Juho Pessi

Java Spring Boot ja Go web-kehityksen back end -kielinä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

18.6.2021

Tiivistelmä

Tekijä:	Juho Pessi
Otsikko:	Java Spring Boot ja Go web-kehityksen back end -kielinä
Sivumäärä:	37 sivua
Aika:	18.6.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile solutions
Ohjaaja:	Lehtori Ilkka Kylmäniemi

Insinöörityössä oli tarkoituksena perehtyä Java Spring Boot -ohjelmistokehyksen ja Go-ohjelmointikielen ominaisuuksiin ja eroihin. Lisäksi vertailtiin, miten teknologiat vertautuvat keskenään yksinkertaisen back end -sovelluksen toteutuksessa, jossa luodaan REST-rajapinnat web-sovellukselle.

Työhön valikoitui Java Spring Boot -ohjelmistokehys, koska se oli kysytty kieli työmarkkinoilla tätä insinöörityötä tehtäessä. Tämän lisäksi vertailuun tarvittiin toinen teknologia eli Go, joka valikoitui mukaan tekijän oman mielenkiinnon takia.

Työn tarkoituksena oli selvittää, miten valitut teknologiat vertautuvat keskenään, ja ymmärtää teknologioiden eroa sen verran, että pystyttiin erottamaan kummankin vahvuudet valittaessa back end -sovelluksen teknologiaa. Tämän lisäksi työtä voidaan käyttää oppaana Gon ja Spring Bootin käyttöönottamisessa ja yksinkertaisten REST-rajapintojen luomisessa omalle projektille tai muussa oppimistarkoituksessa.

Työssä huomattiin, että vertailussa olleilla teknologioilla pystyi helposti toteuttamaan yksinkertaiset REST-rajapinnat, kunhan teknologian asennus ja käyttöönotto oli tehty. Tämän lisäksi kuitenkin huomattiin, että ne poikkeavat toisistaan huomattavasti. Go on käytössä monesti yksinkertaisempi.

Avainsanat: Java Spring Boot, Go, web-kehitys

Abstract

Author:	Juho Pessi
Title:	Java Spring Boot and Go, Comparison between as backend-applications
Number of Pages:	37 pages
Date:	18 June 2021
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Instructor:	Ilkka Kylmäniemi, Senior Lecturer

The purpose of this thesis was to go through the features and the differences of Java Spring Boot framework and Go programming language. Another purpose was to find out how these technologies compared between each other when programming a simple REST-api for a web-application.

The reason why these technologies were chosen for comparison in this study is that Spring Boot was quite a popular requirement in job advertisements at the time this study was written. Also, something was needed to compare with Spring Boot, and therefore Go programming language was chosen to compare these two technologies.

This thesis aimed at answering the following question: How features of Java Spring Boot and Go compare as technologies and what the basic features of the two technologies are. Also, this thesis can be used as a tutorial for making simple REST-apis with Spring Boot and Go.

During the implementation of these REST-apis with the two technologies, it was noticed that if when planning to make some very basic and simple REST-apis for web-applications, implementation would be faster to implement with Go-language instead of Spring Boot, because most of the time Go appears to be more logical and easier to understand than Spring Boot.

Keywords: Java Spring Boot, Go, web-development, REST

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on vertailla Java Spring Boot -ohjelmistokehyksen ja Go-kielen käyttöä web-sovelluksen back end -ohjelmoinnissa ja miten niiden käyttö ja toiminnat vertautuvat keskenään. Insinöörityössä käydään lävitse molempien teknologioiden ominaisuuksia ja muita perustietoja sekä sitä, minkälaista eroja ja yhtäläisyyksiä löytyi, ennen kuin lähdettiin toteuttamaan omaa sovellusta ja vertailua.

Työn tavoitteena on oman osaamisen laajentaminen useampien back end -kielten osaamisessa sekä edellytys toteuttaa back end -sovellus REST-rajapintoineen molempia teknologioita käyttäen. Lisäksi pyritään erottamaan teknologiat keskenään, jotta on helpompi päättää, minkä perusteella valitaan teknologia käyttöön uuden projektin suunnittelussa.

Insinöörityön alussa tutustutaan molempiin teknologioihin. Tämä alkuosio sisältää historiaa, perustietoja, teknologioiden ominaisuuksia sekä heikkouksia ja tutustutaan siihen, miten teknologiat otetaan käyttöön ennen varsinaista sovellusten toteuttamista. Tämän jälkeen tehdään vertailua, miten teknologiat vertautuvat toisiinsa ennakkotietojen perusteella.

Projektiosiossa käydään läpi ohjelmistokokonaisuuden suunnittelu ja toteutus. Osion alussa tehdään tietokantarakenteen, rajapintojen ja käyttöliittymän suunnittelu. Osiossa tehdään REST-rajapintojen toteutus Spring Bootilla ja Golla ja sovellukset yhdistetään React.js:llä toteutettuun käyttöliittymään.

2 Spring Boot -ohjelmistokehys ja Go-ohjelmointikieli

2.1 Spring Boot -ohjelmistokehys

Spring Boot on Spring-ohjelmistokehyksen moduuli, ja Spring itsessään on Java-ohjelmistokehys. Springiä kuvaillaan Springin sivuilla seuraavasti: Spring tekee Java-ohjelmoinnista nopeampaa, helpompaa ja turvallisempaa. Spring-ohjelmistokehyksen suunnittelussa on myös keskitytty nopeuteen ja yksinkertaisuuteen. (1.) Vuonna 2018 tehdyn tutkimuksen (2) mukaan Spring Boot on ollut maailman käytetyin Java-ohjelmistokehys.

Spring on julkaistu vuonna 2002 ja sen päälle on rakennettu useita eri ohjelmistokehyksiä. Rod Johnson yhdessä Juergen Hoellerin ja Yann Cafoffin kanssa kehitti Springin, joka on avoimen lähdekoodin ohjelmistokehys. He kehittivät Springin helpottaakseen ja nopeuttaakseen ohjelmointitiimien työtä. Stackoverflow'n kehittäjille tehdyn kyselyn mukaan Spring oli vuonna 2020 kuudenneksi pidetyin web-ohjelmistokehys. (3.) Spring Boot -moduuli julkaistiin vuonna 2014 (4).

2.2 Spring Bootin ominaisuuksia

Automaattiset asetukset

Spring Boot sisältää ominaisuuden, joka asentaa automaattisesti asetukset käyttäjän puolesta. Spring Boot mahdollistaa asetusten muuttamisen automaattisesti ohjelmassa käytettyjen riippuvuuksien mukaisesti. Toisin sanoen ohjelma käy läpi listan ohjelmassa käytetyistä eri riippuvuuksista ja tekee tarvittavat asetukset tämän listan pohjalta. Esimerkiksi jos MySQL on listattu ohjelmaan riippuvuudeksi, tekee ohjelma ”MySQL connector sisällytetty” -muutokset Spring-sovellukseen. Automaattisten asetusten lisäksi ohjelmaan voidaan sisällyttää myös mukautettu asetus. Tämä tapahtuu luomalla uusi luokka, joka ylikirjoittaa ”MySQL connector” -oletusasetukset. (5.)

Itsenäinen sovellus

Ohjelmistokehyksellä toteutettu sovellus on niin kutsuttu itsenäinen sovellus. Tämä tarkoittaa sitä, että sovellusta suorittaessa ei ole tarvetta erilliselle verkkopalvelimelle. (5.) Toisin sanoen Spring Boot antaa käyttäjälle mahdollisuuden luoda itsenäisiä sovelluksia tai mikropalveluita, jotka ovat helppo ottaa käyttöön. Springin omilla sivuilla (6) sanotaan: ”just run" eli vapaasti käännettynä ohjelman voi vain suorittaa.

Sulautettu verkkopalvelin

Springin mukana tulee mahdollisuus käyttää esikoottuja verkkopalvelimia, ja Spring Boot sisältää Tomcat-, Jetty- ja Undertow-verkkopalvelimet. Ne ovat käytössä heti, eivätkä vaadi minkään uusien lisäosien asentamista. Tämä ominaisuus tarjoaa siis nopeamman ja tehokkaamman sovelluksen käyttöönoton, joka nopeuttaa käynnistyksessä käytettävää aikaa. (7.)

Spring Boot CLI

Spring Boot CLI on Spring Bootin tarjoama komentorivipohjainen käyttöliittymä, joka mahdollistaa yksinkertaisen Spring-pohjaisen verkkosovelluksen luomisen. Tämä verkkosovellus on mahdollista kirjoittaa Groovy-ohjelmointikielen avulla. Groovyn tärkein ominaisuus on tehdä Java-ohjelmoinnista helpompaa, kun taas Spring Bootin yksi ominaisuuksista on tehdä Spring-ohjelmoinnista helpompaa. Toisin sanoen käyttämällä Spring Boot -sovelluksessa Spring Boot CLI:tä yhdessä Groovyn kanssa mahdollistetaan vieläkin yksinkertaisempaa sovelluskehitystä. (8.)

Actuator

Spring Bootin ominaisuus Actuator tekee mahdolliseksi kehittäjän nähdä, mitä kaikkea tapahtuu Spring Boot -sovelluksen sisällä, kun sovellusta suoritetaan. Spring Bootin automaattisten asetusten myötä Spring Bootin sisällä tapahtuu

paljon asioita kehittäjältä piilossa. Actuator tuo mahdollisuuden kehittäjälle nähdä, mitä niin sanotusti sovelluksen pinnan alla tapahtuu. Actuator siis tuo mahdolliset riskit tai ongelmat kehittäjien nähtäville.

Actuator tarjoaa kehittäjille monenlaista tietoa käynnissä olevasta sovelluksesta. Esimerkiksi käyttämällä Actuatoria kehittäjä pystyy selvittämään tarkasti, mikä beans on konfiguroitu Application contextiin, mitkä asetukset autoasetukset ovat valinneet, mitkä muuttujat, mitkä järjestelmäasetukset tai mitkä komentoriviargumentit ovat käytettävissä. Tässä vain muutamia Actuatorin ominaisuuksia. (8.)

Spring Initializr -projektinluontityökalu

Spring Initializr -projektinluontityökalu on kehitetty helpottamaan projektin luontia kunnioittaen projektin rakennetta. Spring Initializr on verkkosovellus, joka auttaa kehittäjää generoimaan projektin. Projektin luonnissa voi valita muun muassa, haluaako käyttää Javaa vai Kotlinia. Lisää Spring Initializrn ominaisuuksista myöhemmin. (8.)

Hallittavuus

Kuten kaikki ohjelmointikielet ja ohjelmistokehykset, Spring Boot sisältää myös huonompia ominaisuuksia.

Yksi suurimpia käyttäjien kohtaamia ongelmia on Spring Bootin hallinnan vaikeus. Tämä johtuu usein ohjelmistokehyksen itsestään asentamista ylimääräisistä riippuvuuksista, jotka monesti päätyvät käyttämättömiksi ja näin ollen vain lisäävät ohjelmiston kokoa. (9.)

Aloittelijalle monimutkainen

Spring Bootin käyttäminen saattaa olla aloittelijalle hieman monimutkaista ja vaikeasti ymmärrettävää. Tämä johtuu siitä, että käyttääkseen Spring Bootia

olisi suositeltavaa osata jo Springiä. Monet Springin ominaisuuksista, kuten Dependency injection, Proxies ja AOP-ohjelmointi, olisi hyvä osata ennen Spring Bootin käyttämistä, sillä monia Spring Bootin peruasioita ei löydy sen omasta dokumentaatiosta. Ohjelmistokehystä ei suositella massiivisten sovellusten tekoon vaan se soveltuukin paremmin mikropalveluiden rakentamiseen. (9.)

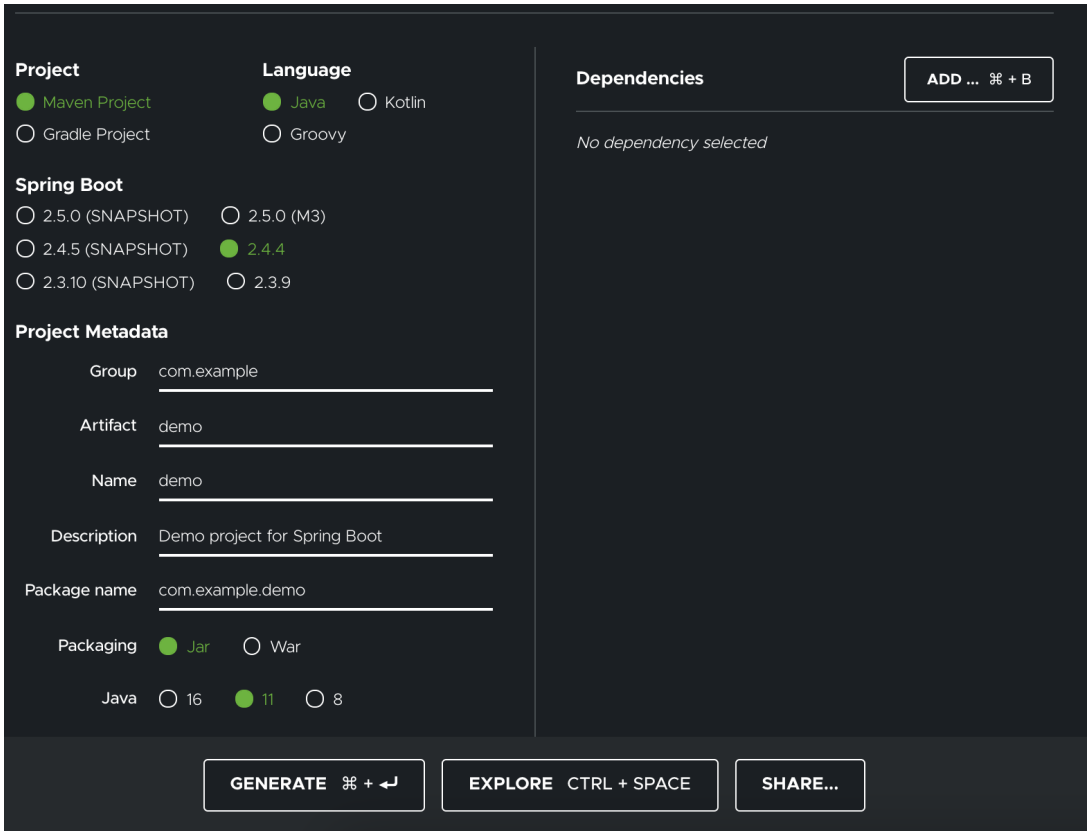
2.3 Spring Bootin käyttöönotto

Spring Bootin käyttöönottoa varten tarvitaan JDK 1.8, Gradle 4 tai uudempi tai Maven 3.2 tai uudempi versio (6). Niiden asentamista ei kuitenkaan käydä läpi tässä. Spring Bootin käyttöönoton voi tehdä usealla eri tavalla. Mahdollisia tapoja on käyttää Spring Bootin CLI:a tai Spring Initializria tai käyttää Eclipseen tai Intellij IDEAan rakennettuja työkaluja. Tässä kuitenkin käydään läpi, miten ohjelman luonti tapahtuu Spring Initializria käyttäen.

Spring Initializr -projektinluontityökalun käyttö

Kenties yksinkertaisimmin Spring Bootin käyttöönotto tapahtuu Spring Initializr -projektinluontityökalun avulla, joka löytyy osoitteesta https://start.spring.io/ (10).

Ensimmäiseksi projektinluontityökalun alussa käyttäjä valitsee, onko kyseessä Maven- vai Gradle-projekti. Tämän jälkeen valitaan käytettävä ohjelmointikieli sekä Spring Boot -versio. Projektin metatieto lisätään kohdassa Project Metadata ja valitaan vielä Java-versio. Lopuksi valitaan Jar- tai War-pakkaus riippuen siitä, onko kyseessä web-sovellus vai ei. Näiden lisäksi projektiin voi lisätä suoraan lisäosia, kuten sulautettuja kontteja, erilaisia tietokantoja ja testikontteja. (Kuva 1.)



Kuva 1. Spring Initializrin asetukset (10).

Riippuvuudet

Spring Boot sisältää kattavan määrän helposti asennettavia riippuvuuksia ohjelmaa luotaessa. Näitä riippuvuuksia voi asentaa jo alussa tai myöhemmässä vaiheessa projektin niin vaatiessa. Kun projekti on luotu käyttäen Mavenia kuten esimerkkikuvassa 2, kansion juuresta löytyy pom.xml-tiedosto. Tiedostossa näkyy juuri tehdyt asetukset ja asennetut lisäosat eli riippuvuudet. Oletuksena projektinluontityökalu asentaa seuraavat riippuvuudet, kuten kuvassa 2 näkyy.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Kuva 2. Projektin ohjelmistoriippuvuudet.

2.4 Go-ohjelmointikieli

Go tai toisin sanoen Golang, on Googlen kehittämä avoimeen lähdekoodiin perustuva ohjelmointikieli (11). Go-ohjelmointikielen sanotaan olevan nykyaikainen C-kieli syntaksinsa puolesta. Kielenä Go on niin sanottu käännetty kieli. Tämä tarkoittaa sitä, että kieli on käännetty muotoon, jonka prosessori ymmärtää. Kieli siis on erilainen kuin esimerkiksi tässä insinöörityössä toisena kielenä oleva Java-pohjainen Spring Boot. Java käännetään ensiksi tavukoodiksi, joka suoritetaan virtuaalikoneiden toimesta. (12.) Tutkimuksen (13) mukaan Go on todistetusti nopeampi ohjelmointikieli kuin yleisesti käytetyt Java ja Python.

Go-ohjelmointikielen historia ylettyy vuoteen 2009, jolloin sen loivat Googlen työntekijät Rob Pike, Robert Griesemer ja Ken Thompson. Heidän tavoitteensa Gota luodessa oli yhdistää muiden ohjelmointikielten parhaita ominaisuuksia, jotka ovat seuraavat:

- helppokäyttöisyys tuottavuuden rinnalla

- tehokas staattinen kirjoitusasu
- edistynyt verkkotyöskentely ja moniydinprosessorin tehokas käyttö. (14.)

2.5 Gon ominaisuuksia

Helppokäyttöisyys

Ohjelmointikielenä Go on uusi tulokas. Kieli on kuitenkin helposti opittavissa, jos pohjalla on jo osaamista vanhoista kielistä, kuten C tai Java. Vaikka termit ja syntaksi ovat hieman erilaiset, kielessä käytetään samanlaisia menettelytapoja ja se on tämän takia helppo kieli oppia. (14.)

Skaalautuvuus

Yksi syistä, miksi Go-kieli on paljon käytetty, on sen mahdollistama tuki funktioiden yhtäaikaiseen suorittamiseen. Tämän mahdollistaa sen sisältämä Goroutines-ominaisuus. Ominaisuus mahdollistaa siis funktioiden suorittamisen niin yksittäin kuin niiden suorittamisen yhtäaikaisesti. Toisin sanoen Goroutines on kevyt työkalu säikeiden suorittamiseen yhtäaikaisesti.

Goroutines käyttää muistia vain 2 kilotavua, mikä tekee muistinkäytöstä hyvin vähäistä. Tämä ominaisuus tekee Go-kielestä skaalautuvan, kuitenkin se pysyy kevyenä, jos tarve yhtäaikaisten prosessien suorittamiseen onkin kasvanut. (14.)

Go-ohjelmointityökalut

Työkalujen käyttöönotto esimerkiksi yritysmaailmassa tai harrastelussa ei ole ongelma avoimen lähdekoodin ansiosta. Tämän lisäksi on paljon editoreita, ohjelmointiympäristöjä ja lisäosia, joita voi helposti ladata Go-kielen GitHub-tietolähteestä. Go sisältää työkalut, jotka mahdollistavat kehittäjille turvallisen tavan

käyttää muistia ja hallita objekteja ja helpon hallinnan ohjelmien keräämille ylimääräiselle datalle sekä staattisen kirjoitusasun. (14.)

Suosio

Go on sijalla yksi kaikkien kysytyimpien ohjelmointikielien joukossa ympäri maailman (15).

Ohjelmistokehyksien puute

Kuten aikaisemmin todettiin, kaikissa ohjelmointikielissä on myös heikkoutensa ja Go-kieli ei ole tässäkään poikkeus.

Tällä hetkellä Golle ei ole olemassa mitään varteenotettavia ohjelmistokehyksiä, joita muut isot ohjelmointikielet sisältävät (16).

Virheiden hallinta

Go-ohjelmat vaativat erillisen funktion palauttaakseen mahdolliset virheilmoitukset. Tämä saattaa aiheuttaa ongelmia virheiden löytämisessä. Tämä johtaa vääriin tapoihin virheidenhallinnassa. Olemassa on kuitenkin työkaluja, jotka auttavat löytämään tämänkaltaisia ongelmia. Niiden lisäksi täytyy kirjoittaa paljon erilaisia lohkoja virheiden tarkistamiseen ja siihen, kuinka niitä tulisi hallita. Lohkojen kirjoittaminen vaikeuttaa nopeasti koodin helppolukuisuutta. (16.)

Paketinhallinta

Oletuksena kielen paketinhallinta ei voi luoda riippuvuuspuita, jos pakettiversiot jostain syystä pitäisi olla juuri tietyt. Tämä johtaa siihen, että koontiversiota luotaessa saattaa syntyä eri pakettiversioita riippuen siitä, milloin koontiversio on luotu. (16.)

Yksinkertaisuuden tuomat ongelmat

Vaikka kielen yksinkertaisuus on yksi Gon parhaita ominaisuuksia, se koituu myös yhdeksi kielen heikkoudeksi. Go saattaa olla helppo kieli valita ohjelmointikieleksi, mutta se vähentää kielen monipuolisuutta. (17.)

Uusi kieli

Uutena kielenä Go on hyvin vakiinnuttanut paikkansa vahvana ohjelmointikielenä, mutta se on silti vielä altavastaajana monien vanhempien, jo paikkansa lunastaneiden kielten joukossa. Kielen kirjasto saattaa olla viisaasti suunniteltu ja tehokas, mutta vanhempiin kieliin verrattuna, esimerkiksi Javaan ja sen valtaviin kirjastoihin sekä käyttäjämääriin, Go on vielä heikommassa asemassa. (17.)

Virtuaalikoneet

Go on tehnyt tietoisen päätöksen olla käyttämättä virtuaalikoneita ja tämä on yksi kielen heikkouksista. Sille on syynsä, miksi tänä päivänä monet suositut ohjelmointikielet sisältävät valmiiksi virtuaalikoneen. Pitää kuitenkin ottaa huomioon, että vaikka virtuaalikoneet mahdollistavat tehokkaamman koodauksen, niin virtuaalikoneet tuovat myös mukanaan huomattavasti Go-kieltä isommat tiedostokoot. (17.)

Gon käyttötarkoitukset

Kuten monien muiden ohjelmointikielten kohdalla, myös Go soveltuu hyvin monenlaisiin projekteihin, halusi sitten työskennellä järjestelmien, verkko-ohjelmoinnin, big datan, koneoppimisen tai ääni- ja videoeditoinnin ym. parissa. Näin ollen ohjelmointikielen käyttötarkoitukset ovat hyvin moninaiset. (14.)

2.6 Gon käyttöönotto

Gon käyttöönotto on yksinkertaista. Ennen varsinaista käyttöönottoa olisi hyvä tietää jotain ohjelmoinnin perusteista. Vaikka kieli on hyvin yksinkertainen, perustieto ohjelmoinnista helpottaa käyttöönottoa. Tämän lisäksi tarvitaan jokin tekstieditori. Kieli myös tulee asentaa koneelle osoitteessa https://golang.org/doc/install löytyvien ohjeiden mukaan. Tämän jälkeen Go on valmis käytettäväksi. (18.)

Kuten huomataan, on ohjelmointikielen käyttöönotto ja aloitus helppoa. Tämän lisäksi yleinen ”Hello World!” -ohjelman luonti on yksinkertaista, kuten esimerkkikoodissa 1 näkyy.

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

Esimerkkikoodi 1. Go, Hello World.

2.7 Java Spring Bootin ja Gon vertailu

Kun on käyty läpi sekä Java Spring Bootin että Gon ominaisuuksia erikseen, nyt on aika suorittaa vertailua näiden kahden välillä. Tässä osiossa käydään läpi, mitä eroavaisuuksia eri lähteiden mukaan kielet tarjoavat, ja myöhemmin projektiosion jälkeen käydään läpi lisää, mitä havaintoja eroavaisuuksista tai yhtäläisyyksistä löytyi, kun oli työskennelty kielien parissa.

Koska Spring Bootin on Java-pohjainen ohjelmistokehys, tässä luvussa tehdään myös vertailua Javan ja Gon välillä sillä.

Kielten väliset yhteneväisyydet

Molemmat kielet sisältävät ominaisuuden, joka estää sovelluksen muistivuotoja. Toisin kuin monissa C-pohjaisissa kielissä kehittäjän tulee huolehtia muistivuodoista, molemmista vertailussa olevista kielistä löytyy tähän automaattinen työkalu. Molemmat kielet sisältävät Garbage collectorin, jonka tehtävänä on estää automaattisesti muistivuotoja, mikä tekee kehittäjien työstä yksinkertaisempaa. (19.)

Vaikka molemmat vertailussa olleista kielistä ovat järjestelmäriippumattomia, Java kuitenkin tarvitsee Java Virtual Machinen komentotulkikseen. Go sen sijaan kääntää koodin binääritiedostoon riippumatta sen suorittavasta järjestelmästä. Novikaun (19) mukaan Java saattaa olla vähemmän järjestelmäriippumaton kuin Go, koska Go tarvitsee toimiakseen uuden binääritiedoston joka kerta uudelle järjestelmälle käännettäessä. Binaarien kääntäminen eri järjestelmille erikseen on aikaa vievää varsinkin testaus- ja DevOps-näkökulmasta. Tämän lisäksi Gon kääntäminen ei toimi kaikissa tapauksissa, varsinkin käytettäessä CGo-työkalua. CGo on työkalu, joka tekee mahdolliseksi Go-pakettien suorittaa C-koodia. (18.) Javan tapauksessa taas kehittäjä voi käyttää samaa JAR-tiedostoa joka paikassa, missä on Java Virtual Machine. Go tarvitsee myös vähemmän keskusmuistia eikä vaadi ylimääräisiä ohjelmia kuten virtuaalikoneiden asentamista toimiakseen. (19.)

Kielten väliset eroavaisuudet

Go ei ole olio-ohjelmointikieli. Yksi Gon ja Javan isoimmista eroavaisuuksista on se, että Go ei sisällä luokkien mahdollisuutta periä toisia luokkia, kuten Javassa on tapana. Go ei sisällä siis ollenkaan olioita vaan asiat hoituvat tietueiden avulla. Go voi simuloida joitain olio-ohjelmoinnin ohjelmointimalleja tarjoamalla käyttöliittymiä ja toteuttaa käyttöliittymät tietueilla. Tietueita voi myös sulauttaa toisiinsa, mutta sulautetut tietueet eivät saa käyttöoikeutta isännöivän tietueen datalle ja metodeille. Go käyttää koostamista perimisen sijaan yhdistäessään dataa.

Go-kielessä ei käytetä riippuvuusinjektointia, vaan kaikki data koostetaan selkeästi yhteen. Tämän takia Novikaun (19) sanojen mukaan suositeltu lähestymistapa Go-ohjelmointiin on käyttää mahdollisimman vähän taikatemppuja. Kaiken tulisi olla todella selkeästi esillä ulkopuolisille koodin arvioijille, ja kehittäjien tulisi ymmärtää, miksi Go-koodi käyttää muistia, tiedostojärjestelmää ja muita lähteitä, Novikau summaa.

Java taas vaatii kehittäjiltä enemmän huomiota kirjoitettaessa koodin toimintalogiikkaa ja sen määrittelyssä, kuinka data on luotu, suodatettu, muutettu ja tallennettu. Sovelluksen infrastruktuurista ja tietokantojen hallinnasta voi helposti huolehtia käyttämällä tähän tarkoitettua ohjelmistokehystä, kuten Spring Bootia.

Koska molemmat ohjelmointikielet ovat C-pohjaisia ja näin ollen kielet muistuttavat syntaksinsa puolesta paljon toisiaan, Java-kehittäjän on usein helppo oppia lukemaan Gon syntaksia ja vastaavasti Go-kehittäjällä on matala kynnys ymmärtää Javan syntaksia.

Mahdollisesti ensimmäisenä huomiona kielten eroissa voidaan nähdä, että Gon syntaksissa ei esiinny kaksoispistettä muuta kuin satunnaisissa tapauksissa. Javassa taas kaksoispistettä käytetään aina rivin lopussa. (19.) Modernien kielten kohdalla tämä kaksoispisteen poisjättäminen saattaa tehdä kielen luettavuudesta selkeämpää.

Hyvänä esimerkkinä kielten eroavaisuuksista on yksinkertainen muuttujien määrittäminen. Javassa muuttuja alustetaan kirjoittamalla muuttujan datatyyppi ensin, minkä jälkeen annetaan muuttujalle nimi, kuten esimerkkikoodissa 2.

```
String nameOfThevariable;
```

Esimerkkikoodi 2. Java-muuttuja.

Gossa taas muuttuja alustetaan päinvastaisessa järjestyksessä, kuten esimerkkikoodissa 3.

```
nameOfTheVariable string
```

Esimerkkikoodi 3. Go-muuttuja.

3 Kuljetustenhallintasovellus

Insinöörityön projektiosuudessa oli tarkoituksena kehittää kuljetusyritykselle sähköinen kuljetustenhallintasovellus. Sovelluksen tehtävänä oli toimia kuljetusyrityksen sähköisenä ajopäiväkirjana.

Projektissa ei ollut tarkoitus löytää parhaita mahdollisia teknologioita toteuttaa sovellusta, vaan pikemminkin tarkastella, miten Spring Boot ja Go soveltuvat tämänkaltaisen sovelluksen back end -kieleksi. Projektissa siis toteutettiin molempia kieliä käyttäen back end -sovellukset, jossa React huolehtii front end -sovelluksesta ja MongoDB tietokannasta.

Työtä aloitettaessa kuljetusyrityksen ajopäiväkirjan hallinta tehtiin vanhanaikaisesti ruutupaperia käyttäen. Kuljettaja vastaanotti puhelun, minkä jälkeen tiedot kirjattiin ruutupaperille, ja tämä tapahtui lähes poikkeuksetta autossa, myös ajon aikana.

Vaikka yrityksessä oli todettu paperille kirjaamisen olevan nopea ja helppo tapa hoitaa asia, oli kuitenkin huomattu, että teksti, joka oli jo kertaalleen kirjoitettu, täytyi monesti puhtaaksikirjoittaa myöhemmin laskutusta varten. Tämän takia oli huomattu tarpeen kasvaneen prosessin sähköistämiselle, mikä poistaisi prosessista yhden välivaiheen.

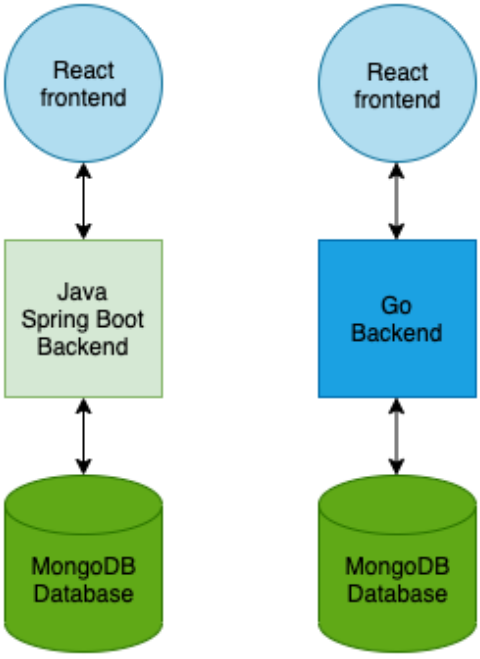
3.1 Projektisuunnitelma

Kuvasta 3 näkee, miten sovellusprojekti koostui kolmesta eri osasta:

- Tietokanta toteutettaisiin tässä projektissa käyttäen MongoDBtä. MongoDB valikoitui tähän projektiin sen yleisyyden takia, sillä tietokantajärjestelmä oli tutkimuksen (20) mukaan viidenneksi käytetyin tietokantajärjestelmä vuonna 2020.

- Käyttöliittymän toteutus tehtäisiin käyttämällä React.js-ohjelmistokehystä. Reactiin päädyttiin, koska se oli tekijälle tutuin ohjelmistokehys. Tämän päätöksen tein siksi, että halusin keskittyä enemmän oppimaan Java Spring Bootia ja Gota ja uuden front end -kielen opetteluun ei tarvinnut käyttää niin paljoa aikaa.

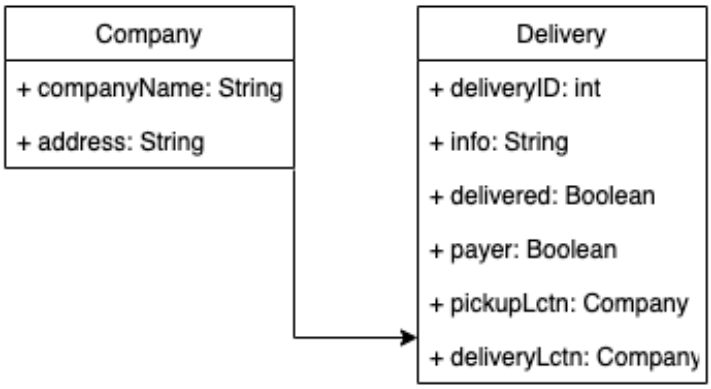
- Back end -osio toteutettaisiin kahdella eli teknologialla. Teknologioiksi valikoituivat Java Spring Boot ja Go. Tällä tavoin pystyttäisiin vertailemaan paremmin kahden eri teknologian käyttämistä back end -kielenä. Nämä valikoituivat projektiin sen takia, että halusin syventää Java Spring Boot -osaamistani. Sen sijaan Go tuli mukaan, koska se on maailman kysytyin ohjelmointikieli, kuten aikaisemmin jo mainittiin.



Kuva 3. Kuljetustenhallintasovelluksen projektikaavio.

Tietokantataulun suunnittelu

Koska sovelluksesta oli tarkoitus rakentaa aluksi pienin toimiva tuote ja näin ollen kirjautuminen jätettiin pois, tietokantarakenne jäisi melko yksinkertaiseksi. Tämän vuoksi tietokantataulun rakenne pysyisi vielä suhteellisen yksinkertaisena. Kuvan 4 kaaviosta löytyvät tällä hetkellä seuraavat taulut: Delivery-taulussa ovat tarvittavat tiedot kuljetustoimeksiannosta. Tauluun liitetään Company-taulu, joka sisältää asiakkaan tiedot sekä tavaran nouto- ja toimitusosoitteet.



Kuva 4. Tietokantataulun rakenne.

Esimerkkikoodi 5 on projektin yhdestä JSON-objektista, josta näkee kaikki tarvittavat tiedot korttia varten.

```
{
    "_id": "6085a16a6df2477a99b6c7a5",
    "date": "maanantai 30.1.2021",
    "info": "Tähän voidaan laittaa kuljetusohjeet",
    "delivered": false,
    "payer": true,
    "pickupLctn": {
        "companyName": "Penan autoliike Oy",
        "address": "Tammitie 75"
    },
    "deliveryLctn": {
        "companyName": "Tommin korjaamo Oy",
        "address": "Pasila 75"
    }
},
```


Esimerkkikoodi 5. JSON-tiedosto.

Ohjelmistossa tarvittavat rajapinnat ovat seuraavanlaiset:

Päivittäisten kuljetustoimeksiantojen haku:	GET /deliveries
Luo uusi kuljetustoimeksianto:	POST /deliveries
Muokkaa kuljetustoimeksiantoa:	PUT /deliveries/{id}
Poista kuljetustoimeksianto:	DELETE /deliveries/{id}
Hae asiakkaat:	GET /companies
Luo uusi asiakas:	POST /companies
Muokkaa asiakasta:	PUST /companies/{id}
Poista asiakas:	DELETE /companies/{id}.

Käyttöliittymä

Käyttöliittymän toteutuksessa oli tarkoitus hyödyntää jo valmiiksi yrityksessä hyväksi todettua kirjaamistapaa. Projekti toteutettiin käyttämällä nykyaikaista korttinäkymää. Kortti luodaan painamalla alaosassa olevaa plusnappia, minkä jälkeen lisätään asiakkaan tiedot ja tavaran nouto- ja toimitusosoite. Tämän lisäksi korttiin voidaan kirjoittaa muuta lisätietoa, esimerkiksi kuljetusohjeet ja tietoa tavarasta, kollien lukumäärä ja koko.

Kortteja voidaan siirtää ja järjestää uudelleen, jotta ajon aikana ei tarvitse selata näkymää edestakaisin löytääkseen seuraavaa kohdetta. Korteissa on myös värikoodaus. Värikoodaus näyttää harmaana kuljetustoimeksiannot, jotka eivät ole vielä kyydissä, mutta ovat tallennettuna järjestelmään. Kun paketti on otettu kyytiin, merkitsee kuljettaja kortin värin keltaiseksi, osoittamaan kyydissä olevaa aktiivista toimeksiantoa. Kun kuljetus on suoritettu alusta loppuun, kuljettaja merkitsee kortin vihreäksi ja näin osoittaa kuljetustoimeksiannon loppuun suoritetuksi. (Kuva 5.)

Kuva 5. Käyttöliittymäsuunnitelman alustava prototyyppi.

3.2 Back end- ja REST-rajapinnan luonti

Aluksi tulee valmistella MongoDB-tietokanta. Tämä tapahtui helposti seuraamalla MongoDB:n omaa dokumentaatiota (21), jossa käydään läpi tietokannan asennus valittuun käyttöjärjestelmään. Lyhyesti asennus Mac-laitteelle tapahtui seuraavilla komennoilla terminaalissa:

lataus koneelle:

```
brew tap mongodb/brew
```

asennus koneelle:

```
brew install mongodb-community@4.4
```

tietokannan käynnistys:

```
brew services start mongodb-community@4.4
```

3.3 Back endin luonti Golla

Seuraavaksi tehtiin Go-kielen käyttöönotto. Tämä tapahtui aiemmin läpi käytyjen ohjeiden mukaisesti. Projektin alussa määritetään projektikansio. Projektikansion sisään tulee aluksi luoda go.mod-tiedosto. Tiedostoa tarvitaan määrittelemään moduulien moduulipolut, juurikansion käyttämät tuontipolut ja riippuvuuksien tarpeet. Niitä tarvitaan koontiversion onnistuneeseen rakentamiseen. Go.mod-tiedoston luonti tehdään projektin juurikansiossa seuraavalla komennolla:

```
go mod init my-rest-api
```

Go.mod-tiedoston luonnin jälkeen projektikansion juureen luodaan main.go-tiedosto. Esimerkkikoodissa 6 on main.go-tiedosto, jonka alussa on import-osio ja tähän osaan tuodaan riippuvuuksia sekä projektin tiedostopolkuja, kuten monissa muissakin ohjelmointikielissä. (18.)

```
import (
    "delivery-handler/helper"
    "delivery-handler/models"
)
```

Esimerkkikoodi 6. main.go import-osio.

Yhdistäminen MongoDB-tietokantaan

Juurikansioon luodaan uusi kansio nimeltä helper ja kansion sisälle uusi tiedosto nimeltä connection.go. Connection.go huolehtii projektin yhdistämisestä back endin ja tietokannan välillä. Esimerkkikoodissa 7 käydään läpi projektin yhdistäminen tietokantaan. ConnectDB-funktion alussa määritetään tietokannan osoite, yhdistäminen tietokantaan ja mahdolliset virheilmoitukset, ja onnistuneesta yhdistämisestä palautetaan collection-niminen tietokanta. GetError-funktio hoitaa mahdollisten virheilmoitusten viestinnän ja hallinnan. (Esimerkkikoodi 7.)

```go
func ConnectDB() *mongo.Collection {

    clientOptions := options.Client().ApplyURI("mongodb://localhost:27017")

    client, err := mongo.Connect(context.TODO(), clientOptions)

    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Connected to MongoDB!")

    collection := client.Database("go_rest_api").Collection("deliveries")

    return collection
}

type ErrorResponse struct {
    StatusCode   int    `json:"status"`
    ErrorMessage string `json:"message"`
}

func GetError(err error, w http.ResponseWriter) {

    log.Fatal(err.Error())
    var response = ErrorResponse{
        ErrorMessage: err.Error(),
        StatusCode:   http.StatusInternalServerError,
    }

    message, _ := json.Marshal(response)

    w.WriteHeader(response.StatusCode)
    w.Write(message)
}
```

Esimerkkikoodi 7. connection.gon tietokantaan yhdistäminen.

Main.go-tiedostossa yhdistäminen tietokantaan aloitetaan esimerkkikoodin 8 osoittamalla tavalla.

```
var collection = helper.ConnectDB()
```

Esimerkkikoodi 8. Komento, jolla aloitetaan yhdistäminen tietokantaan.

Nyt terminaaliin ilmestyy teksti ”Connected to MongoDB!”, kun projektin juuri-kansiossa suoritetaan terminaalikomento:

```
go run main.go
```

Tietokantataulujen määrittäminen

Kun yhteys MongoDB-tietokantaan on luotu, voidaan siirtyä rakentamaan tietokantatauluja. Tätä varten projektin juurikansioon luodaan models-kansio ja kansion sisään luodaan uusi tiedosto nimeltä models.go. Tämä tiedosto sisältää kaikki tarvittavat tietueet tietokantaa varten. Kuten sivulla 19. kuvatussa tietokantamallissa näytettiin, luodaan tiedostoon tietueet Delivery ja Company. Koodissa näytetään myös, miten Delivery-tietueessa käytetään kaksi kertaa Company-tietuetta, kuten aiemmin tehdyissä tietokantamäärittelyissä ja esimerkkikoodissa 9 tulee ilmi. Tietueet määrittelevät jo aiemmin kuvattujen tietokantataulujen rakenteen.

```
type Delivery struct {
    ID           primitive.ObjectID `json:"_id,omitempty" bson:"_id,omitempty"`
    Date         string             `json:"date,omitempty" bson:"date,omitempty"`
    Info         string             `json:"info,omitempty" bson:"info,omitempty"`
    Delivered    bool               `json:"delivered" bson:"delivered,omitempty"`
    Payer        bool               `json:"payer" bson:"payer,omitempty"`
    PickupLctn   *Company           `json:"pickupLctn" bson:"pickupLctn,omitempty"`
    DeliveryLctn *Company           `json:"deliveryLctn" bson:"deliveryLctn,omitempty"`
}

type Company struct {
    companyID   primitive.ObjectID `json:"_id,omitempty" bson:"_id,omitempty"`
    Companyname string             `json:"companyName,omitempty" bson:"companyName,omitempty"`
    Address     string             `json:"address,omitempty" bson:"address,omitempty"`
}
```

Esimerkkikoodi 9. Models.go-tiedosto, joka sisältää tietokantarakenteen.

REST-rajapinnat

Kun yhteys tietokantaan on varmistettu ja tietueet rakennettu models.go-tiedostoon, on aika tehdä kontrolleri REST-rajapinnoille. Esimerkkikoodissa 10 toteutetaan ensimmäinen rajapintafunktio main.go-tiedostoon. Funktion tarkoitus on luoda uusi Delivery-tietue tietokantaan.

Esimerkkikoodin 10 funktiossa otetaan vastaan JSON-objekti ja tämän jälkeen luodaan uusi delivery-niminen tietuemalli. Tämän jälkeen luodaan uusi dekoodausohjelma ja juuri saapunut objekti dekoodataan sovellukselle ymmärrettävään muotoon. Kun tämä on tehty, tietue laitetaan aiemmin tehtyyn delivery-tietuemalliin. Virheiden varalta funktio sisältää myös virheidenhallintaan aiemmin rakennetun GetError-metodin.

```
func CreateDelivery(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    var delivery models.Delivery

    _ = json.NewDecoder(r.Body).Decode(&delivery)

    result, err := collection.InsertOne(context.TODO(), delivery)

    if err != nil {
        helper.GetError(err, w)
        return
    }

    json.NewEncoder(w).Encode(result)
}
```

Esimerkkikoodi 10. Funktio, joka luo projektissa uuden tietueen tietokantaan.

Edellä kuvatun funktion lisäksi main.go-tiedostoon tulee luoda vielä seuraavat asiat main-funktion sisälle: Aluksi main-funktiossa luodaan uusi reititin ja määritellään, mikä funktio ajetaan minkäkin REST-kutsun tullessa. Juuri kuvattua funktiota kutsutaan siis esimerkkikoodissa 11 näkyvällä komennolla.

```
r.HandleFunc("/api deliveries", createDelivery).Methods("POST").
```

Esimerkkikoodi 11. CreateDelivery-funktion ohjaaminen.

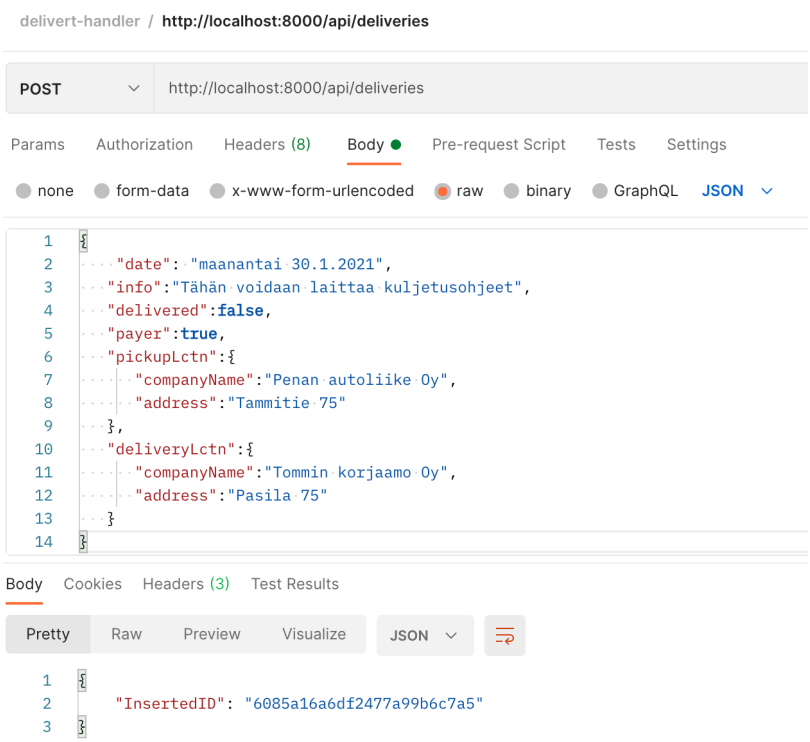
Main-funktion lopussa määritellään vielä portti, jota sovellus kuuntelee. Tässä tapauksessa määritettiin portiksi 8000 ja lopullinen osoite lokaalissa projektissa on http://localhost:8000. (Esimerkkikoodi 12.)

```
func main() {
    r := mux.NewRouter()

    r.HandleFunc("/api/deliveries", getDeliveries).Methods("GET")
    r.HandleFunc("/api/deliveries/{id}", getDelivery).Methods("GET")
    r.HandleFunc("/api/deliveries", createDelivery).Methods("POST")
    r.HandleFunc("/api/deliveries/{id}", updateDelivery).Methods("PUT")
    r.HandleFunc("/api/deliveries/{id}", deleteDelivery).Methods("DELETE")

    log.Fatal(http.ListenAndServe(":8000", r))

}
```

Esimerkkikoodi 12. Main-funktio.

Tämän jälkeen sovellus käynnistetään uudestaan ja voidaan todeta Edellä tehdyn funktion toimivuus esimerkiksi Postman-sovelluksen avulla. Kuten kuvassa 6 näkyy, sovellukselle on luotu POST-kutsu käyttäen http://localhost:8000/api/deliveries-polkua aiemmin määritettyjen tietojen pohjalta. Postman-sovellus antaa onnistuneesta tietojen lisäyksestä varmistuksen tulostamalla seuraavan:

```
"InsertedID": "6085a16a6df2477a99b6c7a5".
```



Kuva 6. Postman-sovellus.

Loput rajapintafunktiot luodaan esimerkkikoodin 10 tapaan, mutta niitä ei tässä käydä sen tarkemmin läpi. Lyhykäisyydessään tässä käytiin läpi esimerkki prosessista, miten työssä rakennettiin back end -sovellus käyttäen Go-ohjelmointikieltä.

3.4 Back end -ohjelman luonti Java Spring Boot -ohjelmistokehyksellä

Seuraavaksi aloitettiin back end -ohjelman koodaaminen Java Spring Bootilla. Spring Boot Back end -sovelluksen luonti aloitettiin sivulla 7 kuvatulla tavalla ja Spring initialzr -työkalussa valittiin seuraavat kohdat:

- project: Maven Project
- language: Java
- spring Boot: 2.4.5
- packaging: Jar

- java: 16.

Näiden lisäksi valittiin tarvittavat lisäosariippuvuudet, jonka jälkeen pom.xml-tiedosto näytti riippuvuuksien osalta kuvan 7 mukaiselta. Riippuvuuksiksi tarvittiin spring-boot-starter-data-mongodb, MongoDB-tietokantaa varten. Riippuvuudet spring-boot-starter-web ja spring-boot-starter-test tulevat projektin luonnissa mukana oletuksena.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Kuva 7. pom.xml-tiedosto.

Tietokantataulujen määrittäminen

Tietokantoja määritettäessä aluksi luodaan projektiin tietokantataulut. Tämä tehdään luomalla projektiin uusi Java-tiedosto nimeltä Delivery.java. Tämä tiedosto sisältää vain Delivery-mallin. Jätän Company-mallin esittämättä tässä, sillä kuten esimerkkikoodista 13 näkyy, on jo pelkkä Delivery-malli aika laaja, vaikka kuvakaappauksesta on rajattu melkein puolet koodista ulos. Tietomallista kuitenkin selviää, miten malli luodaan käyttämällä Spring Boot -ohjelmistokehystä.

```java
@Document(collection = "deliveries")
public class Delivery {
    @Id
    private String id;

    private String date;
    private String info;
    private boolean delivered;
    private boolean payer;

    @Autowired
    private Company company;

    public Delivery() {

    }

    public Delivery(String date, String info, boolean delivered, boolean payer, Company company) {
        this.date = date;
        this.info = info;
        this.delivered = delivered;
        this.payer = payer;
        this.company = company;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    public boolean isPayer() {
        return payer;
    }

    public void setPayer(boolean payer) {
        this.payer = payer;
    }

    public String getDate() {
        return date;
    }

    public void setDate(String date) {
        this.date = date;
    }
```

Esimerkkikoodi 13. Delivery-malli.

Delivery.java-tiedosto sisältää tiedot Delivery-mallista, ja @autowired injektoi Company-mallin mukaan objektiin. Jokaiselle muuttujalle on osoitettu asetus- ja noutofunktiot, kuten esimerkiksi esimerkkikoodissa 14 on nähtävissä.

```
public void setId(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }
```

Esimerkkikoodi 14. Asetus- ja noutofunktiot.

Yhdistäminen MongoDB-tietokantaan

Tietokantataulujen luonnin jälkeen päästään yhdistämään Spring Boot -sovellus MongoDB-tietokantaan. Aluksi luodaan uusi tietolähdekäyttöliittymätiedosto nimeltä DeliveryRepository.java. Tiedostoon tulee tässä vaiheessa vain esimerkkikoodissa 15 nähtävä teksti. Tiedostossa luodaan DeliveryRepository, joka laajentuu MongoRepositoryyn nimeltä Delivery.

```
import org.springframework.data.mongodb.repository.MongoRepository;

public interface DeliveryRepository extends MongoRepository<Delivery, String> {

}
```

Esimerkkikoodi 15. DeliveryRepository.java-tiedosto.

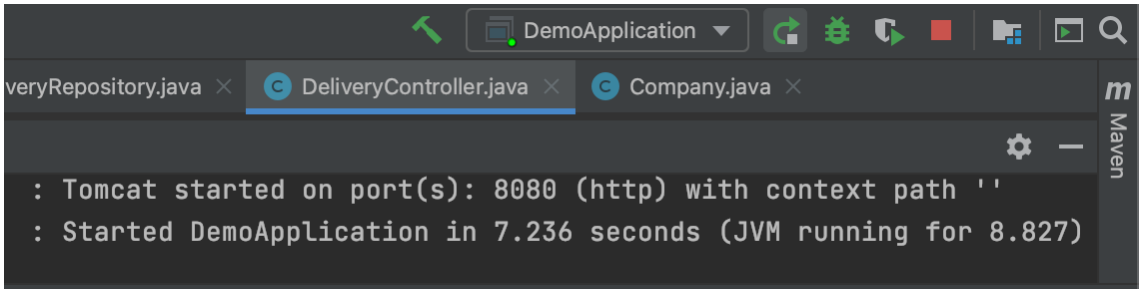
Kun DeliveryRepository on luotu, tarvitaan vielä kontrolleri hoitamaan REST-rajapintojen kutsut. Tätä varten projektiin luodaan tiedosto nimeltä DeliveryController.java ja lisätään tiedostoon esimerkkikoodin 16 mukaiset tiedot.

Kontrollerissa määritellään tässä kohtaa seuraavat asiat: Esimerkkikoodissa 16 kerrotaan aluksi, mikä on sovelluksen verkko-osoite ja portti. Portti on tässä tapauksessa eri kuin Go-sovelluksessa, sillä se vähentää tässä vaiheessa väärinymmärryksiä testattaessa rajapintojen toimivuutta. @RestController kertoo sovellukselle tämän olevan REST-verkkopalvelu, ja @RequestMapping määrittää URI-verkkopäätteen. Näiden lisäksi luokassa DeliveryController injektoidaan Delivery-tietolähde.

```
@CrossOrigin(origins = "http://localhost:8080")
@RestController
@RequestMapping("/api")
public class DeliveryController {

    @Autowired
    DeliveryRepository deliveryRepository;

}
```

Esimerkkikoodi 16. DeliveryController.java-tiedosto.

Tässä vaiheessa kaikki Spring Boot -sovellukseen tehdyt muutokset mahdollistavat yhdistämisen MongoDB-tietokantaan. Tämä voidaan todeta painamalla IntelliJ IDEA-sovelluksen yläkulmassa olevaa run application -nappia, ja onnistuneesta yhdistämisestä alakulmaan tulee teksti ”Started DemoApplication in 7.236 seconds”, kuten kuvassa 8 näkyy.



Kuva 8. IntelliJ IDEA ja sovelluksen käynnistys.

REST-rajapinnat

Kun yhteys tietokannan ja Spring Boot -sovelluksen välille on luotu ja toiminta varmistettu, päästään tekemään REST-rajapintafunktiot kontrollerille. Kuten Go-sovelluksessa läpi käydyssä esimerkissä, myös Spring Bootilla toteutetaan ensimmäinen rajapintafunktio. Funktion tarkoitus on luoda uusi Delivery-tietue tietokantaan. Tämä tapahtuu esimerkkikoodin 17 mukaisesti.

JSON-objekti otetaan vastaan creacteDelivery-funktiossa. Kuten Go-sovelluksessa, aluksi luodaan delivery-olio ja olioon syötetyt tiedot tallennetaan tietokantaan. Tallennuksen onnistuttua saadaan vastaus tietueen tallentamisesta. Jos tallennus ei onnistu esimerkiksi vääränlaisen JSON-objektin takia, funktio huolehtii tässä tapauksessa palauttamalla virheilmoituksen.

```
@PostMapping("/deliveries")
public ResponseEntity<Delivery> createDelivery(@RequestBody Delivery delivery) {
    try {
        Delivery _delivery = deliveryRepository.save(new Delivery(
                delivery.getDate(),
                delivery.getInfo(),
                 delivered: false,
                delivery.isPayer(),
                delivery.getCompany()
                ));
        return new ResponseEntity<>(_delivery, HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Esimerkkikoodi 17. Funktio, joka luo projektissa uuden tietueen tietokantaan.

Edellä kuvattiin tapa, miten POST-rajapintakutsu toteutetaan käyttäen Spring Boot -ohjelmistokehystä. Kuten Go-sovelluksen osalta, tähän ei oteta enempää esimerkkejä muista rajapintakutsuista, koska ne ovat samankaltaisia.

Nyt Spring Boot -sovellus voidaan käynnistää uudelleen ja todeta juuri tehdyn POST-rajapintakutsun toimivuus Postmanilla. Sitä ei käydä tarkemmin enää läpi, sillä toimintatapa on täysin sama kuin aiemmin Go-sovelluksessa kuvattu tapa. Ainut ero tällä hetkellä on verkko-osoitteen eri portti kuin Go-sovelluksessa.

3.5 Käyttöliittymä

Kun back end -sovellukset ja tietokannat ovat kunnossa, tarvitaan projektiin vielä käyttöliittymä. Käyttöliittymän käyttöönottaminen tehtiin projektissa seuraavasti.

React-sovelluksen käyttöönotto

Ennen React-sovelluksen luomista tarkistetaan koneen Node-versio. Node.js on avoimeen lähdekoodiin pohjautuva JavaScript-koodin ajoympäristö. Node on myös alustariippumaton. (22.) Node-versio 15.6.0 oli jo asennettuna koneella, ja tämän todentaminen tapahtui Mac-ympäristössä seuraavalla terminaalikomennolla:

```
node -v
```

Seuraavaksi suoritetaan React-sovelluksen luonti terminaalissa seuraavalla komennolla:

```
npx create-react-app <Sovelluksen nimi>
```

Projektin nimeksi tuli "delivery-handler-frontend”. Tämän jälkeen projekti käynnistetään delivery-handler-frontend-kansiossa komennolla

```
npm start
```

React-sovellus käynnistyy terminaaliin ja avaa selaimeen osoitteen http://localhost:3000/, joka näyttää sovelluksen käynnistyneen onnistuneesti.

React-sovelluksen toteutus

Yhteys tietokannan, back endin ja käyttöliittymän välillä.

Sen jälkeen, kun React-sovellus on saatu käyttöönotetuksi, vuorossa on yhteyden muodostaminen aikaisemmin toteutetun back end -sovelluksen kanssa.

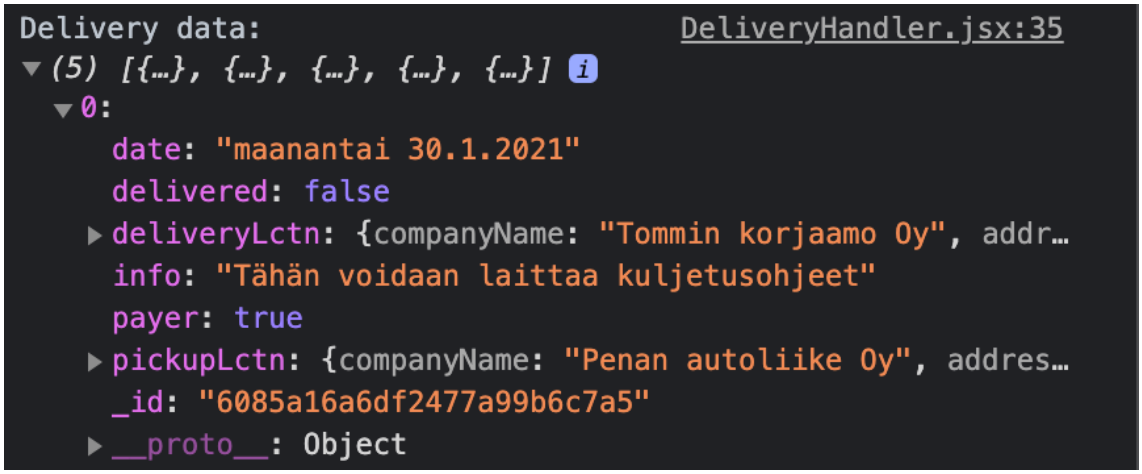
Aluksi luotiin DeliveryHandler.jsx-niminen tiedosto (esimerkkikoodi 18). Tiedostossa käydään aluksi läpi seuraavat asia: Sovellus käynnistyttyään tarkistaa, onko deliveries-taulukko tyhjä. Sovelluksen todettua taulukon tyhjäksi kutsuu sovellus getDeliveryData-funktiota, joka käynnistää GET-kutsun back end -sovellukselle. Onnistuneen kutsun jälkeen tallennetaan Delivery-taulukon tiedot

sovelluksen muistiin. Sitten käytettiin console.log-komentoa toimivuuden testaamiseen.

```
const DeliveryHandler = () => {

    const baseURL = process.env.ENDPOINT || 'http://localhost:8000/api';
    const [deliveries, setDeliveries] = useState([])

    useEffect(() => {
        if (!deliveries.length) {
            getDeliveryData()
        }
    })

    const getDeliveriesFromApi = async () => {
        try {
            const response = await fetch(`${baseURL}/deliveries`);
            return response.json();
        } catch (error) {
            console.error(error);
        }

        return {};
    };

    const getDeliveryData = async () => {

        const [deliveryData] = await Promise.all([getDeliveriesFromApi()]);

        setDeliverydata(deliveryData)
    }

    const setDeliverydata = (deliveryData) => {

        if (deliveryData) {
            console.log('Delivery data:', deliveryData)
            setDeliveries(deliveryData)

        } else {
            console.log('unable to fetch deliveries')
        }

    };
```

Esimerkkikoodi 18. DeliveryHandler.jsx-tiedosto.

Seuraavaksi käydään toteamassa kuvassa 9 näkyvän selaimen inspect-toiminnon avulla, että tietokannan back endin ja käyttöliittymän välillä tieto kulkee moitteettomasti.



Kuva 9. Chrome-selaimen inspect-työkalu.

Tämän jälkeen sovellukseen luotiin myös loput REST-rajapintakutsut. Tässä käytiin lävitse GET-kutsu, koska back end -sovelluksissa käsiteltiin enemmän POST-kutsua.

3.6 Projektin seuraavat askeleet

Projekti on vielä insinöörityöraporttia kirjoitettaessa keskeneräinen, mutta jatkosuunnitelma projektin toteutukselle on seuraavanlainen:

Projektin edetessä huomattiin, että tietokanta tarvitsee ainakin vielä yhden tietokantataulun lisää tarjotakseen sovelluksen suunnitellun toimivuuden. Sovelluksesta puuttuu vielä täysin tietoturva, ja näin ollen käyttäjätietojen lisääminen tietokantaan tulee toteuttaa. Tietoturvan takaamiseksi turvallisten käyttäjän oikeuksien tarkistamisen ja vahvistamisen tulisi olla kunnossa.

Käyttöliittymän visuaalinen muotoilu vaatii vielä työstämistä, ja käyttöliittymän React-sovelluksen toteutus on vielä kesken. Tämän lisäksi yksikkötestien kirjoittaminen on vielä toteuttamatta. Sovellusten valmistuttua ne on tarkoitus siirtää Docker-kontteihin ja tämän jälkeen siirtää kokonaisuus palvelimille.

3.7 Spring Bootin ja Gon vertailu sovellusten toteutuksen jälkeen

Spring Boot ja Go ovat kumpikin C-perheeseen pohjautuvia ohjelmointikieliä, mutta kielten käytössä on huomattavissa paljon eroavaisuuksia ja myös paljon samankaltaisuuksia.

Ensimmäisenä asiana sovelluksia toteutettaessa huomio kiinnittyi niin kutsutun vakiotekstin määrän eroihin. Vakioteksti on tekstiä, joka esiintyy koodissa useampaan kertaan. Vaikka Spring Bootin myötä Javasta on vähennetty vakiotekstin määrää jo puhtaaseen Javaan verrattuna, esiintyy siinä kuitenkin huomattavasti enemmän vakiotekstiä kuin Gossa.

Riippuvuuksien määrä sovelluksessa oli myös isompi Spring Bootissa kuin Gossa. Myös riippuvuuksien tekstimäärän pituus Spring Bootissa oli isompi ja sekavampi kuin Gossa.

Ohjelmointiympäristöjen rakentaminen erosi kielissä huomattavasti. Spring Bootin ohjelmointiympäristön tekemiseen meni huomattavasti enemmän aikaa kuin Gossa. Jos tarkoitus on tehdä nopea testiprojekti, johon tarvitaan back end -ympäristöä, itse lähtisin tekemään projektin Golla.

Spring Boot on kuitenkin nopeakäyttöinen asennuksen jälkeen ja projektin luominen sitä käyttäen melko yksinkertaista. Spring Bootin käyttö saattaa kuitenkin aiheuttaa hankaluuksia, sillä ilman Spring-sovelluskehyksen osaamista Spring Bootin käyttö alussa oli hieman haastavaa. Springin mukana tulevat automaattiset konfiguraatiot lisäsivät hämmennystä paljon, sillä samaan aikaan taustalla tapahtui paljon asioita automaattisesti.

Koska Spring Bootin ja Java ovat olleet jo markkinoilla pidempään, oli tähän paljon helpompi löytää apua ongelmatilanteissa laajemman käyttäjäkunnan takia. Gon kysyntä kuitenkin jatkuvasti kasvaa, joten Gon tilanne tässä asiassa todennäköisesti paranee ajan myötä.

Gon minimaalinen syntaksi peittoaa Spring Bootin käytettävyydellään. Jo aiemmin mainitun vakiotekstin lisäksi Javan syntaksin takia tekstin määrä kasvaa huomattavan isoksi verrattuna Gohun.

Spring Boot tuntui olevan huomattavasti enemmän MVC-mallia hyödyntävä kieli ja Go taas huomattavasti vapaampi sen suhteen.

Sovellusten ajaminen poikkesi jonkin verran toisistaan. Kun Java Spring Bootin uudelleen käynnistämiseen tarvitsi painaa vain IntelliJ IDEAn yläkulmassa olevaa run-nappia, piti Go erikseen sulkea ja käynnistää uudelleen terminaalikomennolla. Go oli kuitenkin tämän käynnistämisen jälkeen nopeammin toimintavalmis siitä huolimatta.

Sovelluksen siirtäminen pilvipalveluun tehdään Java Spring Bootissa JAR- tai WAR-tiedoston avulla, mikä onnistuu myös IntelliJ IDEAn valikon kautta yhtä nappia painamalla. Gossa tämä hoidetaan taas terminaalikomennolla.

4 Yhteenveto

Insinöörityötä aloittaessani tiesin Spring Bootin ja Gon olevan toisiinsa verrattuina hyvin erilaiset. Toteuttaessani sovelluksia tulin kuitenkin huomanneeksi vasta, miten eri maailmasta ne ovat. Gon hyvin nykyaikainen ja karsittu syntaksi ja minimalistinen projektin luonti vetosivat jo ennen vertailun aloittamista ja vahvistivat kantaani hieman kankeaan Spring Bootiin verrattuna.

Kehitystyö Golla tuntui olevan hyvin helppoa ja nopeaa, kun taas Spring Bootin dokumentaatiota selaillessa huomasin tietäväni paljon vähemmän mitä syvemmälle dokumentaatioon etenin. Sovellus Spring Bootilla toteutui, vaikkei ihan kaikkia Springin automaattikonfigurointien takana tapahtuvia asioita vielä sisäistänytkään.

Insinöörityölle isoin antamani tavoite oli useampien back end -kielien oppiminen ja ymmärtäminen. Työni puolesta en ollut päässyt kehittämään osaamistani kuin front end -puolella ja back end oli jäänyt hieman sen varjoon. Insinöörityössä back end -kielinä käytettyjen lisäksi minulla on hieman osaamista JavaEEstä ja Node.js:stä, joten uusien työkalujen oppiminen oli tervetullutta.

Pääsin mielestäni hyvin tavoitteeseen, ja nyt minulla on laajempi osaaminen tulevaisuuden työmahdollisuuksia varten. Loppujen lopuksi kahteen kieleen tutustuminen näin lyhyessä ajassa jätti laajemman osaamisen hieman vajaaksi. Tämä olisi ollut ratkaistavissa ottamalla vain toinen vertailussa mukana olleista kielistä insinöörityön aiheeksi.

Loppujen lopuksi vertailussa tulin siihen tulokseen, että aion toteuttaa projektin lopullisen back end -sovelluksen käyttämällä Gota, koska Go tarjosi mielestäni paremman ohjelmistokokemuksen ja uskon kielen osaamisen olevan tulevaisuudessa parempi valinta, koska sen kysyntä työmarkkinoilla kasvaa jatkuvasti.

Lähteet

1. Why Spring? 2021. Verkkoaineisto. Spring. https://spring.io/why-spring. Luettu 8.3.2021.

2. Maple, Simon & Binstock, Andrew. 2018. JVM Ecosystem Report 2018 – About your Platform and Application. Verkkoaineisto. https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/. Luettu 8.3.2021.

3. Developer survey. 2020. Verkkoaineisto. Stackoverflow. https://insights.stackoverflow.com/survey/2020. Luettu 8.3.2021.

4. Baker, Jordan. 2019. Spring and Spring Boot Frameworks: A Brief History. Verkkoaineisto. https://dzone.com/articles/history-of-spring-framework-spring-boot-framework. Luettu 15.3.2021.

5. Mulders, Michiel. 2019. What is Spring Boot? Verkkoaineisto. https://stackify.com/what-is-spring-boot/. Luettu 15.3.2021.

6. Spring. 2021. Verkkoaineisto. Spring. https://spring.io/. Luettu 15.3.2021.

7. Bos, Spencer. 2020. Java Basics: What is Spring Boot? Verkkoaineisto. https://www.jrebel.com/blog/what-is-spring-boot. Luettu 25.3.2021.

8. Javin, Paul. 2018. Top 5 Spring Boot Features Java Developers Should Know. Verkkoaineisto. https://dzone.com/articles/top-5-spring-boot-features-java-developers-should. Luettu 22.3.2021.

9. Pros and cons of using Spring Boot. 2020. Verkkoaineisto. Scand. https://scand.com/company/blog/pros-and-cons-of-using-spring-boot/. Luettu 8.3.2021.

10. Spring Initialzr. 2021. Verkkoaineisto. Spring. https://start.spring.io/. Luettu 17.3.2021.

11. Kincaid, Jason. 2009. Google's GO: A New Programming Language That's Python Meets C++. Verkkoaineisto. https://techcrunch.com/2009/11/10/google-go-language/. Luettu 30.3.2021.

12. Bai, Yaroslav. Best practices: Why use Golang for your project. Verkkoaineisto. https://uptech.team/blog/why-use-golang-for-your-project. Luettu 2.4.2021.

13. Go versus Java fastest programs. Verkkoaineisto. Benchmarks Game. https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/go.html. 13.3.2021.

14. Osadchiy, Victor. 2021. Why use the Go language for your project? Verkkoaineisto. https://yalantis.com/blog/why-use-go/. 2.3.2021.

15. State of software engineers. 2021. Verkkoaineisto. Hired. https://hired.com/state-of-software-engineers. Luettu 2.3.2021.

16. Pi, Ke. 2018. Advantages and disadvantages of Golang. Verkkoaineisto. https://www.pixelstech.net/article/1541901293-Advantages-and-disadvantages-of-GoLang. Luettu 3.4.2021.

17. Sidorenko, Irina. 2019. Should I Go? The pros and cons of using Go programming language. Verkkoaineisto.https://hackernoon.com/should-i-go-the-pros-and-cons-of-using-go-programming-language-8c1daf711e46. Luettu 14.4.2021.

18. Golang Tutorial: Get started with Go. Verkkoaineisto. Golang. https://golang.org/doc/tutorial/getting-started. Luettu 14.3.2021.

19. Novikau, Aliaksei. 2019. When to Use Go vs. Java | One Programmer's Take on Two Top Languages. Verkkoaineisto. https://spiralscout.com/blog/when-to-use-go-vs-java-one-programmers-take-on-two-top-languages Luettu 25.3.2021.

20. DB-Engines Ranking. 2021. Verkkoaineisto. DB-Engines. https://db-engines.com/en/ranking. Luettu 20.3.2021.

21. Install MongoDB community edition. Verkkoaineisto. MongoDB. https://docs.mongodb.com/manual/administration/install-community/. Luettu 8.3.2021.

22. Nodejs. Verkkosivusto. Nodejs. https://nodejs.org/en/. Luettu 15.3.2021.