Opinnäytetyö AMK

Tieto- ja viestintätekniikka

2021

Toni Kainulainen

REST-RAJAPINTA KIMPPAKYYTI-SOVELLUKSELLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ AMK | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2021 | 22 sivua

Toni Kainulainen

REST-RAJAPINTA KIMPPAKYYTISOVELLUKSELLE

Opinnäytetyössä rakennettiin REST-rajapinnan toteuttava palvelinsovellus kuvitteelliselle kimppakyytisovellukselle käyttäen Spring Boot -ohjelmointikehystä. Rakennetulle rajapinnalle kirjoitettiin automatisoituja testejä käyttäen JUnit-kirjastoa ja tutustuttiin Postman-rajapintatestausalustaan. Työn avulla tunnistettiin sovellustestien tärkeys sovelluskehitystyössä, sisäistettiin Spring Boot -ohjelmointikehyksen käyttö ja selvitettiin REST-rajapinnan määritelmä. Työssä tehtyä REST-rajapintaa voitaisiin tulevaisuudessa käyttää oikean kimppakyytisovelluksen tekoon.

REST-rajapinta ohjelmoitiin käyttäen Spring Boot -ohjelmointikehystä. Ohjelmointityön aikana tehtiin lisäksi yksikkötestejä, joiden tarkoitus on varmistaa, että rajapinta pintaa palauttaa oikeat tiedot verkon yli tehdyille pyynnöille.

Työn tavoitteena oli opetella sovelluskehitystä ja sovelluskehitystyökalujen käyttöä.

ASIASANAT:

REST, ohjelmointirajapinta, automatisoitutesti, Spring Boot

BACHELOR´S / MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and communcation technologies

2021 | 22 pages

Toni Kainulainen

REST-API FOR RIDE SHARE APPLICATION

This thesis is about building a REST-API server application for an imaginary ride sharing application using the Spring Boot programming framework. The API was build by programming unit tests using Junit test library and learned about Postman API testing platform This thesis tries to recognize the importance of automated testing in software development, to get deeper understanding of the Spring Boot programming framework and to explain REST-API. The REST API build in this thesis, could be used to with a real ride sharing mobile application.

The REST-API was programmed using Spring Boot framework. During the programming, some unit tests were written to make sure that the API returns correct values to a request called over internet.

This thesis was written to learn about software development and software development tools.

KEYWORDS:

REST, API, automated test, Spring Boot

SISÄLTÖ

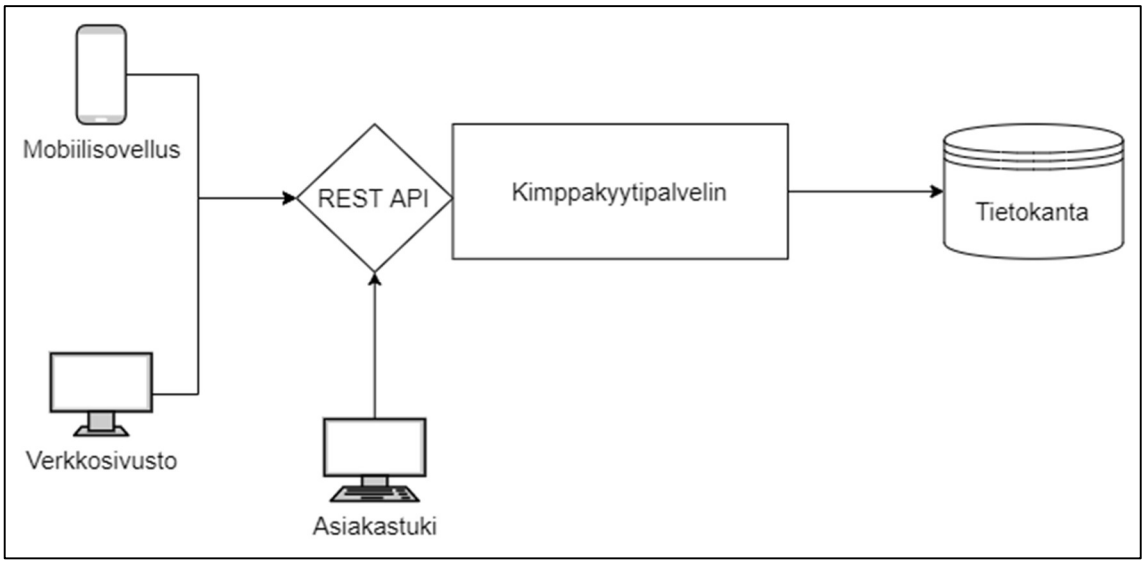
KUVAT

KOODIT

1 JOHDANTO

REST-rajapintojen avulla ohjelmistotuotannosta voidaan tehdä vapaata ohjelmointikielestä tai sovelluskehyksestä riippumatta. REST-rajapinnat toimivat HTTP-kutsujen avulla ja nämä kutsut on määritelty yhdellä tapaa.

Tässä opinnäytetyössä rakennetaan REST-rajapinta automatisoiduilla testeillä teemana kimppakyytien varaus ja julkaisu. Rajapintakokonaisuuden ohessa opinnäytetyössä pyritään myös selittämään, mikä REST-rajapinta on ja mikä on sen merkitys ohjelmistotuotannossa, sekä kerrotaan lyhyesti ohjelmistotestauksesta ja sovellusarkkitehtuurista. Tavoiteltu REST-rajapinta (Kuva 1) rakennetaan käyttäen Java ohjelmointikielellä tehtyä Spring Boot -ohjelmointikehystä.



Kuva 1. Opinnäytetyössä rakennettavan REST-rajapinnan rooli kuvitellussa tuotantokokonaisuudessa.

Rakennettavan rajapinnan testaus suoritetaan kirjoittamalla JUnit-yksikkötestäjä, jotka ovat Java-virtuaalikoneella toteutettavia pieniä ohjelmia projektin sisällä, ja Postman-alustaa, joka on tarkoitettu muun muassa ohjelmistorajapintojen testaukseen, dokumentointiin ja luontiin.

2 REST-SOVELLUSARKKITEHTUURIMALLI

2.1 REST-sovellusarkkitehtuurimalli

REST eli Representational State Transfer on sovellusarkkitehtuurimalli, joka on ihanne modernien web-applikaatioiden vuorovaikutuksessa. REST-arkkitehtuurista on tullut perusta nykyaikaiselle web-arkkitehtuurille. REST-arkkitehtuuri antaa käyttöön ohjaavia periaatteita, joiden avulla jo olemassa olevista web-arkkitehtuureista voidaan tunnistaa niiden heikkoudet ja niiden laajennukseen liittyvät virheet ennen käyttöönottoa. (Fielding & Taylor 2000.)

REST sisältää toistensa kanssa yhteensopivia arkkitehtuurisia rajoituksia, joiden tarkoitus on minimoida viivettä ja verkkoviestintää. Samanaikaisesti rajoitukset pyrkivät maksimoimaan verkkosovellusten itsenäisen toiminnan ja skaalautuvuuden. (Fielding & Taylor 2000.)

Web-kehityksessä applikaatioiden komponentit levittäytyvät usein usean organisaation rajojen yli (Fielding & Taylor 2000), mikä tarkoittaa että web-applikaatiot voivat kasvaa hyvinkin suuriksi kokonaisuuksiksi. Siksi web-applikaatiojärjestelmien tulisikin valmistautua asteittaisiin muutoksiin, joissa vanhat toteutukset pystyisivät elämään uusien laajennusten kanssa ilman, että uusien laajennusten toiminta kärsisi vanhoista toteutuksista (Fielding & Taylor 2000).

2.2 REST-rajapinnan kypsyystasot

REST-rajapinnat on jaoteltu RMM-kypsyystasoihin (Richardson Maturity Model). RMM on Leonard Richardsonin kehittämä (REST API Tutorial) tapa arvoida REST-toteutukset neljään tasoon 0-3, jossa neljännellä tasolla REST-toteutus toteuttaa täysin REST-rajapinnan periaatteet. Mitä paremmin REST-rajapinta pysyy RMM-tasojen rajoitusten sisällä, sitä korkeampi sen RMM-taso on. (Restcookbook.com.)

RMM-tason 0 palvelua ei pidetä lainkaan REST-rajapintaa toteuttavana. Tällaisessa palvelussa HTTP-protokollan käyttötapaan ei kiinnitetä huomiota ja se on vain käytössä väylänä viestien vastaanottoon ja lähetykseen. Esimerkiksi tason 0 palvelussa sen toiminnallisuus päätellään pyynnön sisällön perusteella. (Hellas (né Vihavainen) & Luukkainen 2019c.)

RMM-tason 1 palveluita käsitellään resursseina. Resursseja kuvataan palvelun osoitteina, joiden avulla resursseja voidaan hakea osoitteena tai osoitteen ja tunnisteen perusteella. Esimerkiksi palvelusta voidaan hakea kirjoja osoitteesta /kirja tai jos palvelusta halutaan hakea tiettyä kirjaa, niin se voidaan hakea nimi-tunnisteen avulla osoitteesta /kirja/nimi. (Hellas (né Vihavainen) & Luukkainen 2019c.)

RMM-tason 2 palvelu käyttää kuvaavia HTTP-pyyntöjä. Tason 2 palvelussa käytetään resurssien hallintaan HTTP-metodeja GET, POST, PUT ja DELETE. Näitä metodeja voidaan käyttää esimerkiksi resurssin hakuun, resurssin lisäykseen, resurssin muokkaukseen ja resurssin poistoon. Palvelun vastauksien tulee myös kuvailla resurssien hallinnan toimintoja. Esimerkiksi onnistuneesta resurssin lisäyksestä palvelun tulisi palauttaa vastauskoodi 201. Oleellisin asia tasolla 2 on pyyntöjen erottelu sen mukaan, miten palvelin muokkaa dataa vai muokkaako se sitä ollenkaan. (Hellas (né Vihavainen) & Luukkainen 2019c.)

RMM-tason 3 palvelu on yhdistelmä tason 1 ja 2 palveluista. Tason 3 palvelu myös tarjoaa käyttäjälle mahdollisuuden ymmärtää palvelun toiminnallisuutta palvelimen vastausten perusteella. (Hellas (né Vihavainen) & Luukkainen 2019c.)

RMM-tasot eivät kerro mitään sovelluksen laadusta. RMM-tason 3 sovellus ei ole parempi kuin RMM-tason 2 sovellus ja joskus REST-rajapintaa ei edes tarvita. Sovelluksen laatu ja rajapinnan tarve määräytyvät asiakkaan toiveiden ja käytön mukaisesti. (Hellas (né Vihavainen) & Luukkainen 2019c.)

3 SPRING-SOVELLUSKEHYS

Ohjelmistotuotannossa merkittävä osa sovellusten rakentamisesta perustuu valmiiden kirjastojen käyttöön. Sovelluskehykset ja koodikirjastot eivät ole pakollisia sovelluskehityksessä, mutta niiden käyttö nopeuttaa kehitystyötä, sillä niiden ansiosta kaikkea toiminnallisuutta ei tarvitse keksiä itse. Monissa asioissa koodin itsekirjoitus voi olla myös tehottomampaa, kuten vaikka tietosuojauksia tehdessä on suositeltavaa käyttää jatkuvasti päivittyviä kirjastoja ja sovelluskehyksiä.

Spring on sovelluskehys, joka mahdollistaa eri tasoisten sovellushankkeiden rakentamisen helposti ja joustavasti. Spring-sovellukehyksen avulla voi rakentaa ohjelmiston niin harrastekäyttöön kuin liiketoimintaa ajatellen. Spring-sovelluskehys sisältää kaiken, mitä natiivi Java-ympäristö voi vaatia, ja se tukee myös Java virtuaalikoneella toimivia vaihtoehtoisia kieliä Groovya ja Kotlinia. Spring-sovelluskehys vaatii Java ohjelmistokehityspaketin JDK version kahdeksan tai uudemman. (VMware 2020f.)

Spring Boot on Spring-sovelluskehykseen pohjautuva alustustyökalu, jonka tarkoituksena on lyhentää kehityksessä muodostuvaa koodia (Mohamed 2019). Spring Boot -projekti vaatii vain minimaalisen Spring-kokoonpanon, ja sen avulla on helppo kehittää itsenäisiä tuotantotason ohjelmistoja (VMware 2020b).

3.1 Spring Boot -projektin käyttö

Spring -projekti tulee aloittaa Spring Initializr -työkalun avulla start.spring.io-sivulla. Spring Initializr on nopea tapa alustaa projekti kehittäjän haluamilla komponenteilla (VMware 2020c). Komponenttien määrittäminen projektiin onnistuu myös manuaalisesti esimerkiksi käyttämällä Maven-työkalua komponenttien lataamiseen (Hellas (né Vihavainen) & Luukkainen 2019a). Komponenttien latauksen jälkeen koodin kirjoitus voidaan aloittaa.

Suoraviivaisin tapa päästä alkuun Spring Boot -projektissa on luoda main-metodin sisältävä luokka (Koodi 1) ja merkitä se annotaatiolla *@SpringBootApplication* (REST eBook Baeldung). Main-metodin sisältävää luokkaa käytetään Spring Boot -sovelluksen käynnistämiseen ja annotaatioita käytetään sovelluksen kokonaisuuden määrittelyyn. Kun sovellus käynnistetään, niin se etsii annotaatioilla merkittyjä luokkia, joita se lataa käyttöönsä. Tällaisia luokkia voidaan merkitä esimerkiksi *@Controller*-annotaatiolla (Koodi

2) tai monilla muilla erilaisilla annotaatioilla. Ilman omaa sovellusmäärittelyä Spring Boot -sovellus käyttää oletusmäärittelyä. Sovellus käynnistetään kutsumalla *SpringApplication*-luokan run-metodia (Koodi 1). (Hellas (né Vihavainen) & Luukkainen 2019a.)

```
@SpringBootApplication
public class KimppakyytiApplication {
  public static void main(String[] args) {
    SpringApplication.run(KimppakyytiApplication.class, args);
  }
}
```

Koodi 1. *KimppakyytiApplication*-luokka.

Palvelinsovelluksen käynnistyttyä luokat, jotka on merkitty *@Controller*-annotaatiolla, käsittelevät palvelinsovellukselle tulevat pyynnöt selaimen kautta. Spring-sovelluskehys on vastuussa palvelinsovellukselle tulevista pyynnöistä ja Spring ohjaa pyynnöt oikeaan käsittelyyn annotaatioiden avulla. *@Controller*-annotaatiolla merkitty luokka käsittelee saamansa pyynnöt pyynnönkäsittelyannotaatioilla merkityillä metodeilla (Koodi 2). Tälläisiä annotaatioita ovat esimerkiksi *@GetMapping*- ja *@ResponseBody*-annotaatiot. (Hellas (né Vihavainen) & Luukkainen 2019a.)

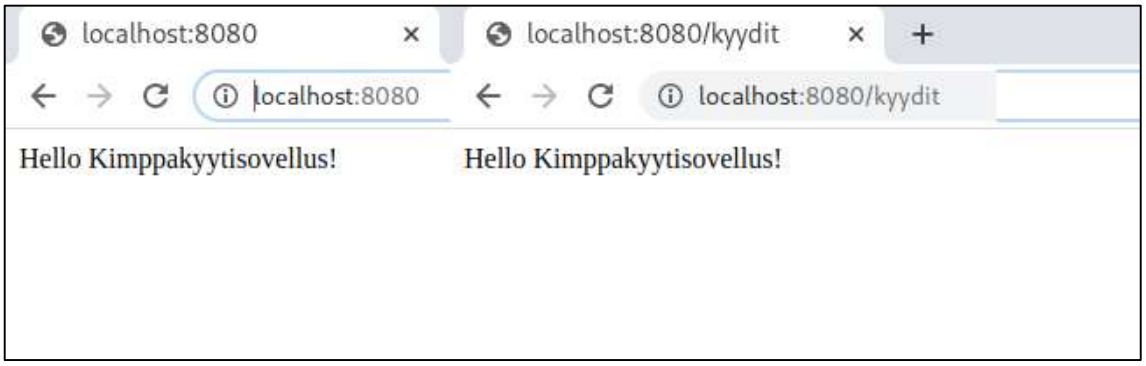
```
@Controller
public class HelloKimppakyyti {
  @GetMapping(”*”)
  @ResponseBody
  public String hello() {
    return ”Hello Kimppakyytisovellus!”;
  }
}
```

Koodi 2. HelloKimppakyyti-luokka.

@GetMapping-annotaatio kertoo Spring-sovelluskehykselle, että kyseisellä annotaatiolla merkitylle metodille ohjataan HTTP-protokollan GET-pyyntöjä. *@GetMapping*-annotaation jälkeen tuleviin sulkeisiin merkitään polkuparametri. Polkuparametri kertoo Spring-sovelluskehykselle, mille metodille selaimen kautta tuleva pyyntö ohjataan. Jos polkuparametri on merkitty asteriskilla niin se tarkoittaa, että kaikki palvelimeen tulevat

GET-pyynnöt ohjataan kyseiseen metodiin. (Hellas (né Vihavainen) & Luukkainen 2019a.)

Polkuparametria voisi ajatella osoitteena, johon käyttäjä yrittää ottaa yhteyttä selaimella. Esimerkiksi jos selaimesta navigoitaisiin sivuston juuriosoitteeseen ja osoitteeseen */kyydit*, niin selaimessa näkyvä tulostus olisi sama (Kuva 2).



Kuva 2. Vastaus kontrollerissa asteriskilla merkittyyn osoitteeseen.

Jos HelloKimppakyyti-luokka sisältäisi osoitteelle */kyydit* *@GetMapping*-annotaatiolla merkityn metodin asteriskiosoitteen lisäksi (Koodi 3), niin selaimessa nähtävä viesti olisi erilainen (Kuva 3).

```
@Controller
public class HelloKimppakyyti {
   @GetMapping(”*”)
   @ResponseBody
   public String hello() {
      return ”Hello Kimppakyytisovellus!”;
   }

   @GetMapping(”/kyydit”)
   @ResponseBody
   public String kyydit() {
      return ”Tältä sivulta näkee kaikki tarjotut kyydit!”;
   }
}
```

Koodi 3. Ohjaus osoitteeseen */kyydit* *HelloKimppakyyti*-luokassa.

Kuva 3. Vastaus osoitteesta */kyydit*.

3.2 Spring Boot @*RestController*

@*RestController*-annotaatio on kontrolleriannotaatio, joka itsessään sisältää annotaatiot @*Controller* ja @*ResponseBody* (VMware 2020g). @*RestController*-annotaatiota käyttävän luokan kaikki polkuja kuuntelevat metodit käyttävät vastauksissaan oletuksena @*ResponseBody*-annotaatiota, joka palauttaa ja vastaanottaa dataa mediatyypiltään *application/json*. Mediatyyppi on valinnainen, mutta oletettu palautusmediatyyppi @*ResponseBody*-annotaatiota käytettäessä on JSON.

Vastaanotettava ja palautettava mediatyyppi voitaisiin vaihtoehtoisesti määritellä @*RequestMapping*-annotaation, kuten @*GetMapping*, avulla. (Hellas (né Vihavainen) & Luukkainen 2019c.) @*RestController*-kontrolleriluokka (Koodi 4) rakennetaan samoin kuin normaali @*Controller*-kontrolleriluokka.

```
@RestController
public class RestKimppakyyti {
  @Autowired
  private KyytiRepository kyytiRepository;

  @GetMapping(”/kyydit”)
  public List<Kyyti> haeKaikkiKyydit() {
    return kyytiRepository.findAll();
  }

  @GetMapping(”/kyydit/{määränpää}”)
  public List<Kyyti> haeKyyditMääränpäänMukaan(
    @PathVariable String määränpää) {
    return kyytiRepository.findByDestination(määränpää);
  }
}
```

Koodi 4. *@RestController*-annotaatiolla varustettu *RestKimppakyyti*-luokka.

@RestController-kontrolleriluokan kirjoituksen jälkeen, jos selaimella navigoitaisiin osoitteeseen */kyydit*, niin palvelin palauttaisi selaimelle JSON-olion (Kuva 4), joka sisältää listan kaikista tietokantaan tallennetuista kyydeistä. Vastaavasti osoitteesta */kyydit/{määränpää}* saataisiin selaimeen vastaukseksi esimerkiksi kaikki kimppakyydit matkalla Helsinkiin.

Kuva 4. Helposti luettavaan muotoon muotoiltu JSON-olio, joka sisältää listan kaikista kimppa-kyydeistä kimppakyytitietokannassa.

JSON-muotoisen datan avulla REST-rajapinnan tarjoavalle palvelimelle on helppo rakentaa asiakaskäyttöliittymä, koska JSON-muotoisen datan käyttö on suoraviivaista JavaScriptin kanssa (Hellas (né Vihavainen) & Luukkainen 2019c). JSON-datalla (JavaScript Object Notation) tarkoitetaan tapaa, jolla JavaScript-oliota kuvataan. JSON on kevyt datamuoto, ja sitä on ihmisten helppo sekä lukea että kirjoittaa. (Json.org.) JSON-dataa voidaan JavaScriptissä käyttää suoraan JavaScript-oliona.

4 REST-RAJAPINNAN TOTEUTTAVAN SOVELLUKSEN RAKENTAMINEN

4.1 Eclipse-ohjelmointiympäristö

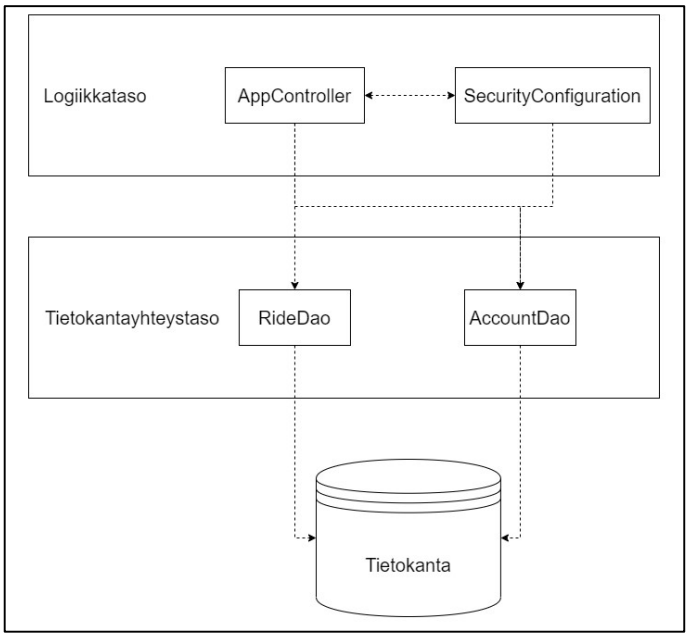
Eclipse-ohjelmointiympäristö tai Eclipse IDE on IBM:män vuonna 2001 aloitetusta Eclipse-projektista myöhemmin 2004 Eclipse Foundationiksi muodostuneen itsenäisen voittoa tavoittelemattoman säätiön kehittämä ohjelmointiympäristö (Eclipse Foundation b). Eclipse ohjelmointiympäristö on kehitetty Java-kehitystä silmällä pitäen, mistä se on kuuluisa. Ohjelmointiympäristöön on tarjolla muitakin ohjelmointikieliä kuten JavaScript, C++ ja PHP, joihin sopivat lisäosat löytyvät Eclipsen Eclipse Marketplace -verkkokaupasta. (Eclipse Foundation a.)

Koska Eclipse on suunniteltu juuri Java-ohjelmistokehitykseen, se sopii mainiosti Java-palvelimen kirjoittamiseen ja Spring-ohjelmointikehyksen soveltamiseen. Laajan ohjelmointikielitukensa ansiosta myös mahdollisen verkkoselainkoodin automaattinen tarkistus tapahtuu samassa ohjelmointiympäristössä, mikä helpottaa projektin resurssien hallintaa.

Tämän opinnäytetyön REST-rajapinta on rakennettu käyttäen Eclipse IDE -työkalua.

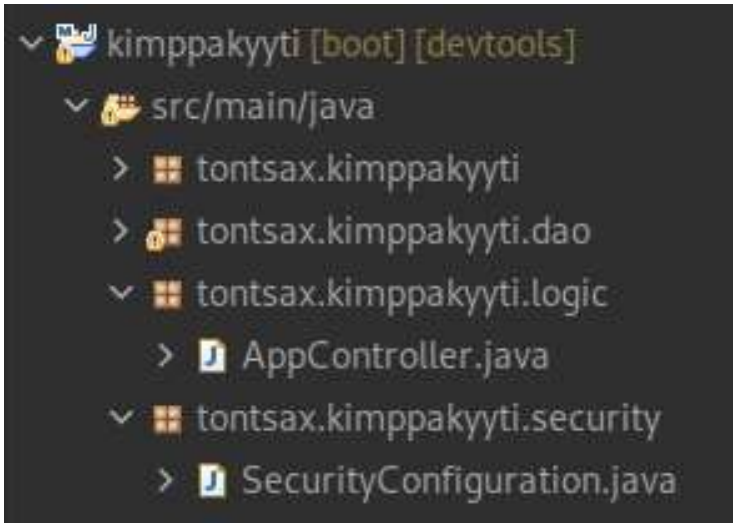
4.2 Sovelluksen rakenne

Tässä opinnäytetyössä REST-rajapinnan palvelinsovellus (Kuva 5) rakennetaan käyttäen kerrosarkkitehtuuria. Sovelluksen rakenne jaetaan erilaisiin kerroksiin, kuten logiikka-, käyttöliittymä- tai tietokantayhteyskerrokseen. Jokainen kerros toteuttaa tehtäviään omalla pelkistetyllä tasollaan ja jokainen kerros käyttää vain sen alapuolella olevan kerroksen palveluja. Kerrokset sisältävät tietokannan hallintaluokat ja sovellusten verkon yli käytävän kommunikoinnin. (Luukkainen 2020b.)



Kuva 5. Palvelinsovelluksen rakenne. (Luukkainen 2020b.)

Kerrosten käyttö ohjelmistokehityksessä on yleensä nähtävissä lähdekoodin luokkia sisältävien kansioiden kansiojaosta (Kuva 6). Esimerkiksi logiikkakerroksen luokat voivat olla saman kansion alla ja luokat voivat käyttää muiden kansioiden tai kerrosten sisällä olevia luokkia.

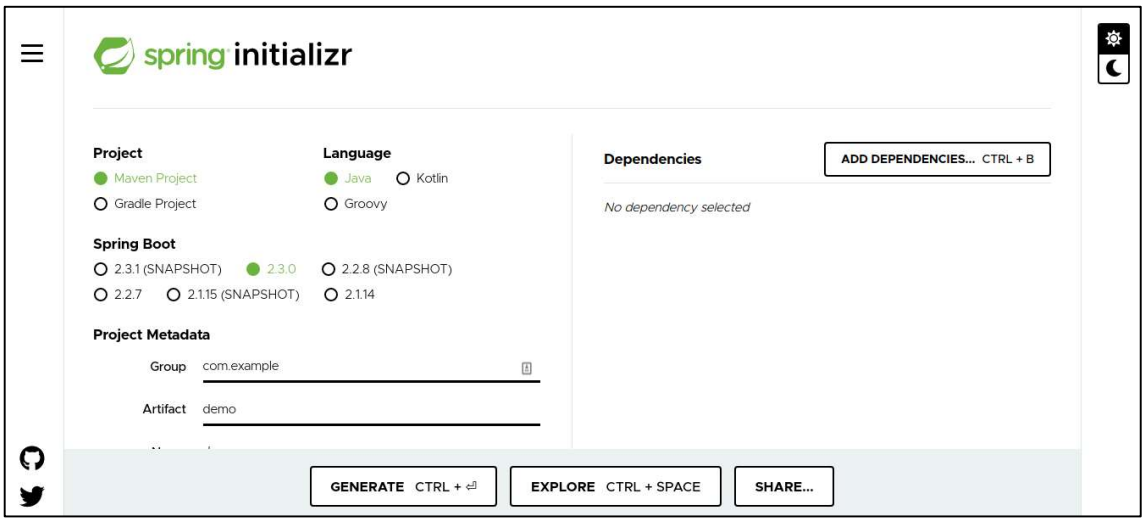


Kuva 6. Palvelinsovelluksen kansiojako.

4.3 Spring Boot -palvelinprojektin alustus

Tässä opinnäytetyössä tehty REST-rajapintapalvelu rakennettiin Spring ohjelmointikehyksellä tehdyn Spring Bootin avulla. Palvelimen tarkoituksena on tarjota REST-rajapintapalvelu esimerkiksi kimppakyytimobiilisovellukselle. Käyttäjän etsiessä kimppakyytiä, käyttäjä antaa sovellukselle hakukriteerit, minkä jälkeen sovellus välittää hakukriteerit palvelimelle, joka etsii kriteerit täyttävät kyydit tietokannasta, lajittelee ne ja lähettää takaisin sovellukselle. Tämän jälkeen sovellus esittää vastaanotetut kyydit, joista käyttäjä voi valita mieleisensä kyydin, jos sellainen löytyy, tai aloittaa kyydin hakuprosessin uudestaan uusin hakukriteerein.

Spring Boot -projekti aloitetaan Spring Initializr -työkalulla (Kuva 7), jonka avulla projektin alustuspakettiin valitaan kehitystyöväline, ohjelmointikieli, Spring Boot -versio, tarvittavat riippuvuudet ja täytetään projektia kuvailevat tiedot. Spring-projektin voi kirjoittaa täysin Java-, Java-alustalla toimivalla Apache Groovy- tai Javan virtuaaliympäristössä toimivalla Kotlin-ohjelmointikielellä. Tietojen valinnan jälkeen Spring Initializr kokoaa ja pakkaa sovelluskehittäjälle sopivan kehityspaketin. Paketti puretaan valittuun sijaintiin tietokoneella, tuodaan sovelluskehittäjän suosimaan ohjelmointiympäristöön ja kehitystyö voi alkaa. Tätä työtä varten valittiin kehitystyövälineeksi Maven, ohjelmointikieleksi Java ja Spring Boot -versioksi Spring Boot 2.3.0.



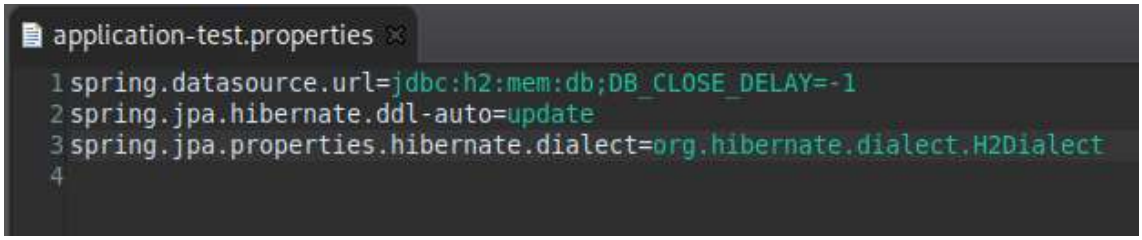
Kuva 7. Osa täytetystä Spring Ininiatizr -lomakkeesta.

Tämän projektin riippuvuuksiksi valitaan Spring Boot kehitystyökalu, Spring Webi, Lombok koodin niin sanotun boilerplaten eli useasti toistuvan samankaltaisen koodin kirjoittamisen vähentämiseksi, Spring konfiguraatioprosessori, Thymeleafi helpottamaan verkkosivun kirjoitusta, Spring Security, H2-tietokanta, Spring Data JPA ja PostgreSQL-tietokanta-ajuri Heroku-palvelinta varten, jos rajapinta halutaan myöhemmin julkaista testikäyttöön.

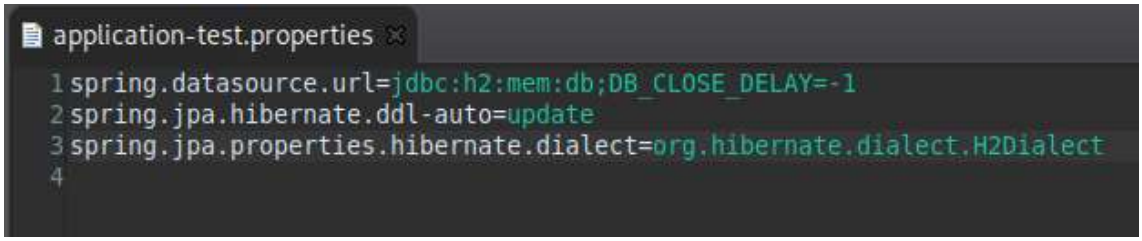
4.4 Automaatiotestien kirjoitus

Tietokantaa käyttäviä testejä kirjoittaessa ei ole suositeltavaa käyttää ohjelmiston ”live-tietokantaa” (Hellas (né Vihavainen) & Luukkainen 2019b), joka mahdollisesti saattaisi sisältää arvokkaita asiakastietoja ja näin vaarantaisi näiden tietojen menetyksen tai hallitsemattoman muutoksen. Testejä kirjoittaessa käytetään erikseen luotua mallitietokantaa, joka poistetaan testien jälkeen. Spring-ohjelmistokehyksen avulla mallitietokanta toteutetaan kirjoittamalla erilliselle profiilille konfiguraatiotiedosto, joka ladataan testejä suorittaessa (Hellas (né Vihavainen) & Luukkainen 2019b). Konfiguraatiotiedoston nimeämisrakenne on application-{profile}.properties, jossa aaltosulkeissa oleva profiilinimi kertoo Springille, millä profiililla kyseistä määrittelyä käytetään (Philip Webb et al. s.a). Käytettävä profiili merkitään @ActiveProfiles(”profiilin nimi”)-annotaatiolla testiluokan yläpuolelle (Hellas (né Vihavainen) & Luukkainen 2019b).

Tämän projektin testikonfiguraatiotiedostoksi (



Kuva 8) määriteltiin application-test.properties. Konfiguraatiotiedoston avulla Springille kerrottiin, että ohjelmaa testattaessa käytetään h2-tietokantajärjestelmää, joka luodaan testien käynnistyessä ja poistetaan testien loppuessa, tietokantaa voidaan tarvittaessa muokata automaattisesti, kerrotaan Hibernate ORM-kirjastolle käytettävä tietokantajärjestelmä ja asetus, jonka perusteella ohjelma kirjoittaa kaikki tietokantaan tehdyt kyselyt sovelluslokiin (Hellas (né Vihavainen) & Luukkainen 2019b). Määrittelytiedosto sijoitettiin projektissa kohteeseen src/main/resources (Tutorialspoint a).



Kuva 8. Palvelimen testikonfiguraatiotiedosto.

Määrittelytiedoston jälkeen projektiin luotiin testiluokka, johon tämän projektin tarvitsemat testit kirjoitettiin. Jotta testiluokka (Koodi 5) käyttäisi testeissään Spring-sovelluskehystä, se on merkittävä kahdella annotaatiolla *@RunWith(SpringRunner.class)* ja *@SpringBootTest*. Edellinen annotaatio kertoo testimoottorille, että testi käyttää Spring-sovelluskehystä ja jälkimmäinen annotaatio kertoo, että testiin ladataan Spring-sovelluskehyksellä kirjoitetut osat testattavasta sovelluksesta käyttöön. (Hellas (né Vihavainen) & Luukkainen 2019b.)

```
@ActiveProfiles(”test”)
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
class KimppakyytiApplicationTests {

   @Autowired
   private MockMvc mockMvc;

   @Test
   void contextLoads() {
   }

}
```

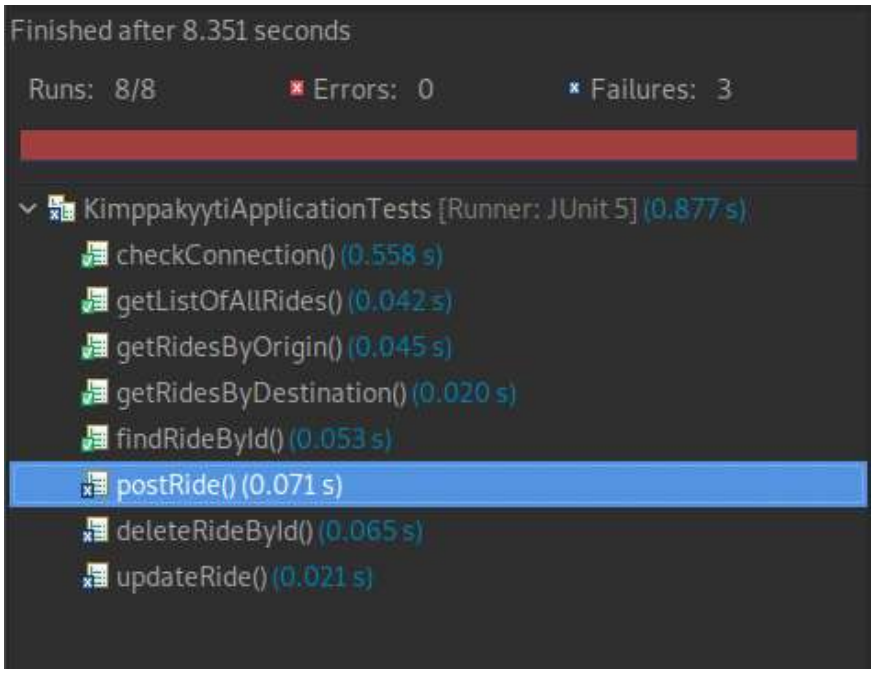
Koodi 5. Alustettu palvelintestiluokka.

Tämän projektin palvelimen testaukseen käytettiin integraatiotestauksen mahdollistavaa MockMvc-oliota, jonka avulla palvelimeen voi testattessa tehdä osoitepyyntöjä ja tarkastella osoitteesta saatuja vastauksia. MockMvc-olio saadaan käyttöön testiluokkaan lisättävällä *@AutoConfigureMockMvc*-annotaatiolla. (Hellas (né Vihavainen) & Luukkainen 2019b.)

Kaikki palvelinsovelluksen testit kirjoitettiin Java-virtuaalikoneelle tarkoitetulla JUnit-testikehyksen versiolla JUnit 5. JUnit on testikehys, joka on kehitetty niin sanotulla ”testaa ensin ja koodaa sitten” -mentaliteetilla. JUnitin ajatusmallissa koodi siis testataan aina ennen sen implementointia, jolloin kehitystyö kuvaisi askeleita ”testaa vähän, koodaa vähän jne”. (Tutorialspoint b.)

JUnit 5.7.0 JUnitin viimeisin versio JUnit 5.7.0 mahdollistaa aikaisempien JUnit 3 - ja JUnit 4 -versioiden testien ajon. JUnit 5.7.0 tarvitsee toimiakseen Javan kehityspakettiversion kahdeksan tai suuremman toimiakseen, mutta sillä voi testata Java 8 -version aikaisempaa koodia. (The JUnit Team.)

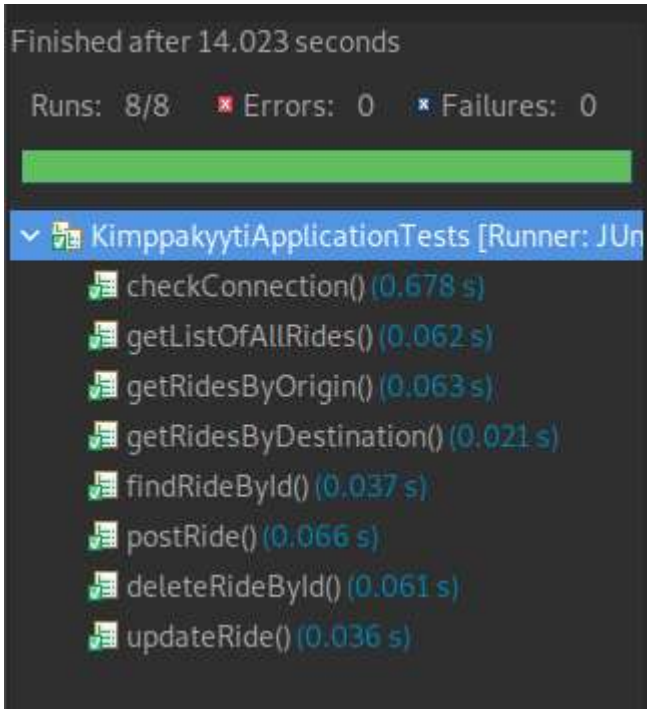
Testiluokkien valmistelun jälkeen testiluokka täytettiin projektille ajatelluilla oleellisilla testeillä. Testit kirjoitettiin niin, että ensin testiluokkaan luotiin uusi testi, minkä jälkeen itse ohjelmistoon kirjoitettiin testattavan toiminnon koodi. Tämän jälkeen toiminnolle tehty testi ajettiin (Kuva 9) ja tuloksia tarkasteltiin. Mahdollisten virheilmoitusten kohdalla, testattavan toiminnon koodia muokattiin, kunnes se läpäisi testin. Näin toimittaessa testien virheilmoitukset muodostuivat enemmänkin kehitystä ohjaaviksi ohjeiksi (Kuva 10) kuin virheellisistä toiminnoista ilmoittaviksi viesteiksi.



Kuva 9. Joukko onnistuneita ja epäonnistuneita testejä.

Kuva 10. Esimerkki testivirheilmoituksesta, joka kertoo että palvelimen vastaanottamasta viestistä puuttuu lähetettävän datan tyyppi.

Testien ajoa, tulosten tarkastelua ja ohjelman tai testien korjausta jatkettiin niin kauan, kunnes palvelinsovellus läpäisi kaikki sille kirjoitetut testit (Kuva 11). Jos palvelinsovellusta testattaisiin vain manuaalisesti esimerkiksi käyttöliittymän kautta, voi testaukseen mennä pitkiäkin aikoja ja se voi muuttua kehittäjän kannalta puuduttavaksi. Manuaalisen käyttöliittymän kautta tehtävä testaus voi myös mahdollisesti viedä kehittäjän keskittymistä palvelinsovelluksen kehittämisestä testauksen suorittamiseen ja näin hidastaa kehitysprosessia entistä enemmän. Manuaalinen testaus voi tuntua turhauttavalta varsinkin, jos sen joutuu suorittamaan aina pienen koodimuutoksen jälkeen.



Kuva 11. Joukko onnistuneita palvelintestejä.

Automatisoitujen testien avulla sovelluskehittäjä voi testata tekemiään muutoksia koodissa usein ja parhaimmillaan sekunneissa (Kuva 11). Automatisoidut testit myös poistavat sovelluskehittäjän tarpeen ajatella itse testausprosessia ja keskittyä pohtimaan kirjoittamaansa koodia sillä välin, kun automatisoidut testit pyörivät.

4.5 Postman

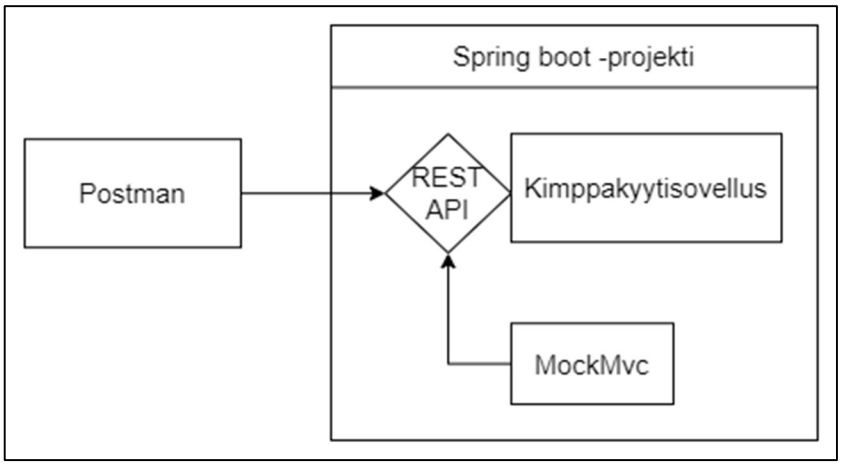
Postman on ohjelmointirajapintojen kehitykseen tarkoitettu yhteistyöalusta (Postman). Sen avulla ohjelmointirajapintojen kehitys on nopeampaa kuin normaalisti muun muassa sen takia, koska se poistaa sovelluskehittäjän tarpeen kirjoittaa kehittämälleen rajapinnalle laajoja testejä täysin alusta.

Postman tarjoaa käyttäjilleen interaktiivisia opastuksia alustansa käyttöön, laajan dokumentaation ja helposti käytettävän käyttöliittymän (Postman Learning Center). Postman-alustaa voidaan käyttää verkkoselaimella tai asentamalla työpöytäsovellus tietokoneelle.

4.6 Postman-testit viestintätoiminnolle

Jos tässä opinnäytetyössä tehtyä REST-rajapintaa käytettäisiin esimerkiksi kimppakyytimobiilisovelluksen teossa, niin kuljettajan ja asiakkaan välillä olisi mukavaa olla jokin keino vaihtaa viestejä toistensa kanssa. Joten rajapintapalveluun lisättiin viestintätoiminnallisuus ja sen testaus suoritettiin omien automaatiotestien kirjoituksen sijasta ohjelmointirajapintojen kehitykseen tarkoitetulla Postman-alustalla. Postman-alusta valittiin viestintätestien kirjoitukseen sen takia, että opinnäytetyötä tehdessä tulisi kokeiltua myös toista tapaa kirjoittaa automaatiotestejä.

Postman-alustalla tehdyt testit kannattaa suorittaa lokaalisti toimivan rajapintapalvelimen ja tietokannan kanssa, jotta julkistettuun tietokantaan ei syntyisi mitään peruuttamatonta vahinkoa. Koska Postman-alusta ei ole osa itse rakennettavaa rajapintaa (Kuva 12), niin tietokannassa olevan datan muokkaus tulee onnistua täysin itse rajapinnan kautta. Tämä tarkoittaa käytännössä esimerkiksi sitä, että kun Postman-alusta on ajanut testit, niin testatun tietokannan poisto tehtäisiin manuaalisesti tai siellä olevan testidatan poisto tulisi toteuttaa rakennettavan rajapintapalvelimen kautta.

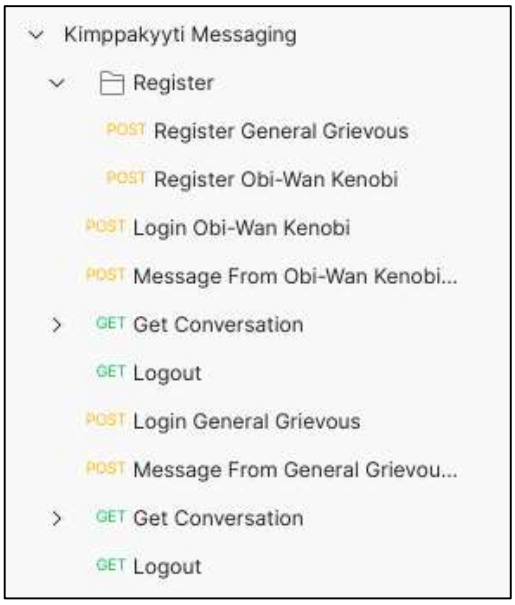


Kuva 12. Postman-alustan asema projektiin verrattuna.

Tässä opinnäytetyössä Postman-alustalla tehtiin testit vain rekisteröintiä, sisäänkirjautumista ja viestintätoiminnallisuutta varten (Kuva 13). Testaukseen käytettiin lokaalisti toimivaa versiota rakennettavasta rajapinnasta. Postman-testien jälkeen rakennettavan rajapinnan koko tietokanta poistettiin testien jälkeen manuaalisesti.

Postman-testien teko oli varsin yksinkertaista. Postman-alustalla ei tarvitse pohtia samankaltaisia yhteys- tai tietokannan täyttöratkaisuja kuin MockMvc:tä ja JUnitia käytettäessä. Postman-alustalla automatisoiduissa testeissä käytetyt HTTP-pyynnöt on helppo tehdä. Ei tarvitse kuin valita HTTP-metodi ja kirjoittaa osoite, jonne Postman-alusta tekee pyynnön. Pyyntöjen lähetyksen tekee myös helpoksi se, että pyyntöä rakentaessa Postman-alustalla on muutamia valmiita, miten pyynnön rungon muotoon. Pyynnön lähetyksen jälkeen Postman saadun vastauksen konsoliin pyynnön alapuolelle. Halutessaan sovelluskehittäjä pystyy hyödyntämään vastauksena saatua dataa Postman-ohjelmointirajapinnan avulla kirjoittaen JavaScriptillä esimerkiksi datan sisällön tarkistustestejä.

Opinnäytetyössä viestintätoimintoa testaavat Postman-pyynnöt kerättiin yhdeksi kokonaisuudeksi (Kuva 13). Tämä tehtiin sen takia, koska Postman pystyy näin suorittamaan useita pyyntöjä peräkkäisessä järjestyksessä. Viestintätoiminnallisuuden pyynnöistä GET-HTTP-metodilla toimivat pyynnöt varustettiin lisäksi vastaanotetun datan tarkistustestillä (Koodi 6) Postman-ohjelmointirajapintaa hyödyntäen. Tarkistustestin tarkoitus oli huolehtia, että palvelin varmasti palauttaa oikeanlaisen keskestelun ja että palvelimelle lähetetyt viestit tallennetaan oikeaan keskusteluun.

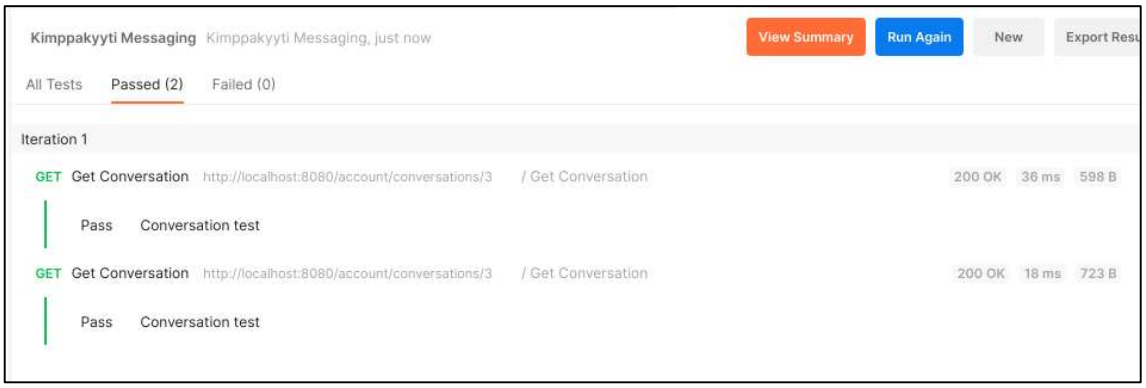


Kuva 13. Postman-testissä käytetty pyyntökokonaisuus.

```
pm.test(”Conversation test”, function () {
   pm.response.to.be.json;
   var jsonData = pm.response.json();

   var sender =pm.environment.get(“obi_wan”);
   var receiver = pm.environment.get(“grievous”);
   var message = pm.environment.get(“obi_message”);

   pm.expect(jsonData).to.have.nested.property(‘messages[0].sender’)
      .to.equal(sender);
   pm.expect(jsonData).to.have.nested.property(‘messages[0].receiver’)
      .to.equal(receiver);
   pm.expect(jsonData).to.have.nested.property(‘messages[0].message)
      .to.equal(message);

});
```

Koodi 6. Kimppakyyti keskustelun sisällön tarkistus palvelimen vastauksesta.

Jokaisen Postman-testauksen jälkeen rakennettavan REST-rajapinnan projektin juurikansiosta poistettiin sinne luotu tietokanta. Tietokannan poiston jälkeen rajapintasovellukseen tehtiin muutoksia, jos testit eivät vastanneet odotettuja tuloksia. Muutosten jälkeen Postman-testikokoelma ajettiin uudelleen ja menetelmää jatkettiin kunnes rajapintasovellus antoi odotetut tulokset testeistä (Kuva 14).



Kuva 14. Postman-alustan tapa esittää automatisoitujen testien tulokset.

Läpäistyjen Postman-testien jälkeen REST-rajapinta todettiin sisältävän riittävästi perustoimintoja, jotta sitä voitaisiin mahdollisesti käyttää erilaisten käyttöliittymäsovelluksissa tai jatkokehittää ilman suurempia vaikeuksia.

5 YHTEENVETO

Tässä opinnäytetyössä rakennettu sovellus on vain pintaraapaisu siitä, millainen REST-rajapinnan toteuttava palvelin voisi olla. Työ osoitti, että REST-rajapinnan rakentaminen on nykyaikaisten ohjelmistotyökalujen ansiosta varsin helppoa ja nopeaa. Työn avulla tutustuttiin sovelluskehityksessä käytettäviin erilaisiin työkaluihin ja kuinka ne voivat tehdä sovelluskehittäjän arjesta helpompaa.

Rajapintaa rakennettaessa myös automatisoitujen testien rooli osoittautui merkitseväksi tekijäksi niiden tuottaman ajallisen säästön ja sovelluskehityksen helpomman hallitsemisen myötä. Työ osoitti, että automatisoidut testit ja testiläheinen sovelluskehitys pitää huolen siitä, että sovelluksessa tapahtuvien muutosten jälkeen mahdolliset sovelluksen toimintavirheet huomataan ennen sovelluksen tuotantojakoon päästämistä.

Opinnäytetyössä rakennetun REST-rajapinnan jatkokehitystarpeita on useita. Niiden ymmärtämiseksi paremmin ne on tässä tapauksessa jaettu kahteen kategoriaan: käyttäjä- ja teknologiakohtaisiin jatkokehitystarpeisiin.

Käyttäjäkohtaisia jatkokehitystarpeita voisivat olla muun muassa käyttäjän syöttämän datan validointimenetelmä, maksuominaisuuden, karttapalvelun ja kuljettajien laadun kertovan tähtiarviointijärjestelmän lisäys.

Teknologisia jatkokehitystarpeita sen sijaan voisivat olla muun muassa palvelimen arkkitehtuurin muuttaminen kerrosrakenteesta mikropalvelinmalliin. Mikropalvelinmalli voidaan myös toteuttaa Spring-sovelluskehyksellä (VMware 2020e), joten sen kehitykseen ei vaadittaisi paljoa uuden aineiston tutkimista.

Rakennettua REST-rajapintaa voitaisiin myös jatkossa soveltaa erilaisten käyttöliittymäsovellusten kanssa esimerkiksi mobiilisovelluksen luonnissa tai asiakasrekisterin hallintasovelluksessa.

LÄHTEET

Eclipse Foundation a. Eclipse desktop & web IDEs. Viitattu 8.5.2020 https://www.eclipse.org/ide/

Eclipse Foundation b. About the Eclipse Foundation. Viitattu 8.5.2020 https://www.eclipse.org/org/

Google Developers. Meet Android Studio. Viitattu 15.9.2020 https://developer.android.com/studio/intro

Hellas (né Vihavainen) & Luukkainen 2019a. Web-palvelinohjelmointi Java Osa 1 1.3 Springsovelluskehys. Kurssimateriaali Helsingin yliopiston ja massiiviset avoimet verkkokurssit MOOC.fi verkkokurssiopetuksessa.

Hellas (né Vihavainen) & Luukkainen 2019b. Web-palvelinohjelmointi Java Osa 4 4.5 Sovellusten testaaminen. Kurssimateriaali Helsingin yliopiston ja massiiviset avoimet verkkokurssit MOOC.fi verkkokurssiopetuksessa.

Hellas (né Vihavainen) & Luukkainen 2019c. Web-palvelinohjelmointi Java Osa 6 6.4 Rajapinnat ja REST. Kurssimateriaali Helsingin yliopiston ja massiiviset avoimet verkkokurssit MOOC.fi verkkokurssiopetuksessa.

Json.org. Introducing JSON. Viitattu 9.6.2021 https://www.json.org/json-en.html

Matti Luukkainen 2020a. Ohjelmistotuotanto avoin yliopisto. Osa 3. Helsingin avoimen yliopiston ohjelmistotuotantokurssin verkkomateriaali kevät 2021. Viitattu 15.9.2020 https://ohjelmistotuotanto-hy-avoin.github.io/osa3/

Matti Luukkainen 2020b. Ohjelmistotuotanto avoin yliopisto. Osa 4. Helsingin avoimen yliopiston ohjelmistotuotantokurssin verkkomateriaali kevät 2021. Viitattu 15.9.2020 https://ohjelmistotuotanto-hy-avoin.github.io/osa4/

Mohamed 2019. Online recruitment application. Insinöörityö. Vaasan ammattikorkeakoulu Vaasa. 47 s.

Philip Webb, Dave Syer, Josh Long, Stéphane Nicoll, Rob Winch s.a. Spring Boot Reference Guide. Viitattu 3.6.2020 https://docs.spring.io/spring-boot/docs/1.0.1.RELEASE/reference/html/boot-features-external-config.html

Postman Learning Center. Introduction. Viitattu 11.5.2021 https://learning.postman.com/docs/getting-started/introduction/

Postman. API Platform: API Tools & Solutions. Viitattu 11.5.2021 https://www.postman.com/api-platform

REST API Tutorial. Richardson Maturity Model. Viitattu 12.5.2021 https://restfulapi.net/richardson-maturity-model/

Restcookbook.com. What is the Richardson Maturity Model. Viitattu 12.5.2021 https://restcookbook.com/Miscellaneous/richardsonmaturitymodel/

Roy T. Fielding & Richard N. Taylor 2000. Principled Design of the Modern Web Architecture.

The JUnit Team. JUnit 5 User Guide. Viitattu 13.10.2020 https://junit.org/junit5/docs/current/user-guide/

Tutorialspoint a. Spring Boot Application Properties. Viitattu 3.6.2020 https://www.tutorialspoint.com/spring_boot/spring_boot_application_properties.htm

Tutorialspoint b. JUnit - Overview. Viitattu 14.10.2020 https://www.tutorialspoint.com/junit/junit_overview.htm

VMware 2020a. Spring framework. Viitattu 19.5.2020 https://spring.io/projects/spring-framework

VMware 2020b. Spring boot 2.3.0. Viitattu 22.5.2020 https://spring.io/projects/spring-boot

VMware 2020c. Building a RESTful web service. Viitattu 22.5.2020 https://spring.io/guides/gs/rest-service/

VMware 2020d. Spring initializr. Viitattu 20.5.2020 https://start.spring.io/

VMware 2020e. Spring. Viitattu 17.8.2020 https://spring.io/

VMware 2020f. Spring Framework Overview. Viitattu 1.9.2020 https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/overview.html

VMware 2020g. RestController (Spring Framework 5.3.8 API). Viitattu 12.5.2021 https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html