GraphQL CV-pankin toteutuksessa

GraphQL oli hieman vaikeampi osa toteutusta, koska siitä ei ollut aikaisempaa kokemusta. GraphQL:n nettisivuilta etsittiin erilaisia esimerkkejä siitä miten teknologiaa käytetään. Tämän lisäksi etsittiin erilaisia esimerkkejä miten sitä oli käytetty Kotlinin kanssa. Valitettavasi uuden teknologian kanssa käy helposti niin, ettei mitään hyvää esimerkkiä löydy. Tähän ongelmaan myös tässä työssä törmättiin. Tutkimuksia päätettiin jatkaa erilaisten Javalla rakennettujen versioiden muuntamisella Kotliniksi. Tämä oli hyvää kokemusta GraphQL:n käytöstä kaikkien valittujen teknologioiden kanssa. Lopulta löytyi esimerkki, jossa oli kaikki valitut teknologiat yhdessä: Spring Boot 2.0, MongoDB, Kotlin ja GraphQL. Projekti kloonattiin sen github osoitteesta ja päätettiin rakentaa CV-pankin sen päälle.

3.4 Tietomallinnus

Ensimmäisenä ongelmaksi lyhyen protoilun jälkeen nousi tietomallinnus: esimerkki projektin tietomalli oli hieman liian kevyt tarvittavan käyttäjätiedon tallentamiseen. CV-pankki piti mallintaa suhteellisen monimutkaisesta JSON-mallista. Tähän menikin runsaasti aikaa, kun kaikki JSON:in kentät tuli muuntaa Kotlinin data-luokiksi - eräänlaisiksi tyypitetyiksi luokiksi. Tämä kyseinen ominaisuus oli ominainen Kotlinille, mutta jo Javassa oli kirjastoja, joilla samankaltainen ominaisuus saatiin aikaiseksi lombok-nimistä kirjastoa hyväksi käyttäen. Ominaisuus tarkoittaa siis data-määritteen lisäämistä sanan class eteen, joilloin luokka muuttuu dataluokaksi, jolle ei tarvitse kirjoittaa erikseen datan käsittelyyn tarvittavia metodeja, vaan ne tulevat perittynä itse kielestä.

Esimerkkinä Käyttäjä dataluokka ja miten sitä käytetään:

```
data class Käyttäjä(
        val nimi: String,
        val ikä: Int
)

val käyttäjä = Käyttäjä(nimi = ”Testi Käyttäjä”, ikä = 29)
println(käyttäjä.nimi)
// tulostaa ”Testi Käyttäjä”s
```

Esimerkissä siis luodaan dataluokka nimeltään Käyttäjä ja sille annetaan parametriksi nimi ja ikä. Molemmat näistä parametreistä ovat muuttumattomia, mikä tarkoittaa, että niitä ei voi luokan instantioinnin jälkeen muuttaa. Tämä on hyvä strategia dataluokissa, jotka yleisesti tarkoitettu tiedon näyttämiseen ja mallintamiseen tietokannasta. Alemmilla rivillä luodaan uusi käyttäjä, joka on myös muuttumaton, antamalla yksinkertaisesti luokalle Käyttäjä nimen ”Testi Käyttäjä” ja iäksi 29. Sen jälkeen näytetään metodilla println(), miten tulostetaan olion Käyttäjä ominaisuus nimi.

CV-pankissa näillä dataluokilla on kaikilla yksi yhteinen kenttä: ID. Tämä kenttä kertoo MongoDB:lle, että tämä kenttä mallintaa kokoelman tunnustekentän, jolla voidaan yksilöidään kokoelmat. Dataluokissa käytetään metatietokenttiä, joiden syntaksi on lainattu Javasta: @-merkki ja metadatan (taikka annotaation) nimi. Näitä annotaatioita on runsaasti ja niillä voidaan esittää erilaista metatietoa luokasta.

MongoDB:tä käytettäessä dataluokilla on annotaatiolla @Document(collection = <kokoelman_nimi>), jolla kerrotaan MongoDB:lle, että tämä dataluokka vastaa MongoDB:n jotain kokoelmaa (esimerkissä <kokoelman_nimi>). Toinen tässä projektissa käytössä oleva annotaatio dataluokille on @Id, jolla kerrotaan jo yllä, että sen avulla tiedetään, mikä on tunnistekenttä. Muita annotaatioita ei dataluokille tarvittu, mutta näitä on paljon muitakin.

Tietomallinnuksen jälkeen piti toteuttaa tiedontallennuskerros, joka jakaantuu kahteen osaan: tietolähteeseen ja tiedonhakuobjektiin. Tietolähde kuvasi tietokantaa rajapinnan (interface) muodossa ja tiedonhakuobjekti toteutti rajapinnan, aivan kuten monessa oliosuunnittelumallissa on.

Tietolähteessä kuvattiin kyselyt ja tiedon muokkaukset, jotka sitten toteutettiin tietolähdeobjektissa. Rajapintaa olisi voinut käyttää sellaisenaan, mutta joihinkin metodeihin jouduin tekemään muutoksia, jotta se olisi toiminut haluamallani tavalla. Tiedonhakuobjektia sovelluksessa kutsui kyselyidenratkaisija (QueryResolver) ja muutostenratkaisijaluokat (MutationResolver), joiden avulla GraphQL osaa hakua tietoa ja muuttaa sitä. Ratkaisijaluokat on tärkeimpiä luokkia GraphQL:n toiminnan kannalta sillä ilman niitä GraphQL ei voisi palauttaa tietoja graafimaisina rakenteina. Se ei vain yksinkertaisesti tietäisi, mitä haun tulisi palauttaa tai missä muodossa tieto tulisi tallentaa. Näiden lisäksi GraphQL:n määritykset vaativat syöttöluokkien (Input) ohjelmoimista koska data-luokat ovat yleensä muuttumattomia, ei niillä siten voida muuttaa (mutatoida) tietoa. Syöttöluokat ovat periaatteessa vastinkappaleita data-luokille, jopa siihen asti, että ne pitävät saman tiedon sisällään, ilman ID-kenttää.

Syöttöluokkien jälkeen on palveluluokkia, joita voidaan käyttää apuna tiedonhakuobjektissa tekemään jonkin operaation. Näitä luokkia suunniteltiin ja toteutettiin kaksi kappaletta, joista toinen hoitaa sähköpostin lähettämistä ja toinen pdf-muuntoa JSON-muotoisesta CV-datasta. Näiden välillä on vahva yhteys, koska sähköpostinlähetysluokka saa alustettaessa käyttöönsä pdf-muunnosluokan Springin IoC-mallin mukaisesti. Kyseessä oleva malli tarkoittaa siis sitä, että perinteisesti new Luokka-tyyppinen alustaminen on Spring viitekehyksen hoidettavissa silloin, kun uusia luokkia tarvitaan, tai jos vanha instanssi on elossa, niin käytetään sitä. Malli tulee sanoista Inverse of Control ja Springissä se on oletuksena tuottaa singleton-tyyppisiä luokkia, joiden toiminta selitettiin edellä. IoC on erittäin käytetty tapa nykyaikaisissa viitekehyksissä ja se voidaan suorittaa niin luokille, kuin pelkästään luokan kentille. Jälkimmäinen on tehokkaampaa, mutta ensimmäinen tapa on yleisemmin käytössä, koska se on yksinkertaisempi ja esimerkiksi Kotlinia käytettäessä ohjelmointikielenä todella helppoa.

3.5 CV-pankin koodin tärkeimmät komponentit

CV-pankissa on useita luokkia ja tässä alikappaleessa kuvataan toteutuksen osalta niistä tärkeimmät sovelluksen toiminnan kannalta.

3.5.1 Dataluokat

CV-pankin ytimen luo CurriculumVitae dataluokka, jota käytetään henkilön CV-tiedon esittämiseen järjestelmässä. Luokka sinällään on yksinkertainen ja omaa vain yhden pakollisen kentän: ID. Muut kentät ovat vapaasti käytettävissä: malli ei pakota mitään näistä. Syy tähän on myös ilmeinen: voidaan luoda CV vähäisillä tiedoilla ja lisätä tietoa sitä mukaan, kuin sitä ilmentyy – esimerkiksi henkilö saa kokemusta erilaisista teknologioista työtehtävissään.

Dataluokkia on useita CV-pankissa, mutta CurriculumVitae on niistä tärkein, koska se toimii päällimmäisenä kokoelmana sisältäen toisia dataluokkia ja on ainut luokka, joka tallennetaan tietokantaan. CurriculumVitae ei Kotlinilla tehtynä ole kovinkaan iso luokka, mutta kätkee sisäänsä paljon metodeja, jotka eivät avaudu ohjelmoijalle ilman tietynlaista ymmärrystä Kotlin ohjelmointikielestä.

Kuvasta 5 huomaa, että ensimmäinen omaisuus CurriculumVitae-luokkaa tarkasteltaessa on annotaatio @Document(collection = ”curriculumVitae”). Tällä määritellään luokan olevan presentaatio MongoDB:n dokumentista. Parametri ”collection” kertoo taas sen, että luokan kokoelman nimi on MongoDB:ssä curriculumVitae. Tämä noudattaa MongoDB:n nimeämiskäytäntöjä. Tällä varmistetaan, että luokan instanssit tallentuvat aina oikeaan kokoelmaan tietokannassa.

Seuraava tärkeä ominaisuus on luokan määrittelyn etuliite data, jolla kerrotaan, että luokka on dataluokka ja omaa useita metodeja, jotka kuvataan aikaisemmin tämän opinnäytetyön kappaleessa kaksi.

Tämän jälkeen luokassa tulee suuri määrä muuttujia, joista moni on jokin toinen dataluokka, pois lukien id, jonka arvo arvotaan automaattisesti satunnaiseksi merkkijonoksi. Huomioitava seikka on myös se, että kaikki luokan muuttujat ovat muuttumattomia (etuliite val). Muuttumattomuudella pyritään takaamaan tiedon eheys, koska joka kerta tietoa käsitellessä tehdään uusi olio.

Tietoa muutetaan myöhemmin esiteltävillä syöteluokilla (input class).

```
1  package org.mmkulmala.cvbank.data
2
3  import org.springframework.data.mongodb.core.mapping.Document
4  import java.util.*
5  import javax.persistence.Id
6
7  @Document(collection = "curriculumVitae")
8  data class CurriculumVitae(
9      @Id
10     val id: String = UUID.randomUUID().toString(),
11     val name: String,
12     val meta: Meta,
13     val info: Info,
14     val contact: Contact,
15     val location: Location,
16     val projects: List<Project>,
17     val social: List<Social>,
18     val employment: Employment,
19     val education: Education,
20     val skills: Skills,
21     val samples: List<Sample>,
22     val references: List<Reference>,
23     val languages: List<Language>,
24     val interests: List<Interest>
25 )
```

KUVA 5. CurriculumVitae dataluokka

3.5.2 DAO luokka

Seuraava tärkeä luokka on CurriculumVitaeDao-luokka, joka ilmentää tietokanta operaatioita CV-pankissa. Luokka saa käänteisen kontrollin (Inverse of control) takia parametrikseen CurriculumVitaeRepository rajapinnan. Tämä rajapinta siis saadaan automaattisesti käyttöön Spring Boot viitekehyksen kirjastoista, jolloin CurriculumVitaeDao-luokka voi käyttää sen metodeja tietojen manipuloimiseen ja hakemiseen.

CurriculumVitaeDao-luokassa ei ole yhtään muuttujaa, ainoastaan metodeja. Luokalla on annotaatio @Component, joka kertoo Spring Boot viitekehykselle, että kyseinen luokka voidaan sen sisäisten mekanismien avulla instantioida automaattisesti, kun sovellusta suoritettaessa.

Luokan metodit on helppo jakaa kahteen aliryhmään: apumetodeihin ja operaatiometodeihin. Ensimmäisiä käytetään lähinnä apuna operaatiometodeissa, joita kutsutaan toisaalta CV-pankissa. Operaatiometodit jakautuvat puolestaan

hakumetodeihin ja muuttamismetodeihin. Hakumetodit tekevät hakuoperaatioita tietokantaan, kun muuttamismetodit muuttavat tietoa tietokannassa.

Hakumetodeita CurriculumVitaeDao-luokassa on seuraavia: getCurriculumVitaeById, getCurriculumVitaeByName, getCurriculumVitaeBySkills ja getFreePersonsBySkillAndTime. Metodi getCurriculumVitaeById on yksinkertaisin hakumetodi saaden parametrikseen tietokannan tunnistekentän, jonka tietoja haetaan. Metodi palauttaa tunnusta vastaavan CV:n. Metodi getCurriculumVitaeByName tekee saman operaation, kuin edellinen hakumetodi, mutta saa parametrikseen CV:n omistajan nimen. Tämän avulla on helpompi tehdä hakuja CV:eisiin. Metodi getCurriculumVitaeBySkills palauttaa jollain taito joukkiolla varustetun käyttäjän CV:n. Tämä on tärkeä metodi erilaisiin asiakastöihin henkilöstöä etsittäessä, koska yleensä ohjelmistokonsultoinnissa asiakkaat voivat antaa vain listan taidoista, joita he toivovat konsultilla olevan. Viimeinen hakumetodi on tärkeä myynnille: getFreePersonsBySkillAndTime. Tämä metodi kertoo suoritettaessa, ketä konsultti on vapaa tietyllä ajanjaksolla ja tietyllä taito yhdistelmällä. Metodi saa siis syötteekseen taitoja ja ajanjakson jolloin tämän tulisi olla saatavilla. Paluuarvona palautuu näiden perusteella ne CV:t, joilla annetut ehdot täyttävät.

Muuttamismetodeja luokassa on: createCurriculumVitae, deleteCurriculumVitae ja updateCurriculumVitae. Metodi createCurriculumVitae luo uuden CV:n tietokantaan. Kaikkia tietoja ei tarvitse luokalle heti antaa - tietoja voidaan päivittää myöhemmin. Päivitys on updateCurriculumVitae metodin vastuulla. Ja viimeinen muutosmetodi deleteCurriculumVitae poistaa tietokannasta CV:n tunnisteen perusteella.

```
@Component
class CurriculumVitaeDao(
        private val curriculumVitaeRepository: CurriculumVitaeRepository
) {
    // using finnish time format
    private fun String.toDate() : Date {
        return SimpleDateFormat( pattern: "dd.MM.yyyy").parse( source: this)
    }

    fun getCurriculumVitaeById(id: String) :Optional<CurriculumVitae!> = curriculumVitaeRepository.findById(id)

    fun getCurriculumVitaeByName(name: String) :CurriculumVitae = curriculumVitaeRepository.findByName(name)

    fun getCurriculumVitaeBySkills(skills: Skills) :List<CurriculumVitae> = curriculumVitaeRepository.findBySkills(skills)

    fun getFreePersonsBySkillAndTime(skills: Skills, free: String): List<CurriculumVitae> {
        val cvs = getCurriculumVitaeBySkills(skills)
        val cvsWithFree = ArrayList<CurriculumVitae>()

        for (cv in cvs) {
            for (proj in cv.projects) {
                if (proj.end.toDate() <= free.toDate() || proj.end.isNullOrEmpty()) {
                    cvsWithFree.add(cv)
                }
            }
        }
        return cvsWithFree
    }
```

KUVA 6. CurriculumVitaeDao-luokka ja haku metodit

```
    fun createCurriculumVitae(name: String, meta: Meta, info: Info, contact: Contact, location: Location, projects: List<Project>,
                              social: List<Social>, employment: Employment, education: Education, skills: Skills, samples: List<Sample>,
                              references: List<Reference>, languages: List<Language>, interests: List<Interest>) :CurriculumVitae =
            curriculumVitaeRepository.save(CurriculumVitae(name = name, meta = meta, info = info, contact = contact, location = location, projects = projects,
                    social = social, employment = employment, education = education, skills = skills, samples = samples,
                    references = references, languages = languages, interests = interests))

    fun deleteCurriculumVitaeById(id: String) :Unit = curriculumVitaeRepository.deleteById(id)

    fun updateCurriculumVitae(id: String, updatedCV: CurriculumVitaeInput) :CurriculumVitae = curriculumVitaeRepository.save(
            CurriculumVitae(id = id, name = updatedCV.name,
                    meta = Meta(format = updatedCV.meta.format, version = updatedCV.meta.version),
                    info = Info(label = updatedCV.info.label, characterClass = updatedCV.info.characterClass, brief = updatedCV.info.brief,
                            image = updatedCV.info.image, quote = updatedCV.info.quote),
                    contact = Contact(website = updatedCV.contact.website, phone = updatedCV.contact.phone, email = updatedCV.contact.email,
                            other = updatedCV.contact.other.map { Other(label = it.label, flavor = it.label, value = it.value) }),
                    location = Location(address = updatedCV.location.address, city = updatedCV.location.city, region = updatedCV.location.region,
                            code = updatedCV.location.code, countryCode = updatedCV.location.countryCode),
                    projects = updatedCV.projects.map { Project(title = it.title, category = it.category, role = it.role, url = it.url, start = it.start, end = it.end,
                            repo = it.repo, description = it.description, summary = it.summary, keywords = it.keywords,
                            media = it.media.map { Media(category = it.category, url = it.url) }) },
                    social = updatedCV.social.map { Social(label = it.label,network = it.network,user = it.user,url = it.url) },
                    employment = Employment(summary = updatedCV.employment.summary, history = updatedCV.employment.history.map { History(employer = it.employer, url = it.url,
                            position = it.position, summary = it.summary, start = it.start, end = it.end, keywords = it.keywords, highlights = it.highlights)}),
                    education = Education(summary = updatedCV.education.summary, level = updatedCV.education.level, degree = updatedCV.education.degree,
                            history = updatedCV.education.history.map { DegreeHistory(institution = it.institution, title = it.title, url = it.url, start = it.start,
                                    end = it.end, grade = it.grade, summary = it.summary, curriculum = it.curriculum) }),
                    skills = Skills(sets = updatedCV.skills.sets.map { org.mmkulmala.cvbank.data.Set(name = it.name, level = it.level, skills = it.skills) },
                            list = updatedCV.skills.list.map { Skill(name = it.name, summary = it.summary, level = it.level, years = it.years, proof = it.proof) }),
                    samples = updatedCV.samples.map { Sample(title = it.title,summary = it.summary,url = it.url,date = it.date) },
                    references = updatedCV.references.map { Reference(name = it.name, flavor = it.flavor, private = it.private,
                            contact = it.contact.map { ReferenceContact(label = it.label, flavor = it.flavor, value = it.value) }) },
                    languages = updatedCV.languages.map { Language(language = it.language, level = it.level) },
                    interests = updatedCV.interests.map { Interest(name = it.name, summary = it.summary, keywords = it.keywords) }))
```

KUVA 7. CurriculumVitaeDao-luokan tiedonmuuttamis metodit

3.5.3 Ohjausluokat

Seuraavana käydään läpi CV-pankin toimintaan ohjaavat luokat: CurriculumVitaeQueryResolver ja CurriculumVitaeMutationResolver.

CurriculumVitaeQueryResolver ohjaa GraphQL:n kautta tulleet kyselyt CurriculumVitaeDao-luokalle. Luokka saa ohjeet operaatioon skeemalta, jossa määritellään GraphQL-kyselyt, joita CV-pankissa käytetään. Luokka saa jo muista luokista tutun annotaation @Component ja parametrikseen se saa CurriculumVitaeDao:n (joka instantioidaan käyttöön Spring Boot viitekehyksen toimesta tarvittaessa). Tämän luokan metodit ovat vastinpareja CurriculumVitaeDao-luokan hakumetodeille. Metodit ovat: curriculumVitae, curriculumVitaeByName, curriculumVitaeBySkills ja curriculumVitaeByFreeStatusAndSkills.

```
package org.mmkulmala.cvbank.graphql.query

import com.coxautodev.graphql.tools.GraphQLQueryResolver
import org.mmkulmala.cvbank.dao.CurriculumVitaeDao
import org.mmkulmala.cvbank.data.Set
import org.mmkulmala.cvbank.data.Skill
import org.mmkulmala.cvbank.data.Skills
import org.mmkulmala.cvbank.graphql.input.SkillsInput
import org.springframework.stereotype.Component
import java.util.*

/**
 * Created by marno kulmala on 13/09/2018.
 */
@Component
class CurriculumVitaeQueryResolver(
        private val curriculumVitaeDao: CurriculumVitaeDao
): GraphQLQueryResolver {
    fun curriculumVitae(id: String) :Optional<CurriculumVitae!> = curriculumVitaeDao.getCurriculumVitaeById(id)

    fun curriculumVitaeByName(name: String) :CurriculumVitae = curriculumVitaeDao.getCurriculumVitaeByName(name)

    fun curriculumVitaesBySkills(skills: SkillsInput) :List<CurriculumVitae> =
            curriculumVitaeDao.getCurriculumVitaeBySkills(
            Skills(sets = skills.sets.map { Set(name = it.name, level = it.level, skills = it.skills) },
                    list = skills.list.map { Skill(name = it.name, summary = it.summary,
                            level = it.level, years = it.years, proof = it.proof) })
    )

    fun curriculumVitaesByFreeStatusAndSkills(skills: SkillsInput, free: String) :List<CurriculumVitae> =
            curriculumVitaeDao.getFreePersonsBySkillAndTime(
            Skills(sets = skills.sets.map { Set(name = it.name, level = it.level, skills = it.skills) },
                    list = skills.list.map { Skill(name = it.name, summary = it.summary,
                            level = it.level, years = it.years, proof = it.proof) }), free = free
    )
}
```

KUVA 8. CurriculumVitaeQueryResolver-luokka

GraphQL:n kautta tulleet kyselyt voivat myös olla luonteeltaan tietoa muuttavia ja niiden toimintaa ohjaa CurriculumVitaeMutationResolver-luokka. Ja kuten kyselyjä ohjaava luokka, niin myös tämäkin luokka kytkeytyy CurriculumVitaeDao-luokkaan, mutta vain niiden pyyntöjen osalta, joilla muutetaan tietoja CV-pankissa. Luokassa on tuttu annotaatio @Component ja CurriculumVitaeDao parametrina. Metodeja on kolme: createCurriculumVitae, deleteCurriculumVitae ja updateCurriculumVitae. Metodit eroavat vain parametriensa ja nimiensä osalta CurriculumVitaeDao-luokan muutosmetodeista.

Parametriksi luokan metodit saavat syöteluokkia, jotka sitten muunnetaan dataluokiksi. Syöteluokista lisää myöhemmin.

```
@Component
class CurriculumVitaeMutationResolver(
        private val curriculumVitaeDao: CurriculumVitaeDao
) : GraphQLMutationResolver {
    fun createCurriculumVitae(input: CurriculumVitaeInput) :CurriculumVitae =
            curriculumVitaeDao.createCurriculumVitae(
                    input.name,
                    Meta(format = input.meta.format, version = input.meta.version),
                    Info(label = input.info.label, characterClass = input.info.characterClass, brief = input.info.brief,
                            image = input.info.image, quote = input.info.quote),
                    Contact(website = input.contact.website, phone = input.contact.phone, email = input.contact.email,
                            other = input.contact.other.map { Other(label = it.label, flavor = it.label, value = it.value) }),
                    Location(address = input.location.address, city = input.location.city, region = input.location.region,
                            code = input.location.code, countryCode = input.location.countryCode),
                    input.projects.map { Project(title = it.title, category = it.category, role = it.role, url = it.url, start = it.start, end = it.end,
                            repo = it.repo, description = it.description, summary = it.summary, keywords = it.keywords,
                            media = it.media.map { Media(category = it.category, url = it.url) }) },
                    input.social.map { Social(label = it.label,network = it.network,user = it.user,url = it.url) },
                    Employment(summary = input.employment.summary, history = input.employment.history.map { History(employer = it.employer, url = it.url,
                            position = it.position, summary = it.summary, start = it.start, end = it.end, keywords = it.keywords, highlights = it.highlights)}),
                    Education(summary = input.education.summary, level = input.education.level, degree = input.education.degree,
                            history = input.education.history.map { DegreeHistory(institution = it.institution, title = it.title, url = it.url, start = it.start,
                                    end = it.end, grade = it.grade, summary = it.summary, curriculum = it.curriculum) }),
                    Skills(sets = input.skills.sets.map { Set(name = it.name, level = it.level, skills = it.skills) },
                            list = input.skills.list.map { Skill(name = it.name, summary = it.summary, level = it.level, years = it.years, proof = it.proof) }),
                    input.samples.map { Sample(title = it.title,summary = it.summary,url = it.url,date = it.date) },
                    input.references.map { Reference(name = it.name, flavor = it.flavor, private = it.private,
                            contact = it.contact.map { ReferenceContact(label = it.label, flavor = it.flavor, value = it.value) }) },
                    input.languages.map { Language(language = it.language, level = it.level) },
                    input.interests.map { Interest(name = it.name, summary = it.summary, keywords = it.keywords) }
            )

    fun deleteCurriculumVitae(id: String) :Unit = curriculumVitaeDao.deleteCurriculumVitaeById(id)

    fun updateCurriculumVitae(id: String, input: CurriculumVitaeInput) :CurriculumVitae = curriculumVitaeDao.updateCurriculumVitae(id, input)
}
```

KUVA 9. CurriculumVitaeMutationResolver-luokka ja metodit

Repository-rajapinta on tärkeä, vaikka se on kooltaan pieni. Sitä tarvitaan, jotta CV-pankki voi helposti käsitellä tietoa tietokannassa. Suurin osa rajapinnan metodeista tulee peritystä MongoRepository-rajapinnasta, jolle kerrotaan tyyppimääritteiksi CurriculumVitae-dataluokka ja merkkijono. Tämä tarkoittaa sitä, että halutaan MongoDB:n pitävän sisällään CurriculumVitae kokoelmia ja näiden tunnuskenttä on merkkijono. Luokka saa @Repository annotaation, jonka avulla Spring Boot tietää, että ko. luokka on tietolähde pitäen sisällään tiedot tietokantayhteydestä.

CurriculumVitaeRepository-rajapinnassa määritellään vain kaksi metodia haku toiminnoilla, joita ei löydy suoraan MongoRepository-rajapinnasta.

```
@Repository
interface CurriculumVitaeRepository : MongoRepository<CurriculumVitae, String> {
    fun findByName(name: String): CurriculumVitae

    fun findBySkills(skills: Skills): List<CurriculumVitae>
}
```

KUVA 10. CurriculumVitaeRepository-luokka

3.5.4 Palvelut

CV-pankissa on myös palveluluokkia, joita hyödynnetään sähköpostin lähettämisessä ja CV:n muuntamisessa JSON:sta PDF:ksi.

Ensimmäinen palvelu-luokka on EmailService. Tämän luokan tehtävä on lähettää sähköpostiviestinä halutulle käyttäjälle mahdollisen konsulttikandidaatin CV PDF-muotoisena. Tällä hetkellä tämä on testattu vain ainoastaan Googlen Gmailin kautta, mutta periaatteessa mikä tahansa sähköpostijärjestelmä kävisi operaatioon. Luokka on komponentti-tyyppinen (annotaatio), joten Spring-viitekehys löytää sen helposti ja automatiikan avulla luo siitä tarvittaessa instanssin. Parametrikseen luokka saa ominaisuus-tiedoston (properties) EmailProperties ja ResumeCreationService-palvelun. Ensimmäisestä parametrista saadaan sähköpostia koskevat ominaisuudet ja toisesta saadaan PDF-muunnokseen tarvittava palvelu. EmailService-palvelulla on yksi metodi: sendCVAsAttachment. Tämä metodi hoitaa sähköpostilähetyksen ja PDF-muunnoksen kutsumiset. Parametriksi metodi saa lähettäjän sähköpostiosoitteen, tämän salasanan, kenelle sähköposti lähetetään ja minkä niminen CV halutaan lähettää.

```
@Component
class EmailService(
        private val emailProperties: EmailProperties,
        private val resumeCreationService: ResumeCreationService
) {

    fun sendCVAsAttachment(senderEmail : String, password : String,
                           toMail : String, resumeName: String) {
        val email = MultiPartEmail()
        email.hostName = emailProperties.hostname
        email.setSmtpPort(emailProperties.smtpport.toInt())
        email.setAuthenticator(DefaultAuthenticator(senderEmail, password))
        email.isSSLOnConnect = true
        email.setFrom(senderEmail)
        email.addTo(toMail)
        resumeCreationService.createPDF(resumeName)
        email.attach(File( pathname: "output/$resumeName.pdf"))
        email.subject = emailProperties.subject + "$resumeName"
        email.send()
    }
}
```

KUVA 11. EmailService palvelu

ResumeCreationService on Spring-komponentti (@Component annotaatio), joka saa parametriksi MongoDB:n ominaisuus-tiedoston. Tätä käsitellään sovelluksessa palveluna.

Ominaisuus-tiedostosta saadaan tarvittavat tiedot, jotta tietokantayhteys MongoDB:hen onnistuu: yhdistämisportti ja tietokannan nimi. Metodeja palvelulla on yksi julkinen (public) ja kaksi yksityistä (private). Näistä käyttäjä pystyy kutsumaan vain julkista metodia. CreatePDF-metodi on julkinen ja tekee parametrina annetun nimisestä CV:stä PDF:n. Tämä tapahtuu ottamalla MongoDB:hen yhteys annetuilla tiedoilla. Tämän jälkeen suoritetaan haku CV:n nimellä ja tallennetaan tiedosto väliaikaisesti fyysisesti kansioon, josta se voidaan lisätä sähköpostin liitteeksi.

Kaksi yksityistä metodia ovat apumetodeja julkiselle metodille. Toinen yksityinen metodi tallentaa tiedostoja ja toinen osaa suorittaa järjestelmän muita ohjelmia, jota tarvitaan HackMyResume-sovelluksen suorittamiseen. Tämä on samanlainen operaatio kuin Javassa: käynnistetään vain uusi nimetty prosessi. Ratkaisu on tässä tapauksessa hieman kehittyneempi, koska Kotlinin ominaisuuksiin kuuluu kyky ylikirjoittaa String valmisluokan ominaisuuksia: CV-pankissa merkkijonoille tässä tapauksessa lisätään kyky ajaa komento merkkijono-muotoisena.

```
@Component
class ResumeCreationService(
        private val mongoProperties: MongoProperties
) {

    fun createPDF(resumeName: String): Boolean {
        //get com.mongodb.MongoClient new instance
        val client = KMongo.createClient()
        //normal java driver usage
        val database = client.getDatabase(mongoProperties.mongodbDatabase)
        //KMongo extension method
        val col = database.getCollection("curriculumVitae", CurriculumVitae::class.java)
        // search for right resume
        val cv : CurriculumVitae? = col.findOne( filter: CurriculumVitae::name eq resumeName)
        // write it to output folder for hackmyresume
        writeToFile(cv.toString(),  fileName: "output/$resumeName.json")

        "hackmyresume BUILD output/$resumeName.json TO output/$resumeName.pdf -t modern".runCommand()
        return File( pathname: "output/$resumeName.pdf").exists()
    }

    private fun writeToFile(content: String, fileName: String) {
        val writer = PrintWriter(fileName)
        writer.append(content)
        writer.close()
    }

    private fun String.runCommand(workingDir: File? = null) {
        val process = ProcessBuilder(*split( ...delimiters: " ").toTypedArray())
                .directory(workingDir)
                .redirectOutput(Redirect.INHERIT)
                .redirectError(Redirect.INHERIT)
                .start()
        if (!process.waitFor( timeout: 10, TimeUnit.SECONDS)) {
            process.destroy()
            throw RuntimeException("execution timed out: $this")
        }
        if (process.exitValue() != 0) {
            throw RuntimeException("execution failed with code ${process.exitValue()}: $this")
        }
    }
}
```

KUVA 12. ResumeCreationService palvelu

3.5.5 Lisäys-luokat

Palveluiden, resolvereiden (tehtävä on päätellä annetuille luokille näiden tietoja käsittelevä luokka) ja data-luokkien lisäksi CV-pankki tarvitsee lisäys-luokkia, joiden avulla sovelluksen tietoja voidaan muuttaa.

Lisäys-luokat on samankaltaisia, kuin data-luokat, mutta niillä ei ole tunnuskenttää, koska lisäys-luokkien muuttujien arvoista yleensä luodaan uusia data-luokkia, jotka sitten tallentaan tietokantaan. Tämä tarkoittaa myös sitä, että vaikka puhutaan lisäys-luokista, niin niidenkään dataa ei luomisen jälkeen muuteta. Vaan voisinkin paremminkin puhua tiedonsiirtoluokista. Tällä tavoin varmistetaan tiedoneheys ja voidaan toteuttaa järjestelmä niin, että data-luokat kuvaavat tietokannat taulut, taikka tässä tapauksessa kokoelmat. Niitä ei siis käytetä muuhun.

```
data class CurriculumVitaeInput(
        val name: String,
        val meta: MetaInput,
        val info: InfoInput,
        val contact: ContactInput,
        val location: LocationInput,
        val projects: List<ProjectInput>,
        val social: List<SocialInput>,
        val employment: EmploymentInput,
        val education: EducationInput,
        val skills: SkillsInput,
        val samples: List<SampleInput>,
        val references: List<ReferenceInput>,
        val languages: List<LanguageInput>,
        val interests: List<InterestInput>
)
```

KUVA 14. CurriculumVitateInput lisäys-luokka

3.5.6 CV-pankki sovellus

Tämän jälkeen jää jäljelle enää CVBankApplication, joka on sovelluksen suorittamiseen tarvittava luokka. Tässä luokassa ei ole mitään muuta erikoista – se vain kutsuu Springin SpringApplication.run-metodia käynnistääkseen CV-pankin. Spring tietää, että kyseessä on käynnistys-luokka, koska luokalle annetaan annotaatio @SpringApplication. Tämän voisi perinteisen ohjelmoinnin termein käsittää olevan itse sovellus (tai applikaatio), mutta Springin terminologiassa tämä on se luokka, jonka avulla sovellus suoritetaan. Springin omat komponentit huolehtivat määritysten mukaisesti mitä suoritettavaan sovellukseen tulee mukaan.

```
@SpringBootApplication
class CVBankApplication

fun main(args: Array<String>) {
    SpringApplication.run(CVBankApplication::class.java, *args)
}
```

KUVA 13. CVBankApplication-luokka

3.5.7 GraphQL

CV-pankki tarvitsee toimiakseen vielä GraphQL-määritteet ja HackMyResume sovelluksen. GraphQL:lle tehtiin määritystiedosto, jonka avulla saavutetaan loistava siirrettävyys tiedoston toimiessa liikuteltavan tiedon kuvauksena: pystytään ottamaan ohjelmointikieli X ja toteuttamaan CV-pankki sillä kopioimalla vain GraphQL:n määritystiedosto uuteen sovellukseen.

GraphQL määritys alkaa kertomalla mitä määritellään. Näin myös CV-pankissa.

```
schema {
    query: Query
    mutation: Mutation
}
```

KUVA 15. GraphQL:n määritystiedoston alku

CV-pankin GraphQL määritystiedostossa ensimmäisenä kerrotaan, että kyseessä on skeema ja annetaan sillä kyselytyypit ja muuttamistyypit. Tämän jälkeen kuvataan lisäyskentät.

```
input CurriculumVitaeInput {
    name: String
    meta: MetaInput
    info: InfoInput
    contact: ContactInput
    location: LocationInput
    projects: [ProjectInput]
    social: [SocialInput]
    employment: EmploymentInput
    education: EducationInput
    skills: SkillsInput
    samples: [SampleInput]
    references: [ReferenceInput]
    languages: [LanguageInput]
    interests: [InterestInput]
}
```

KUVA 16. Lisäyskenttä GraphQL:n määritystiedostossa

Tämän jälkeen annetaan GraphQL:n tyypit (vastaavat data-luokkia). Tämä periaatteesssa tuplamääritys on pakko tehdä, jotta myös GraphQL tietäisi, mitä tyyppejä on mahdollista käyttää. Ja myös siksi, ettei Spring tiedä taas mihin data-luokkaan tyyppi sitoutuu, joille resolveri kerro sille sitä. Kuvasta 17 havaitsee hyvin GraphQL:n tyyppijärjestelmän: on yksittäisiä tyyppejä ja niitä voi olla myös useita listassa. Lista merkitaan ”[ ]”-merkeillä. Huutomerkki taasen kertoo, että jokin arvo on pakollinen.

```
type CurriculumVitae {
    id: ID!
    name: String
    meta: Meta
    info: Info
    contact: Contact
    location: Location
    projects: [Project]
    social: [Social]
    employment: Employment
    education: Education
    skills: Skills
    samples: [Sample]
    references: [Reference]
    languages: [Language]
    interests: [Interest]
}
```

KUVA 17. GraphQL:n tyyppimääritys

CV-pankissa ei ole määritelty data tyypeille mitään muuta pakolliseksi arvoksi, kuin tunnuskenttää ja tämäkin vain sen takia, että tiedon eheys tietokannassa säilyy (cv-dokumentit yksilöityvät).

GraphQL:lle voidaan määritellä tyypiksi myös Queryn (eli kyselyn) ja Mutationin (muuttamiseen tarvittavat tyypit). CV-pankissa Query-tyyppejä on viisi, yksi enemmän kuin niihin liitettyjä metodeja – viides on versio kysely, jolla varmistetaan määritteiden versionumero. Mutation-tyyppejä on kolme ja ne liittyvät kolmeen muutosoperaatioita hoitaviin metodeihin. Kysely tyypeillä ja muutos tyypeillä on CV-pankissa enemmän pakolllisia kenttiä: kaikki kyselyt ja muutettavat tiedot on pakko antaa parametrina. Ilman tätä määritystä sovellus ei toimisi.

```
# All query endpoints available for CurriculumVitae bank
type Query {

    # The API Version
    version: String

    # Get curriculum vitae with skills
    curriculumVitaesBySkills(skills: SkillsInput!): [CurriculumVitae]

    # Get curriculum vitae with skills and if person is free
    curriculumVitaesByFreeStatusAndSkills(skills: SkillsInput!, free: String): [CurriculumVitae]

    # Get curriculum vitae with ID
    curriculumVitae(id: ID!): CurriculumVitae

    # Get currciculum vitae with name
    curriculumVitaeByName(name: String!): CurriculumVitae
}
```

KUVA 18. Kysely tyyppi CV-pankin GraphQL määritystiedostossa

```
# All data change endpoints for cvbank. You can use createCurriculumVitae to create whole CurriculumVitae collection.
type Mutation {

    # Create a new CurriculumVitae collection
    createCurriculumVitae(input: CurriculumVitaeInput!): CurriculumVitae!

    # Create a new CurriculumVitae collection
    updateCurriculumVitae(id: ID!, input: CurriculumVitaeInput!): CurriculumVitae!

    # Delete CurriculumVitae collection
    deleteCurriculumVitae(id: ID!): String
}
```

KUVA 19. Muuttamistyyppi CV-pankin GraphQL määritystiedostossa

3.6 Muut CV-pankin toimintaan vaikuttavat komponentit

Edellä käytiin läpi tärkeimmät CV-pankin lähdekoodin osat, mutta sovellus tarvitsee lisäksi vielä pari komponenttia: Docker konttialustan ja HackMyResume node.js sovelluksen.

3.6.1 Docker

CV-pankin toteutukseen otettiin mahdollisimman yksinkertainen kontti ratkaisu. Ja täksi varmistui miltei heti Docker, koska tästä oli eniten kokemusta.

Dockeria voidaan käyttää monella tavalla, mutta tavallisin on Dockerfile nimisen tiedoston sijoittaminen sovelluksen projektihakemistoon. Tähän tiedostoon voidaan helposti lisätä halutut komponentit, jotka halutaan konttiin. Ensimmäinen CV-pankinkin tapauksessa on mistä peruspaketista kontti halutaan rakentaa – CV-pankin tapauksessa tämä on mahdollisimman pieni Debian linuxin versio. Seuraavaksi voidaan määritellä mahdolliset levyvoluumit, mikä tehtiin myös CV-pankissa (tmp väliaikais hakemisto). Tämän jälkeen CV-pankki sovellus kopioidaan sen projekti hakemistosta konttiin sisään.

```
FROM re6exp/debian-jessie:latest
VOLUME /tmp
ADD target/cvbank-0.0.1-SNAPSHOT.jar cvbank.jar
```

KUVA 20. CV-pankin Dockerfile tiedoston alku

Tämän jälkeen suoritetaan Javan asennus, jotta CV-pankkia voidaan ajaa kontista. Javan asennuksen jälkeen suoritetaan node.js:n asennus, joka on vaatimuksena HackMyResume:n asentamisella (node.js toimii sen moottorina). Viimeisenä määritellään CV-pankki tulokohdaksi (entrypoint) konttiin; mikä periaatteessa tarkoittaa, että kontin ulkopuolelta ei voida suorittaa mitään muuta, kuin portissa 8080 toimiva CV-pankki.

3.6.2 HackMyResume

HackMyResume oli oikeastaan alkusysäys CV-pankin suunnittelemiseksi ja toteuttamiseksi. Sillä suoritetut kokeilut johtivat useisiin muutoksiin lopullista sovellusta toteutettaessa: esimerkiksi JSR JSON malli muuttui FRESH malliksi, koska FRESH malli esitti CV:n paremmin ja se oli selkeämpi.

HackMyResumella voidaan luoda viimeisteltyjä CV:itä ja muuntaa ne useisiin eri muotoihin (PDF, word jne) käyttäen komentoriviä. Sen lisäksi sillä voidaan tehdä analyysia CV:eille esimerkiksi siten, että etsitään kaikki taidot, jotka sen omistajalla on. CV-pankissa on ajateltiin aluksi käytettävän vain muunnosominaisuutta analysointiominaisuuden toimiessa lähinnä bonuksena.

Docker on läheisessä suhteessa HackMyResumeen CV-pankissa, koska se toimii HackMyResumen suorituspaikkana. CV-pankki hoitaa HackMyResumen suorittamisen

automaattisesti sitä tarvittaessa, mutta tällä hetkellä analysointiominaisuuden (ja muut HackMyResumen) saa käyttöönsä vain kirjautumalla ssh:n avulla suoraan konttiin. Ominaisuuden suunnittelu jäi ajan puutteen vuoksi vähemmälle huomiolle, eikä sitä käsitellä tässä työssä sen enempää.

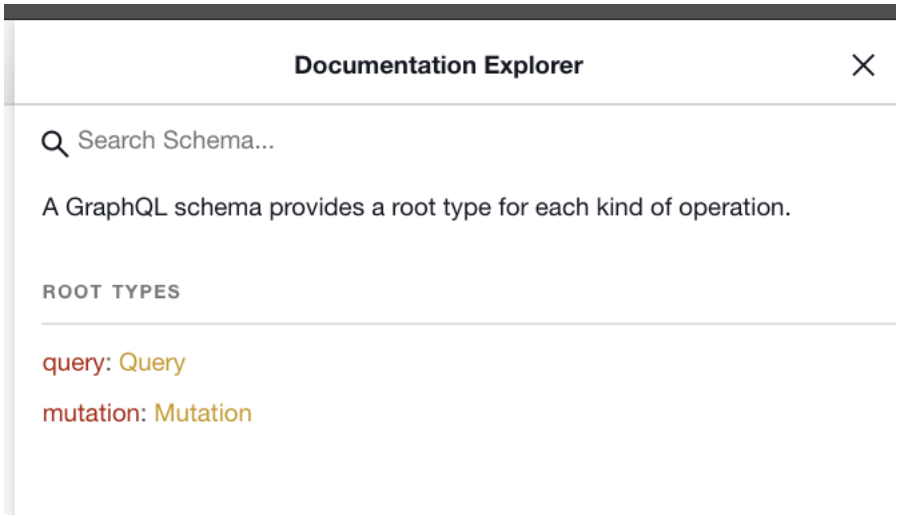
4 CV-PANKIN TESTAUS

CV-pankkia testattiin lähinnä sen ainoan kehittäjän toimesta. Testaus oli jaettu kahteen eri osaan: yksikkötestit ja toiminnallinen testaus. Yksikkötestejä ei ole montaa, mutta ne suunniteltiin lähinnä auttamaan sovelluksen kehittämisessä silloin, kuin jotain uutta toiminnallisuutta toteutettiin. Tähän tarkoitukseen koko sovelluksen sisällön testaava yksikkötesti oli erittäin hyvä, koska sitä ajetaan aina, kun sovellus käännetään. Tämän testin epäonnistuminen tarkoitti yleensä sitä, että jokin muutos ei ollut oikea ja se rikkoi jotain. Paras ominaisuus testissä oli, että se ajoi aina GraphQL-skeemamuunnoksen, joka tarkisti, että skeemassa määritellyille asioille löytyy konkreettinen luokka, ja jonkin puuttuessa se ilmoitti siitä epäonnistumisella.

Itse testaus tapahtui manuaalisesti GraphiQL integroidun kehittimen kautta. Yleisimmät tapaukset testauksessa olivat uuden CV:n luonti, CV:n päivittäminen, CV:n poisto, CV:n lähettäminen sähköpostitse PDF:nä. Tämän lisäksi testattiin erikoistapauksia: hae tietyillä taidoilla CV:tä ja hae kaikki CV:t, joissa on tietty taito ja joissa henkilö on vapaana. Jälkimmäisissä huomattiin poikkeamia, joita ei vielä ehditty korjaamaan, mutta niistä luotiin asianmukaiset testihavainnot. Tämän lisäksi yleiset testitapaukset kirjattiin README.md-tiedostoon, jotta ketä tahansa voi toistaa testit.

Näiden lisäksi testattiin, kuinka hyvin sovellus kestää päällä oloa useita tunteja. Ominaisuus on tärkeä mahdollista tuotantoympäristö asennusta silmällä pitäen. Vielä tärkeämmäksi tämä testaus tulee, jos sovellus toimii pilvipalveluna jossain julkisessa pilviympäristössä. Tällä testauksella pyrittiin selvittämään, että vaikka sovellus on koko ajan toiminnassa pilvessä, niin se ei kaadu omia aikojaan.

Testauksessa ei haettu kaikkia CV-pankin kokoelmien kenttiä, vaan haluttiin tietää yleensä tärkeimmät, joista tunnus ja nimi olivat käytetyimmät. Tosin kyselyistä voidaan palauttaa mitä tahansa kokoelman kenttiä. Nämä saa helposti tietoonsa GraphiQL:n dokumentoinnista, joka voidaan luoda näppärästi skeematiedostoon. Tällöin kaikkia kyselyitä ja muutoksia ei tarvitse muistaa ulkoa.



KUVA 21. GraphiQL:n dokumentaatio

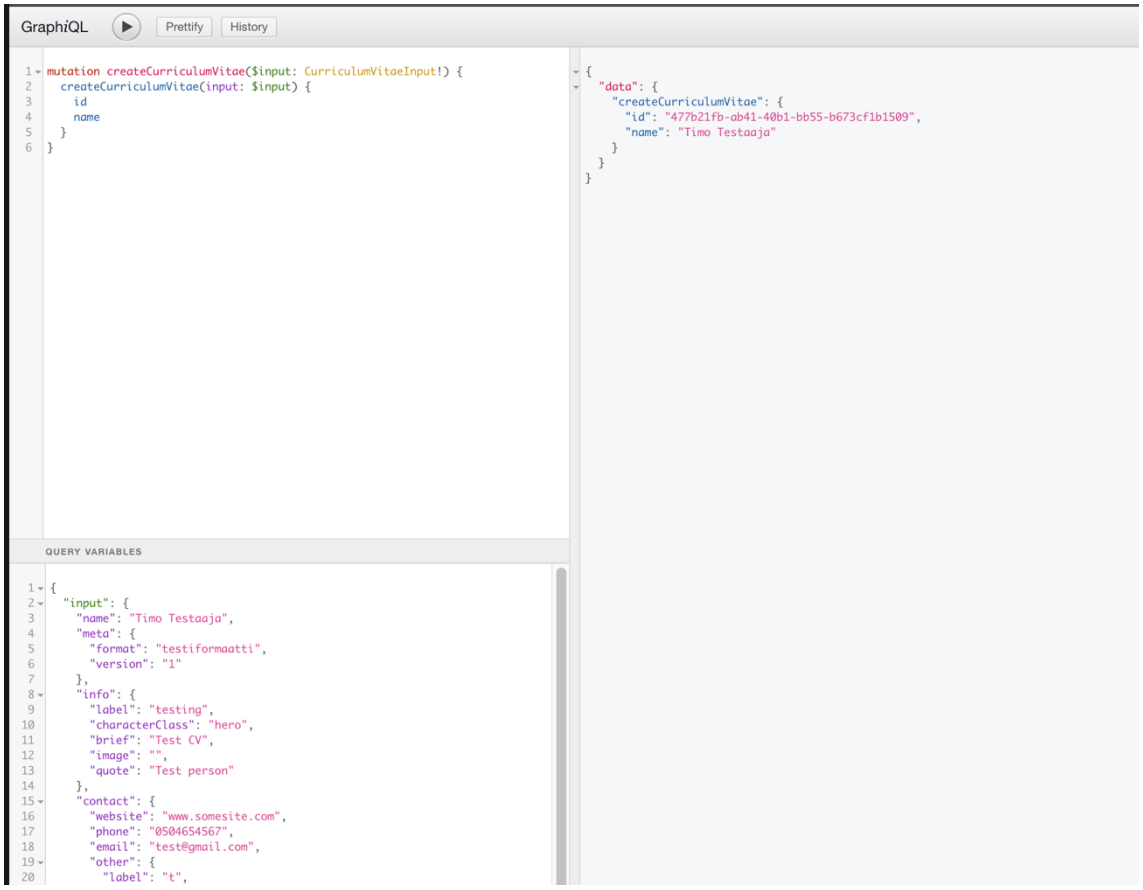
Kuvassa 21 on oletusnäkymä GraphiQL:n dokumentaatiosta. Hakua (query) ja muutosta (mutation) klikkaamalla siirtyy syvemmällä dokumentaatioon ja voi selata eri paluuarvoja ja tähän kyseiseen skeemaan määriteltyjä GraphQL:n tyyppejä.

4.1 Ensimmäinen testitapaus: uuden cv:n luominen

Ensimmäinen testitapaus oli uuden CV:n luominen CV-pankin avulla. Tämä on myös yksi tärkeimmistä testitapauksista, koska ilman tämän testin takana olevaa operaatiota ei järjestelmään saada uusi CV:itä. Testiin tarvittiin GraphiQL ja CV-pankki. Testi itsessään koostuu GraphQL-muuttajasta ja sen vaatimasta muuttuja tiedosta.

Muuttaja (mutation) luo MongoDB dokumenttikantaan uuden kokoelman tarvittavine tietoineen. Mikään kenttä ei ole pakollinen, vaan tietoja voidaan helposti lisätä päivittämällä kokoelmaa myöhemmin.

Kuvasta 21 näkee, että ylävasemmalla olevaan ikkunaan kirjoitetaan GraphQL-kyselyt, tai muuttajat ja sen vieressä oikealla näkyy tulosjoukko. Alavasemmalla näkyy kyselyissä ja muuttajissa tarvittavat muuttujat JSON-muodossa.

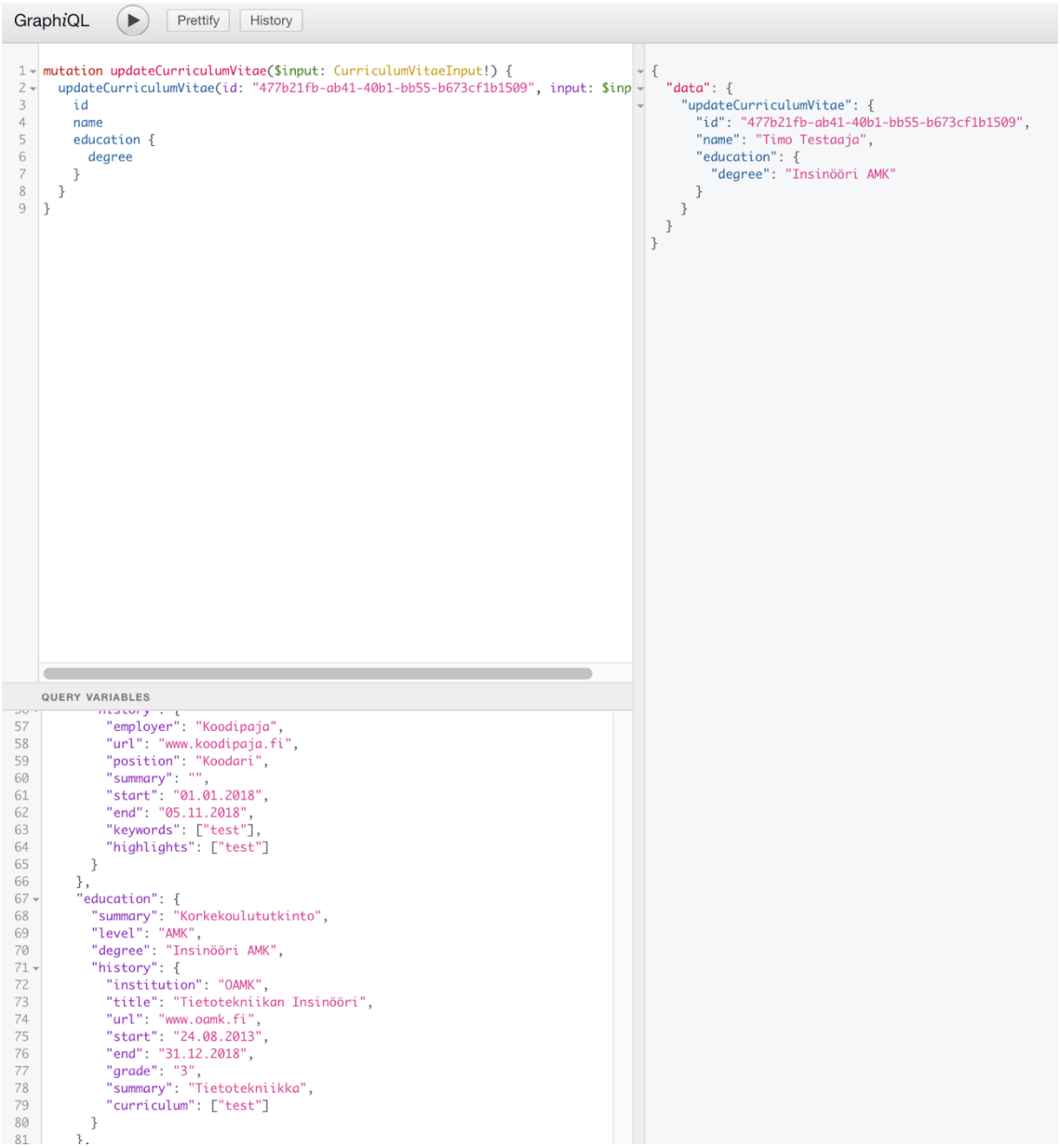


KUVA 21. Luo uusi CV GraphiQL:n avulla CV-pankkiin

4.2 Toinen testitapaus: päivitä CV

Toinen testitapaus on CV:n päivittäminen eli jo olemassa olevan CV:n tietojen ajanmukaistaminen tai muuttaminen. Tämä testi suoritetaan samalla tavalla kuin CV:n lisääminenkin ainoana eroa oli se, että käytetään sanaa ”update”, kun lisäämisessä käytetään sanaa ”create”. Huomioitavaa on myös, että muuttujissa on oltava koko muutettava tieto ja muutettava vain niitä kohtia, jotka halutaan muuttaa. Valitettavasti nykyinen versio ei tue osittaista päivittämistä.

Kuvassa 22 näkyy koulutietojen lisääminen (jälleen alavasemmalla) ja JSON-muuttujaa käytetään ylävasemmalla olevassa GraphQL-muutoksessa. Tulos näkyy oikealla.



KUVA 22. Päivitetään CV:tä

4.3 Kolmas testitapaus: CV:n poistaminen

Kolmas testitapaus oli suhteellisen selkeä näin GDPR-aikakaudella: käyttäjän data pitää pystyä poistamaan hänen sitä halutessaan.

Poisto testitapaus ei tarvitse muuttujia, ainoastaan GraphQL-muutoslauseen.

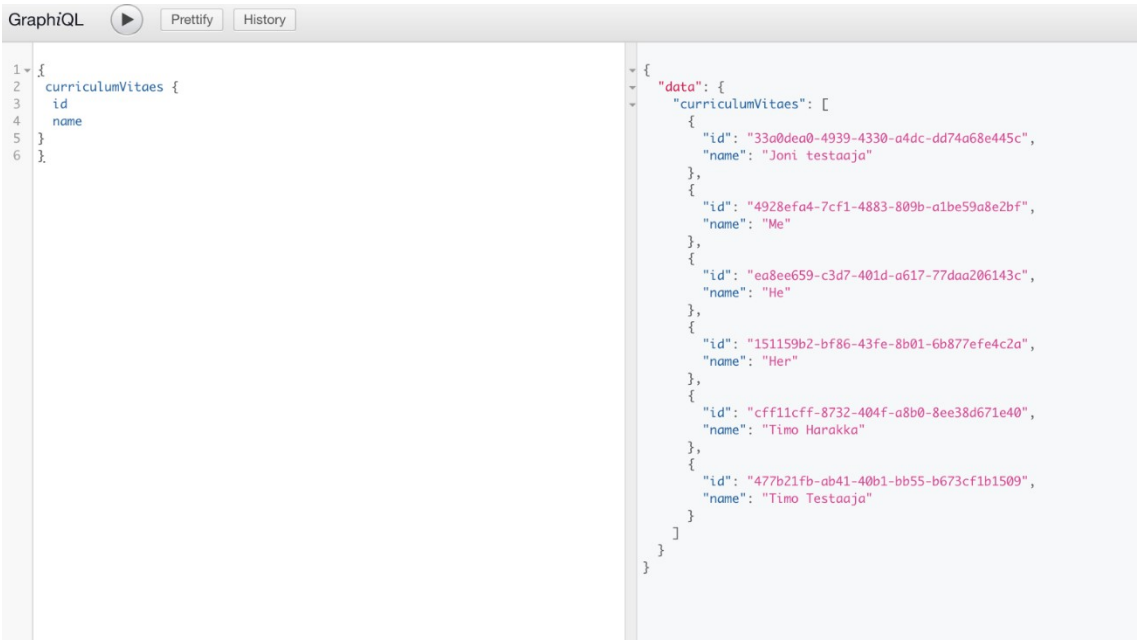
Kuvassa 23 näkyy CV:n tunnus, joka yksilöi poistettavan kokoelman. Oikealla on tulosjoukko.

KUVA 23. CV:n poistaminen

4.4 Neljäs testitapaus: CV:n hakemiset eri tavoin

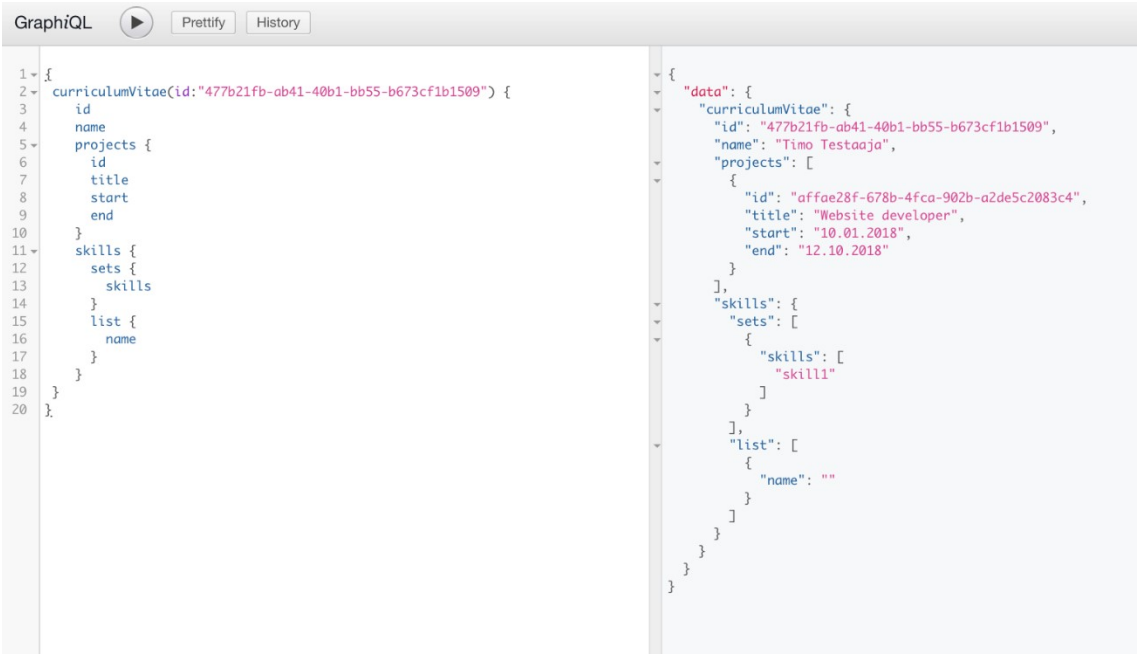
CV-pankista CV:itä voidaan hakea usealla eri tavalla. Testitapauksia oli kolme: haetaan kaikki järjestelmän CV:t, haetaan yksi CV tunnuksen avulla ja haetaan CV henkilön nimellä. Näiden lisäksi oli pari erikoishakua, joita ei saatu valmiiksi. Näiden kohdalla sovittiin, että ne tehdään vuoden 2020 alussa.

Kuvassa 24 näkyy kaikkien CV:eiden haku (näistä haluttiin vain tunnus –ja nimi-kentät). Vasemmalla siis haku ja oikealla tulosjoukko. Tämän kyselyn avulla on helppo tietää mitä CV:itä on CV-pankissa ja mitkä ovat niiden tunnukset: tunnustietoa tarvitaan monissa muissa muutoksissa ja kyselyissä.



KUVA 24. Haetaan kaikki CV:t CV-pankista

Seuraavat kyselyt yksilöidään joko tunnuksen tai sitten henkilön nimen perusteella. Tunnuksella haettaessa tarvitsee tunnus hakea, joko kaikista CV:eistä (kuten edellä kuvattiin), tai sitten se tarvitsee olla muutoin tallella. Nimellä haku onkin helpompi tapa hakea CV:itä, koska henkilön nimi varmasti tiedetään. Kyselyt toimivat samoin, kuin edellä kuvatut kyselyt ja muutokset.



KUVA 25. Haetaan CV tunnuksen avulla

KUVA 26. Haetaan CV nimen perusteella

4.5 Viides testitapaus: muunnetaan CV PDF:ksi ja lähetään sähköpostitse

Viimeinen testitapaus oli suhteellisen vaikea, koska se ei ole enää GraphQL:n peruskäyttöä. Tästä testitapauksesta ei ole kuvaa, koska testiin tarvitaan oikea sähköpostiosoite ja sen salasana.

Testi on alla olevan kaltainen GraphQL-kysely:

```
{
sendAsEmail(senderEmail: "some.email", password: "that.email.pwd",
toMail: "receivers.email", resumeName: "name.of.cv.to.be.sent")
}
```

Kyselyyn tarvitaan sähköpostiosoite, josta viesti lähtee (ja viestin lähettäjän salasana), sähköpostin vastaanottaja ja lähetettävän CV:n nimi. Operaatiota kokeiltiin useampaan otteeseen eri sähköpostiosoitteisiin (gmail) ja viesti saapui perille joka kerta onnistuneesti CV liitteenä PDF-muotoisena.

Näillä testeillä CV-pankkia voi ruveta käyttämään koeluontoisesti. Tiedon eheyttä ja tietoturvaa ei testattu. Tietoturvatestaus vaaditaan, jotta CV-pankkia voi käyttää

tuotannossa. Tosin tietoturva jätetään pois sovelluksen tästä vaiheesta, mutta se on tarkoitus toteuttaa myöhemmin.

Näiden lisäksi sovellukselle ei ole tehty testausta suuremmin kuormituksen suhteen. Kuten muidenkin lisätestien kohdalla todettiin, tämäkin testaus suoritetaan myöhemmin tarpeen tullen.

5 POHDINTA

CV-pankin tavoitteena oli luoda sovellus, jonka avulla pystyisi ylläpitämään konsulttien CV:itä ja tarvittaessa lähettää asiakkaille sähköpostilla konsultin CV.

Toteutus onnistui tavoitteiden mukaisesti, vaikkakin tavoitteita jouduttiin projektin aikana muuttamaan. Nämä muutokset kirjattiin ylös ja sovittiin, että ne tehtäisiin myöhemmin paremmalla ajalla.

CV-pankin suunnittelu ja toteutus onnistuivat suhteellisen hyvin, mutta kuten monella muullakin sovelluksella, niin tämänkin sovelluksen valmistumisen jälkeen on löytynyt paljon muutettavaa. On myös ilmaantunut uusia viitekehysteknologioita ja kilpailevia tiedonsiirtotapoja.

Aluksi koko ajatus lähti eräänlaisena ”Proof of consept” –tyyppisenä ratkaisuna, johon voi hyvin yhdistää monia uusia teknologioita. Tästä ajatuksesta seurasi raivoisi parin kuukauden ohjelmointi urakka, joka huipentui valmiiseen sovellukseen.

Prototyypissä oli pieniä poikkeamia suunnitellusta sovelluksesta, mutta ne voidaan kuitenkin korjata. Myös tutkimustyö jatkoa varten on alkanut ja luultavasti prototyyppi jää elämään omaa elämäänsä, mutta virallisempi versio CV-pankista tehdään hieman erilaisin teknologioin.

Ensimmäinen asia, jonka huomasin CV-pankkia tehdessäni, oli se että miten helppoa Kotlinilla oli ohjelmoida, jos osasi hyvin Javaa. Tosin hieman alkuihastuksen jälkeen huomasin, että Kotlinissakin oli omia vaikeitakin piirteitä, joiden oppimiseen menisi vielä aikaa. Ja luultavasti itse ohjelmointikieli pysyy Kotlinina, vaikka moni muu asia varmasti muutetaankin tästä nykyisestä versiosta.

Tietokanta ratkaisuna MongoDB oli toimiva. Se piti tiedon sisällään missä tahansa muodossa se sitten tulikaan sinne ja sitä pystyi käsittelemään helposti. Työn tekeminen olisi ollut varmasti huomattavasti hankalempaa, jos tietokannaksi olisi valittu perinteisempi tietokantajärjestelmä. Tämäkin jäänee paikallaan tulevaisuutta ajatellen.

GraphQL oli myös hieno löytö. Se yhdisti kaksi asiaa toisiinsa ja helpotti näin sovelluksen kehittämistä. Kyselyt yhdistyivät viestintäprotokollaan, joilloin saatiin mukavasti aikaan vain yksi päätepiste sovellukseen (endpoint). Tämä on monia muitakin sovelluksia helpottava toimenpide, koska yleensä alustapuolen koodissa voi olla kymmeniä päätepisteitä, joita jokaista on ylläpidettävä. Ongelmiakin GraphQL:ssä oli: tiedon tyypittäminen, sen esittäminen ja erilaisten kyselyiden rakentaminen. CV-pankkiin tuli paljon data-luokkia, joilla tieto esitetään sovelluksessa, mutta ainoastaan yksi tietorakenne: CurriculumVitae. Osa tästä monimutkaisuudesta tuli Javan tavasta esittää tieto luokkina. Toisaalta hyvänä puolena oli se, että kaikki luokat edustavat jotain rakennetta GraphQL:n tyypeissä. Siten sovellusta oli helppo ylläpitää muidenkin kuin alkuperäisen kehittäjän. Tiedon esittäminen luokkina toimii myös hyvänä opetuskeinona niille, jotka eivät ole paljon Javan kanssa tehneet sovelluksia.

Testaus on asia, jota CV-pankissa pitää selvästi parantaa. Nyt esteenä sille oli lähinnä pieni tiimikoko ja aika. Suunnittelin kuitenkin, että tulevaisuudessa suurin osa ominaisuuksista olisi automaattitestauksen alaisuudessa. Toki ajattelin myös, että paremmat testitapaukset on myös syytä luoda nyt niitä on kovin vähän. Toisaalta näitä varmasti tulee lisää, jos sovellusta käytetään enemmän.

Jälkikäteen ajateltuna Springin valinta ei ollut kaikkein optimaalisin ratkaisu, mutta se oli nopein, sillä minulla on kyseisestä viitekehyksestä vuosien kokemuksen. Seuraava askel on viedä opitut asiat seuraavalla askeleelle ja muodostaa prototyyppimäisestä sovelluksesta paremmin pilvialustalle sopiva. Tämä varmasti tarkoittaa sitä, että isosta viitekehyksestä joudutaan luopumaan ja tilalle otetaan jokin pienempi, joka kuitenkin tuo riittävät toiminnallisuudet. Tällä hetkellähän Spring Boot on kokoelma useita eri teknologioita, joita ei välttämättä CV-pankissa tällä hetkellä käytetä.

Kaiken kaikkiaan CV-pankin suunnittelu ja toteuttaminen olivat mielenkiintoisia kokemuksia. Myös testaukseen liittyvät toiminnot tuli mietittyä tavallista tarkemmin, koska nyt oli itse vastuussa kaikesta. Erittäin opettavaista ja varmasti käyttökelpoista muissakin mahdollisissa projekteissa, joita tulevaisuudessa teen.

LÄHTEET

1. Hämäläinen, Pertti 2016. Mikropalvelut nuosivat hypen huipulle – mitä hyötyä niistä on? Saatavissa: https://www.tivi.fi/Kaikki_uutiset/mikropalvelut-nousivat-hypen-huipulle-mita-hyotya-niista-on-6534379. Hakupäivä 21.9.2018

2. Vase, Tuomas 2016. Docker ja konttiteknologiat – ratkaisuja ja suorituskykyä. Saatavissa: https://eu.landisgyr.com/better-tech/docker-ja-konttiteknologiat-ratkaisuja-ja-suorituskykyä. Hakupäivä 21.9.2018

3. Quick Programming Tips 2017. History of Spring Framework and Spring Boot. Saatavissa: *https://www.quickprogrammingtips.com/spring-boot/history-of-spring-framework-and-spring-boot.html*. Hakupäivä 18.9.2018

4. FRESH. The FRESH Resume Schema. Saatavissa: https://github.com/fresh-standard/fresh-resume-schema. Hakupäivä 20.9.2018