Ville Haapamäki

Testikäyttäjien etsintäsovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotuotanto

Insinöörityö

20.2.2024

Tiivistelmä

Tekijä:	Ville Haapamäki
Otsikko:	Testikäyttäjien etsintäsovellus
Sivumäärä:	34 sivua
Aika:	20.2.2024

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Simo Silander Tiiminvetäjä Samppa Mattila

Tässä insinöörityössä käsitellään ratkaisua Nordean testiympäristössä esiintyvään testikäyttäjien ongelmaan. Tällä hetkellä kaikkien työntekijöiden tulee tietää testikäyttäjien pankkitunnukset erikseen, jotta he löytävät tarvittavat tiedot testaamiseen. Erilaisia käyttäjäyhdistelmiä on monia satoja, mikä tekee oikean testikäyttäjän löytämisestä lähes mahdotonta. Sovelluksen avulla pystytään suodattamaan käyttäjiä heidän tietojensa perusteella.

Sovellus on suunniteltu toimimaan mobiililla (Android/iOS) ja verkossa, joten sovelluksen avuksi päädyttiin luomaan palvelin, joka palauttaa testikäyttäjät käyttöliittymille. Tässä insinöörityössä keskitytään palvelinpuolen ja käyttöliittymän kehitykseen webissä. Palvelimen puolella tehtävä on löytää, tallentaa ja palauttaa käyttäjät käyttöliittymälle. Käyttöliittymässä käyttäjä pystyy suodattamaan saamiaan tietoja halutulla tavalla.

Ongelman ratkaiseminen auttaa kaikkia ohjelmiston parissa työskenteleviä. Myös esimerkiksi mainostiimit voivat helpommin ottaa markkinointikuvia halutuilla käyttäjillä. Testaajien ja kehittäjien ei enää tarvitse tallentaa tai muistaa käyttäjiä ulkoa, ja automatisoiduissa testeissä testikäyttäjät löytyvät nopeammin.

Avainsanat: Spring Boot, Java, Angular

Abstract

Author:	Ville Haapamäki
Title:	Test User Discovery Application
Number of Pages:	34 pages
Date:	20 February 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer
	Samppa Mattila, People Leader

The thesis deals with a solution to a test user problem in Nordea's test environment. Currently, all employees need to know the bank IDs of the test users individually to find the necessary information for testing. There are hundreds of different combinations of users, which makes it almost impossible to find the right test user. The application planned allows users to be filtered based on their personal data.

The application is designed to work on mobile (Android/iOS) and web, so it was decided to create a server that returns test users. This thesis focuses on the server side and user interface development on the web. The task of the server side is to find, store and return users to the user interface. On the user interface, the user can filter the data to get the information as desired.

Solving this problem would help everyone working on the software. It would also make it easier for advertising teams, for example, to take marketing pictures with the users they want. Testers and developers would no longer need to memorise or remember users, and automated tests would find test users more quickly.

Keywords: Spring Boot, Java, Angular

Sisällys

Käsitteet

CD/CI: *Continuous Integration and Continuous Deployment.* CI tarkoittaa jatkuvaa integraatiota. CD tarkoittaa jatkuvaa toimitusta tai julkaisua.

HTML: *HyperText Markup Language*. Merkintäkieli, jota käytetään verkkosivujen luomiseen. Se määrittää verkkosivun rakenteen ja sisällön, mutta ei ulkoasua tai toiminnallisuutta.

JSON: *JavaScript Object Notation*. Kevyt, helppolukuinen tiedonvaihtoformaatti.

Testikäyttäjä: Käyttäjä, joka on luotu testaamista varten. Nämä käyttäjät toimivat vain testiympäristöissä.

Testiympäristö: Ympäristö, jossa uudet toiminnot testataan ennen tuotantoon vientiä.

TTL: *Time To Live*. Aikamääre kuvaamaan sitä, kuinka kauan tietty tieto pidetään tallennettuna muistissa.

URL: *Uniform Resource Locator.* Verkko-osoite, joka kirjoitetaan verkkoselaimeen, kun vieraillaan verkkosivustolla. Se viittaa koko osoitteeseen, johon kuuluvat alkuosa verkkotunnuksen nimi ja kaikki muut polut, parametrit tai ankkurit

SDK: *Software development kit.* Kokoelma ohjelmistotyökaluja ja ohjelmia, joita kehittäjät käyttävät sovellusten luomiseen tietyille alustoille tai ohjelmointikielille.

1 Johdanto

Tässä insinöörityössä käsitellään testikäyttäjien etsintäsovellusta. Sovellus tehdään avuksi Nordean testiympäristöön, jonka avulla pystyttäisiin etsiä testikäyttäjiä. Nordea on Pohjoismaiden suurin pankki, joka toimii Ruotsissa, Suomessa, Norjassa ja Tanskassa. Näissä maissa Nordea toimii yksityis- ja yrityskäyttäjille [1].

Tällä hetkellä testaukseen tarvitaan valepankkitunnus, jolla kirjaudutaan sisään mobiili- ja nettipankkiin. Tarvittavien testikäyttäjien löytämiseksi yleensä kysytään apua testaajilta, jotka ovat tallentaneet muistiinpanoihinsa useita mahdollisia käyttäjiä. Tässä työssä toteutettavan sovelluksen avulla voidaan löytää erilaisia testikäyttäjiä suodattamalla niiden tietoja. Tämä parantaa merkittävästi kehitys- ja testaustyön tehokkuutta ja säästää aikaa etsinnässä.

Ongelmaan törmätään pankkikorttien kanssa. Jokaisella neljällä maalla on omat pankkikorttinsa ja niihin liittyvät toiminnot. Tämän lisäksi kullakin maalla on myös yrityspuolen pankki, jolla on taas omat toimintonsa ja korttinsa. Yhteensä eri osia on kahdeksan. Korteilla voi olla viisi eri tilaa, kuten esimerkiksi ”aktiivinen”, ”pois käytöstä” ja ”estetty”. Korttien tiloja ei voida muuttaa manuaalisesti. Jos jokin kortti estetään, sitä ei voi muuttaa enää aktiiviseksi. Jokaisessa maassa ja segmentissä on noin 15 erilaista pankkikorttityyppiä, kuten esimerkiksi ”luotto”, ”luottokortti opiskelijalle”, ”yhdistelmäkortti” ja ”debit”. Nämä korttityypit voivat olla myös eri järjestelmissä, joilla on omat toimintansa, kuten sallitut käyttömaat. Ymmärrettävästi oikean testikäyttäjän ja halutun kortin löytäminen on lähes mahdotonta ilman etukäteen annettua tietoa.

Työssä toteutetaan ensiksi sovellus Javalla Spring Bootin avulla. Sovelluksen tarkoituksena on etsiä potentiaalisia käyttäjiä ja samalla hakea näiden tiedot tallennetuista palveluista. Samalla luodaan sovellukseen rajapinnat, jotka palauttavat nämä tiedot käyttöliittymälle. Tämän jälkeen voidaan aloittaa sovelluksen käyttöliittymän kehittäminen, joka mahdollistaa tietojen suodattamisen ja nopean kirjautumisen eri testiympäristöihin.

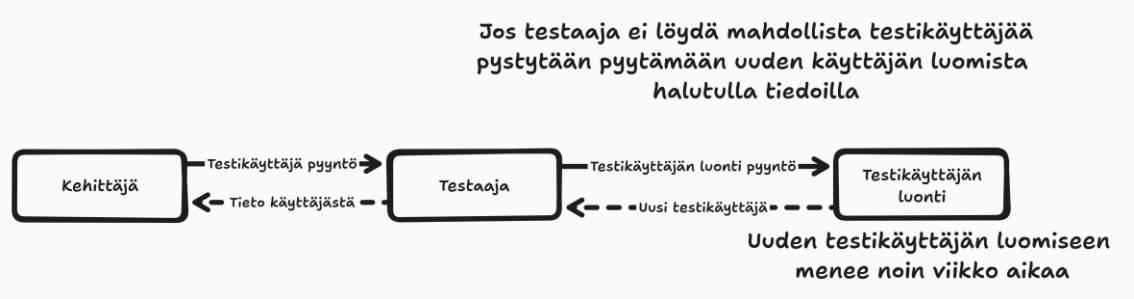
2 Taustat ja tavoitteet

2.1 Testaus

Nordeassa testaukselle on olemassa oma testiympäristönsä, jossa testataan uudet toiminnallisuudet ja vikojen korjaukset ennen tuotantoonvientiä. Yksityis- ja yrityskäyttäjät neljässä eri maassa muodostavat yhteensä kahdeksan eri testiympäristöä.

Kehitys- ja testausprosessia helpottaakseen on uusien toiminnallisuuksien kehityksessä yleisesti määritelty, millä käyttäjillä testaaminen on mahdollista. Toiminnallisuuksien kehityksessä mainitut testikäyttäjät ovat yleensä ainoita käyttäjiä, joita käytetään testausvaiheessa. Ongelmaksi muodostuu tämän myötä se, ettei toiminnallisuutta pysty testaamaan esimerkiksi käyttäjillä, joilla on vähemmän lupia.

Tuotannossa tapahtuvien vikojen korjauksessa yritetään toistaa tilannetta testiympäristössä, jotta voidaan olla varmoja korjauksesta. Oikean testikäyttäjän löytäminen onkin tämän takia todella vaikeaa, monesti joudutaan pyytämään uuden testikäyttäjän luomista.



Kuva 1. Testikäyttäjän etsintä tällä hetkellä kehittäjän näkökulmasta

Kuvassa 1 esitetään nykytilanne testikäyttäjän löytämisessä. Oikean testikäyttäjän hakemiseen kuluu todella paljon aikaa usealta eri ihmiseltä, mikä puolestaan hidastaa vikojen etsimistä ja korjaamista.

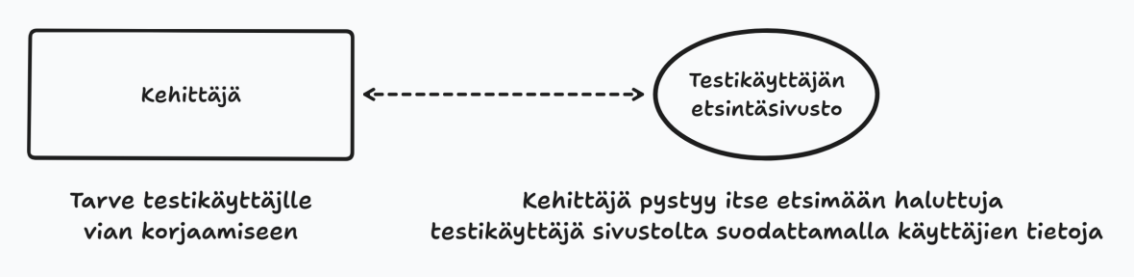
2.2 Ohjelman tausta

Jotta oikeaa kohtaa pystytään testaamaan halutulla testikäyttäjällä, tulee ensin selvittää testikäyttäjien tunnukset. Pientäkin muutosta saatetaan joutua testaamaan kaikissa eri ympäristöissä, joten kaiken kaikkiaan testattavaa tulee kahdeksaan eri ympäristöön. Kaikissa ympäristöissä on esimerkiksi erilaisia kortteja useita kymmeniä. Tämän lisäksi käyttäjiä voi olla hyvinkin erilaisia, joilla kaikilla on omat rajoitteensa (mm. alaikäinen, vanhempien käyttäjä, ulkomailla asuva käyttäjä, puuttuvia tietoja). Tämän vuoksi halutun testikäyttäjän etsiminen ilman ennakkotietoa on lähes mahdotonta. Jotta kehittäjät ja testaajat voivat varmistua toimintojen oikeanlaisesta toimivuudesta, kuluu testikäyttäjien etsimiseen todella paljon aikaa. Tämän lisäksi tuotannossa tapahtuvien ongelmien toistaminen testiympäristössä on hankalaa, sillä joissain tilanteissa ei pystytä löytämään tarpeeksi samantyyppisiä testikäyttäjiä ongelman toistamiseen. Myös käyttäjien tiedot muuttuvat ajoittain; kortit voivat olla jo vanhentuneet tai niiden tila muuttunut.

Testiympäristö tulee olla mahdollisimman samanlainen kuin tuotantoversio. Tämä muodostuu ongelmaksi esimerkiksi korteissa, joissa pankkikorttia suljettaessa sitä ei pysty enää avaamaan, vaan kortti on suljettu lopullisesti. Tässä tapauksessa täytyy taas hakea uusi testikäyttäjä, jolla olisi mahdollisesti samantapainen kortti.

2.3 Tavoitteet

Ohjelmiston tavoitteena on saada nopeutettua ja parannettua testauskattavuutta kaikissa sovelluksen ohjelmissa. Palvelinpuolen sovellus olisi vastuussa tietojen etsimisestä, käyttäjien tallentamisesta, käyttäjien tietojen päivittämisestä ja rajapinnan avulla tietojen palauttamisesta käyttöliittymälle.



Kuva 2. Kuvaus, kuinka testikäyttäjäsovellus auttaisi kehittäjiä

Käyttäjien eri tiedot tallennetaan samassa muodossa kuin miten ne palautuvat testiympäristöstä. Käyttäjien tietoja tulee myös mahdollisesti päivittää automaattisesti tai manuaalisesti, jotta sovellukseen tallennetut tiedot pysyvät ajan tasalla. Käyttöliittymässä pystyttään suodattamaan kaikkia kenttiä, joita käyttäjille on tallennettu.

2.4 Käytetyt teknologiat

2.4.1 Bitbucket

Versionhallinnaksi valitaan Bitbucket. Bitbucket on Atlassianin kehittämä verkkopalvelu, joka tarjoaa versionhallintaa Git-ohjelmistoille [2]. Muut palvelinpuolen ja web-sovellukset on myös tallennettu Bitbucketiin, joten uuden projektin luominen ei ole ongelma.

2.4.2 Splunk

Olemassa olevien testikäyttäjätunnuksia ei voida tietää ennakolta. Yksi tapa, jolla pystytään löytää useampi testikäyttäjätunnus, mutta ei kaikkia, on Splunk Enterprisen avulla. Splunk Enterprise on Splunk-ohjelmistoyrityksen luoma sovellus, joka mahdollistaa palvelimien lokien hakemisen, analysoinnin ja visualisoinnin [3]. Tämän avulla pystytään suodattamaan kaikki käyttäjätunnukset, joilla on jokin lokimerkintä. Splunkin hyödyntäminen ei kuitenkaan ole täydellinen ratkaisu, koska jotta testikäyttäjä saadaan Splunkiin, tulee niille olla jokin

lokiviesti kirjattuna. Kuitenkin Splunkin hyödyntäminen auttaisi paljon ongelman ratkaisussa.

2.4.3 Jenkins

Ohjelmisto tulee testata ja liikuttaa palvelimelle, josta käyttöliittymä pystyy käyttämään rajapintoja. Tähän tarvitaan siis jatkuvaa integraatiota ja julkaisuputkea (CD/CI). Koska myös muut sovellukset käyttävät jo Jenkinsiä tätä varten, voidaan Jenkins ottaa käyttöön myös tämän ongelman ratkaisuksi. Jenkins on avoimen lähdekoodin automaatiopalvelin, jota hyödynnetään jatkuvan integraation ja jatkuvan toimituksen palvelinympäristöissä [4].

Jenkinsin avulla saadaan siirrettyä ohjelmisto eri ympäristöihin. Jenkinsin tulee testata jokainen vetopyyntö (pull request), koota ohjelman päärepositoriossa ja siirtää ohjelma testiympäristöön käyttöliittymien käytettäväksi.

2.4.4 Spring Boot

Palvelinpuolen kehitykseen valitaan Spring Boot avuksi. Spring Boot on Spring Frameworkin moduuli, joka on suunniteltu yksinkertaistamaan Spring-sovellusten asennusta ja kehittämistä [5]. Ohjelmisto helpottaa sovelluksen konfigurointia ja nopeuttaa palvelimen kehitystyötä.

2.4.5 Hazelcast

Haettujen testikäyttäjien tiedot täytyy tallentaa jonnekin. Yhtenä mahdollisuutena tietojen tallentamiseen ovat tietokannat, jonne voitaisiin tallentaa kaikki tiedot helposti. Tietokantojen käyttäminen on kuitenkin kallista. Toinen mahdollisuus olisi Hazelcast, jota pystytään käyttämään NoSql-tietokantana. Hazelcast on avoimen lähdekoodin muistipohjainen alusta Javalle [6]. Hazelcast-Instanssi on jo luotu muita sovelluksia varten testiympäristöön, joten sitä pystytään käyttämään tietojen tallennukseen.

2.4.6 Postman

Palvelua testataan kehitysvaiheessa muutosten yhteydessä. Tähän tarkoitukseen voidaan käyttää Postman-ohjelmaa. Postman on alun perin Abhinav Asthana kehittämä ohjelmistokehittäjille suunniteltu työkalu, joka helpottaa API-kehitystä ja testausta [7]. Ohjelmiston avulla pystytään lähettämään palveluihin halutut tiedot sekä testaamaan ohjelman rajapintojen toimivuus.

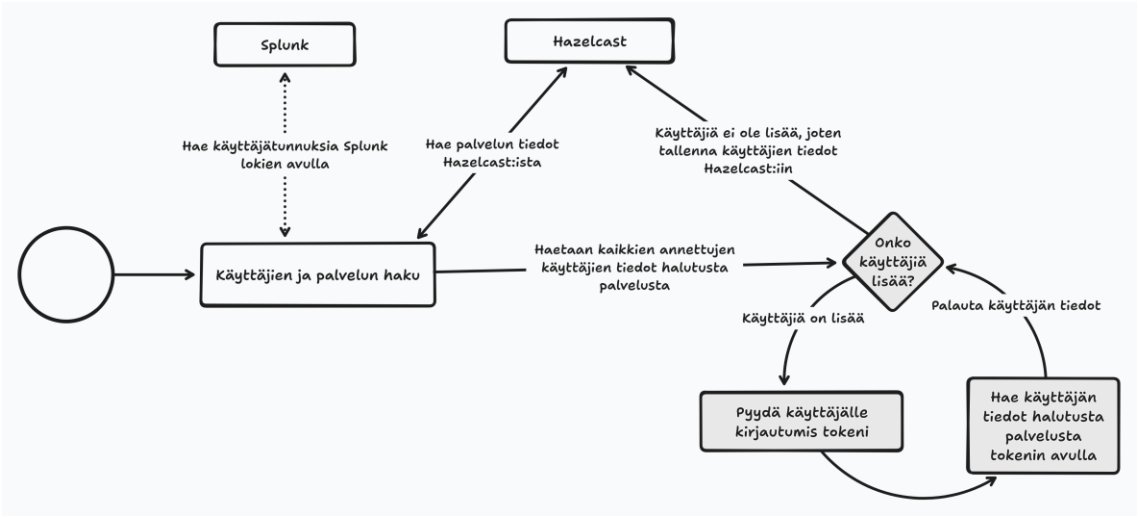
2.4.7 Angular

Käyttöliittymän tulee olla helposti kehitettävissä ja mahdollisimman helppo ylläpitää. Yrityksen muut nettisovellukset on kehitetty Angular-kirjaston avulla, joten yhtenäisyyden takia valitaan myös tämän sovelluksen käyttämään Angularia. Angular on Googlen kehittämä TypeScript-pohjainen avoimen lähdekoodin ohjelmistokehys [8].

3 Palvelupuolen kehitys

3.1 Yleinen toimivuus etsintäskriptissä

Testikäyttäjien hakuskriptin ideana on hakea testikäyttäjät ja niiden tiedot halutusta palvelusta (kuten kortit, tilit, lainat). Tiedot tallennetaan Hazelcast-instanssiin, josta pystytään hakemaan tietoja myöhemmin.



Kuva 3. Testikäyttäjän tietojen hakuskriptin toiminta kuvana.

Kuvassa 3 on esitetty, kuinka käyttäjien tietohaku toimii käytännössä. Alussa, jos testikäyttäjälistaa ei ole annettu, haetaan tunnukset Splunk-lokien avulla. Testikäyttäjätunnuslistan pystyy antamaan myös pyynnössä, jolloin Splunk-hakua ei tarvitse tehdä. Hazelcastiin on tallennettu jo tarvittava tieto palvelusta, jota käytetään testikäyttäjän tietojen hankkimiseen.

3.2 Testikäyttäjien etsintä Splunkin avulla

Ennen palveluiden tietojen hakua tarvitaan testikäyttäjiä, joilta tietoja etsitään. Kahdeksaan eri ympäristöön testikäyttäjän etsiminen manuaalisesti vie paljon aikaa, joten lisätään testikäyttäjien etsintämahdollisuus palveluun. Testikäyttäjien etsintään voidaan käyttää Splunk-lokeja avuksi. Vaatimuksena on, että testikäyttäjälle on joskus kirjauduttu sisään ja lokiviesti on tallennettu Splunkiin. Manuaaliseen testikäyttäjien etsintään käytetään Splunkin selainversiota. Lokiviestejä on monista eri ympäristöistä miljardeittain, jonka takia etsinnässä menee todella kauan. Splunkin avulla pystyy lokiviestejä kuitenkin suodattamaan monilla eri tavoilla. Monien kokeilujen jälkeen päädyttiin seuraavaan suodatustekstiin:

index="*_test*" source="*cards*" | rex field=_raw "user=(?<user>\d+)"| regex user="^\d{8}\b" | stats count by user | fields - count | head 500

Suodatustekstin osien selvennökset:

- ”Source” kertoo, mistä palvelusta lokiviesti tulisi olla kirjattu. Koska monilla testikäyttäjillä ei ole kortteja, keskityttiin vain testikäyttäjiin, joilla on jokin lokiviesti korttipalvelusta.

- ”rex field=_raw "user=(?<user>\d+)"” kertoo, mitä tietoa halutaan palautuvan. ”_raw” kentästä tulisi löytyä ”user=” kenttä, josta palautetaan sen arvo. Näin saadaan pelkästään käyttäjätunnukset tietoon.

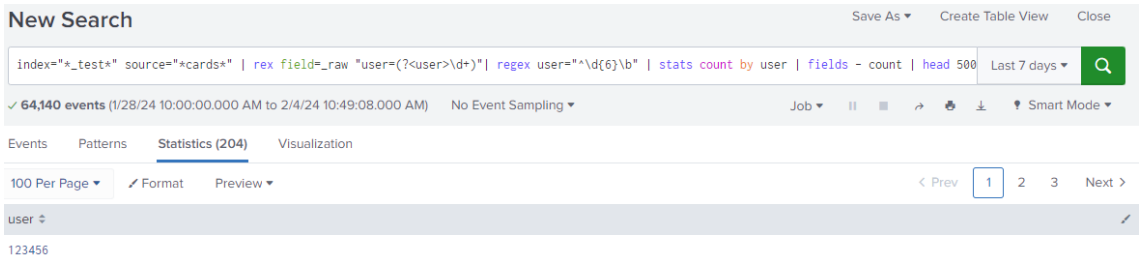
- regex user="^\d{8}\b" kertoo käyttäjätunnuksien pituuden. Esimerkiksi nykyään Suomessa kaikki testikäyttäjät ovat kahdeksanmerkkisiä, joten tiedetään, että vain kuusinumeroiset tunnukset tulee toimimaan.

- ”stats count by user” lajittelee testikäyttäjät järjestykseen eniten lokiviestiä kirjanneesta tunnuksesta vähiten kirjattuun. Tämän avulla saadaan tietoon, mitkä testikäyttäjät saattaisivat olla suosituimpia.

- ”fields – count” poistaa ”count”-kentän, jotta pelkät käyttäjätunnukset palautuisivat.

- ”head 500” valitaan 500 suosituinta testikäyttäjää. Tämä täytyy tehdä, koska monilla mailla saattaa löytyä yli 2 000 testikäyttäjää eikä kaikilta näiltä haluta etsiä tietoja.

Lisätään suodatusteksti Splunkin sivulle ja varmistetaan sen toimivuus:



Kuva 4. Suodatustekstin toimivuuden varmistus Splunkin sivuilla

Kuvassa 4 on nähtävissä varmistus suodatustekstin toimivuudesta. Etsinnän aikaväli on viimeiset seitsemän päivää, joista löytyi yhteensä 64 140 tapahtumaa. Kuvassa 4 vasemmassa alakulmassa nähdään vain käyttäjätunnukset palautettuna, joten suodatuksen toimivuudesta voidaan olla varma.

3.3 Käyttäjätunnusten etsiminen koodissa

Varmistuksen jälkeen tulee käyttäjien haku lisätä koodiin. Splunkilla on oma SDK käytettävissä yrityskäyttäjille ja laaja dokumentaatio siitä, kuinka SDK:tä tulisi käyttää [9]. Lisätään Splunk-riippuvuus projektin pom.xml-tiedostoon esimerkkikoodin 1 mukaisesti.

```
<dependency>
    <groupId>com.splunk</groupId>
    <artifactId>splunk</artifactId>
    <version>1.6.0.0</version>
</dependency>
```

Esimerkkikoodi 1. Splunk-riippuvuus pom.xml-tiedostossa

Kirjautumiseen on kolme eri tapaa: bearer-tokenin käyttäminen, käyttäjän ja salasanan käyttö tai istuntoavaimen käyttö. Näistä vaihtoehdoista valitaan käyttäjän ja salasanan käyttö kirjautumisen yksinkertaisuuden vuoksi. Dokumentaatiossa kerrotaan, miten kirjautumisen tulisi tapahtua käyttäjätunnuksen ja salasanan kanssa.

```
import com.splunk.*;

public class SplunkTest {

    public static void main(String[] args) {
        // Luo ServiceArgs-olio ja aseta siihen kirjautumistiedot
        ServiceArgs loginArgs = new ServiceArgs();
        loginArgs.setUsername("admin");
        loginArgs.setPassword("yourpassword");
        loginArgs.setHost("localhost");
        loginArgs.setPort(8089);

        // Luo yhteys Splunkiin
        Service service = Service.connect(loginArgs);
    }
}
```

Esimerkkikoodi 2. Splunk-yhteyden muodostaminen SDK:n avulla

Esimerkkikoodissa 2 tarvitsee kertoa käyttäjänimi, salasana, host ja portin numero. Kun nämä neljä asiaa on vaihdettu oikeiksi, saadaan Splunk yhdistettyä yrityksen Splunk-palveluun.

```
public List<String> getUsersFromSplunk(ValidCountry country, ValidSeg-
ment segment) {
    // Hae suodatus teksti oikealle maalle ja segmentille
    String searchQuery_normal = splunkUtil.getSearchQuery(country,
segment);
    log.info("Started searching users with string: " + search-
Query_normal);
    JobArgs jobargs = new JobArgs();
    // Aseta haku normaaliksi
    jobargs.setExecutionMode(JobArgs.ExecutionMode.NORMAL);
    Job job = initService().getJobs().create(searchQuery_normal,
jobargs);
```

```
    // Odota kunnes haku on valmis
    while (!job.isDone()) {
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Hae tulokset ja käytä sisäänrakennettua XML-parseria tulosten
näyttämiseen
    Map<String, Object> arguments = new HashMap<>();

    //Tallenna tulokset
    InputStream resultsNormalSearch = job.getResults(arguments);

    ResultsReaderXml resultsReaderNormalSearch;

    try {
        resultsReaderNormalSearch = new ResultsReaderXml(resultsNor-
malSearch);
        HashMap<String, String> event;
        List<String> foundUsers = new ArrayList<>();
        while ((event = resultsReaderNormalSearch.getNextEvent()) !=
null) {
            for (String key : event.keySet())
                foundUsers.add(event.get(key));
        }
        return foundUsers;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

Esimerkkikoodi 3. Käyttäjien etsintäkoodi

Esimerkkikoodi 3 on suurimmaksi osaksi samanlaista kuin Splunkin dokumentaatiossa [10], mutta yksinkertaisuudessaan koodissa haetaan aluksi

suodatusteksti ”getSearchQuery”-funktiota käyttäen, joka palauttaa suodatustekstin. ”getSearchQuery”-metodi tarvitsee tiedon maasta ja segmentistä oikeiden käyttäjätunnuksien valitsemiseen. Kun tulokset saadaan, palautetaan vain käyttäjätunnukset kuvassa 4 nähdyllä tavalla.

3.4 Tietojen haku palvelusta

Tiedot palautuvat palvelusta yleensä listana. Palveluiden tiedossa on URL, josta voidaan hakea testikäyttäjän tiedot. Testikäyttäjälle haetaan ensiksi token authentikaatio -palvelusta token, jota voidaan käyttää palveluiden tietojen pyyntiin. Testikäyttäjiä on paljon ja tietojen haussa eri palveluista kuluu paljon aikaa. Jotta tallennusprosessia nopeutettaisiin, käytetään Javan tarjoamaa ”Callable”-toimintoa, jolla pystytään suorittamaan hakuja rinnakkain. Java ”Callable”:n avulla voidaan suorittaa toiminnot toisella säkeellä (thread) ja antaa ohjelman edetä pääsäkeellä. ”Callable” palauttaa ”Future”-tuloksen, jonka arvo saadaan “get”-metodilla. ”get”-metodi pysäyttää pääsäkeen suoritukset, kunnes futuren tehtävä on valmis.

```
public class UserCreator implements Callable<List<User>>
```

Esimerkkikoodi 4. Luokka, joka toteuttaa ”Callable”-luokan.

Esimerkkikoodissa 4 kerrotaan luokka, joka palauttaa listan käyttäjistä. Luokka tarvitsee ”call”-metodin, jonka tulos tullaan palauttamaan Future-objektina.

```
public void updateUsers(String serviceName, List<String> userIds,
ValidCountry country, ValidSegment segment) {
    List<User> usersCollection = new ArrayList<>();
    List<Future<List<User>>> futures = new ArrayList<>();

    ServiceData serviceData = getService(serviceName);

    // Luo UserCreator-olio ja lisää se futures-listaan
    UserCreator userCreator = new UserCreator(userIds, serviceData,
dependencyService, depedencyObjects);
    futures.add(executorService.submit(userCreator));
```

```
    // Käy läpi kaikki futures-listan alkiot ja lisää niiden palautta-
mat käyttäjät usersCollection-listaan
    for (Future<List<User>> future : futures) {
        usersCollection.addAll(future.get());
    }
    UserDataList usersList = null;
    // Haetaan käyttäjät hazelcastista
    usersList = dataCacheService.getTestUsersServiceEntry(service-
Data.getServiceName(), country, segment).toFuture().get();
    // Päivitettään viimeisin muokkaus aika hazelcastiin
    dataCacheService.putServiceInfoEntry(serviceData.getServiceName(),
country, segment, TestUserInfoModal.builder().lastUpdated(Local-
DateTime.now()).build()).toFuture().get();

    if (usersList == null)
        usersList = new UserDataList();
    // Käy läpi kaikki käyttäjät ja lisää ne usersList olioon
    for (User user : usersCollection) {
        if (usersList.getUsers() == null || usersList.getUsers().is-
Empty()) {
            List<User> users = new ArrayList<>();
            users.add(user);
            usersList.setUsers(users);
            continue;
        }

        //Add user to first to the list
        usersList.getUsers().add(0, user);
    }

    // Tallennetaan käyttäjät hazelcastiin
    dataCacheService.putTestUsersServiceEntry(serviceData.getService-
Name(), country, segment, usersList).toFuture().get();
}
```

Esimerkkikoodi 5. Käyttäjien tietojen etsinnän pääluokka jo valmiiksi annetulla listalla.

Lyhyesti käyttäjien päivittämisessä luodaan alussa lista, jonne tallennetaan ”UserCreator”-oliot. Olio ottaa vastaan listan käyttäjätunnuksista, joiden tiedot tullaan etsimään annetusta palvelusta. Listaan lisäämisen jälkeen odotetaan, että kaikki listassa olevat Futuret ovat valmiita. Nämä tallennetaan listaan, joka lopussa tallennetaan Hazelcastiin.

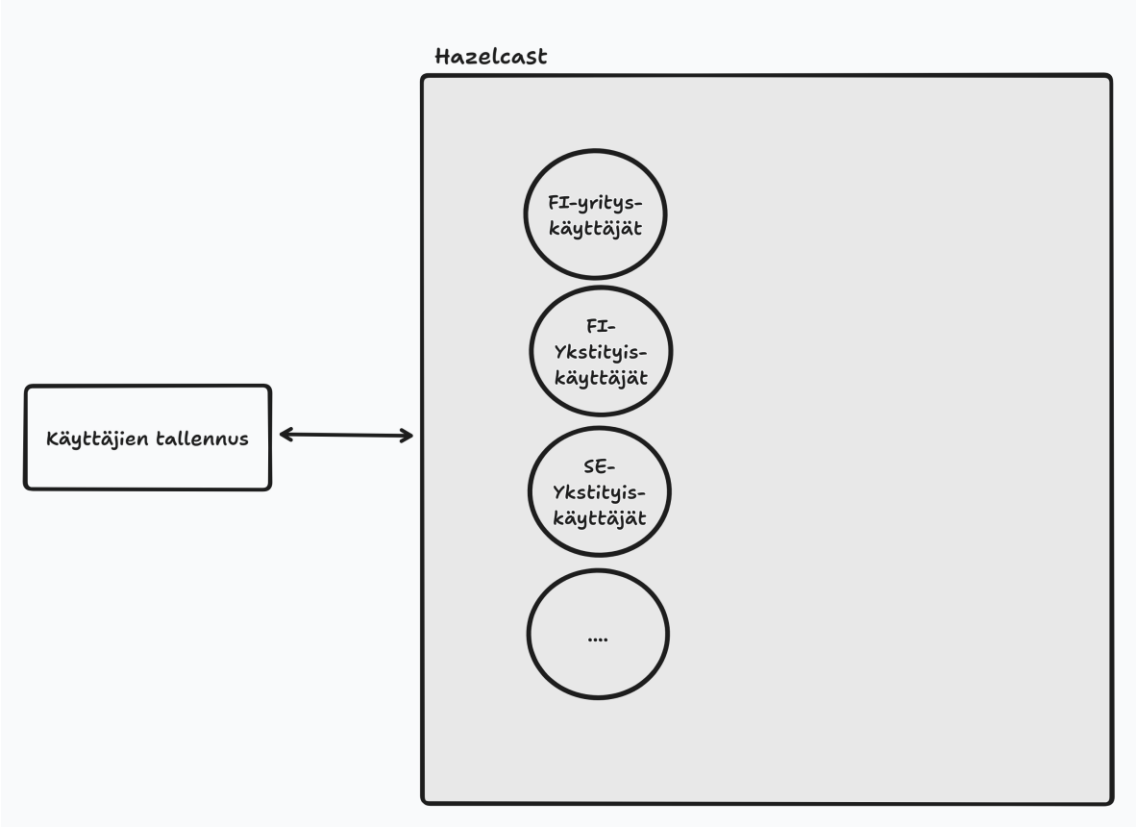
```
public void createUsers(String serviceName, Country country, Segment segment)
{
    //Etsi käyttäjät Splunkista
    List<String> users = splunkService.searchSplunk(country, segment);
    //Jaa käyttäjät listoihin, joissa on maksimissaan 20 käyttäjää
    List<List<String>> userSets = Lists.partition(users, 20);
    //Hae palvelun tiedot
    ServiceData serviceData = getService(serviceName);
    List<User> usersCollection = new ArrayList<>();
    List<Future<List<User>>> futures = new ArrayList<>();
    //Luo käyttäjät ja lisää ne future listaan
    for (List<String> userSet : userSets) {
        UserCreator userCreator = new UserCreator(userSet, serviceData, null,
null);
        futures.add(executorService.submit(userCreator));
    }

    //Odota, että kaikki käyttäjä tiedot on heattu
    for (Future<List<User>> future : futures) {
        usersCollection.addAll(future.get());
        dataCacheService.putServiceInfoEntry(serviceName, country, segment,
TestUserInfoModal.builder().lastUpdated(LocalDateTime.now()).build()).toFu-
ture().get();
    }

    //Lisää käyttäjät Hazelcastiin
    dataCacheService.putTestUsersServiceEntry(serviceName, country, segment,
new UserDataList(usersCollection)).block();
}
```

Esimerkkikoodi 6. Käyttäjien luonti ilman ennalta olemassa olevaa käyttäjälistaa

Esimerkkikoodissa 6 on funktio käyttäjien luomisesta. Funktiossa ei oteta vastaan listaa testikäyttäjätunnuksista, kuten esimerkkikoodissa 5. Testikäyttäjälista haetaan Splunkista alussa. Tämän jälkeen testikäyttäjät jaetaan uusiin listoihin, joissa on maksimissaan 20 tunnusta. Lista annetaan ”UserCreator”-luokalle, joka hakee kaikki käyttäjien tiedot halutusta palvelusta. Luokkien ollessa valmis pyydetään luokista palautuvat vastaukset silmukassa takaisin käyttämällä ”future.get()”-metodia. Lopussa tallennetaan kaikki saadut tiedot Hazelcastiin.

3.5 Tietojen tallennus Hazelcastiin

Käyttäjien tietoja tulee hakea monista eri palveluista ja tallentaa ne Hazelcastiin. Yksi yksinkertainen tapa olisi tallentaa kaikki käyttäjien tiedot yhteen tauluun, josta niitä pystyisi hakemaan.

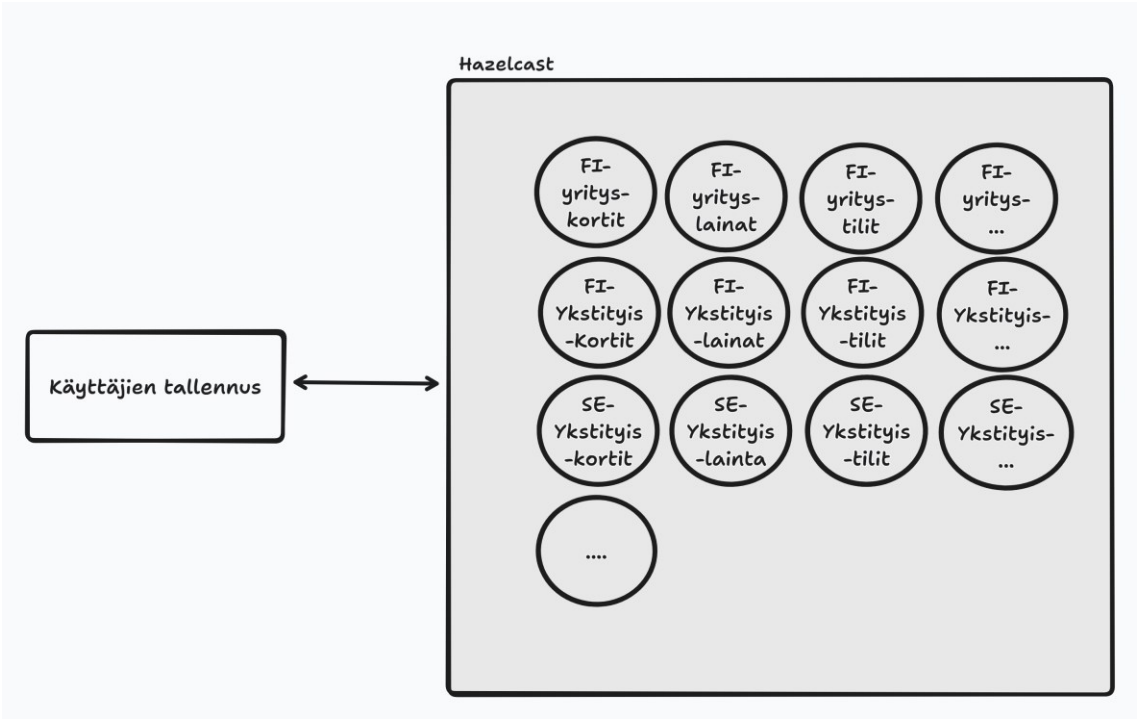


Kuva 5. Etsintäsovelluksen yksinkertainen tallennus Hazelcastiin

Kuvassa 5 on yksinkertaistettu kuva tallennusprosessista. Yksinkertaisuudessaan testikäyttäjät tallennettaisiin maa- ja segmenttitauluihin, joiden tiedoissa olisi kaikkien palveluiden testikäyttäjätiedot. Instanssien muistien koot tekevät tämän ongelmalliseksi. Ohjelma kaatuu, jos käyttäjillä on paljon tietoa ja nämä tiedot yritetään tallentaa välimuistiin.

Kuva 6. Java heap size error Splunk-lokeista, kun tietoa on tallennettu liikaa välimuistiin

Tämä myös rajaa ohjelman skaalautumista vain muutamaan palveluun. Yksi tapa olisi hajottaa taulut pienempiin osiin, jotka palautettaisiin vain haluttaessa.



Kuva 7. Hajotettu malli Hazelcast-tallennuksesta

Kuvassa 7 on taulut hajotettuna pienempiin osiin. Taulujen nimistä saadaan selville maa, segmentti ja palvelu. Nyt pystytään tallentamaan kaikkien haluttujen

palveluiden tiedot omaan tauluunsa, jolloin välimuistiin tallentaminen ei muodostu enää ongelmaksi.

Hazelcastin konfiguraatiota tarvitsee muuttaa, jotta muistia ei tyhjennettäisi. Hazelcastillä on ”Time to live” -aika, joka kertoo milloin, tieto tulisi poistaa muistista [11].

```
<hazelcast>
    <map name="default">
        <time-to-live-seconds>0</time-to-live-seconds>
    </map>

</hazelcast>
```

Esimerkkikoodi 7. Hazelcast-konfiguraatio, jossa ”Time To Live” on muutettu arvoon 0

Esimerkkikoodissa 7 on kuvattu tapa, kuinka ”Time to live” -aikaa pystyy muuttamaan Hazelcastin konfiguraatiossa. Konfiguraatiomuutos on tiedostossa ”hazelcast.xml”, josta Hazelcast osaa etsiä konfiguraatiomuutokset. Kentän arvoksi laitetaan 0, jotta varmistutaan, ettei tietoja poisteta ikinä muistista.

3.6 Palvelimen rajapinnat

Rajapinnat, jotka palvelimella on oltava, on esitetty seuraavassa taulukossa.

Taulukko 1. Palvelimen rajapinnat

Selvennys	**Tapahtuma**
Hae palveluun tallennetut testikäyttäjät	GET /users
Tieto milloin palvelu on viimeksi päivitetty	GET /info
Lisää/Päivitä palvelun testikäyttäjiä	POST /users
Hae palvelut	GET /service
Lisää uusi palvelu	POST /service

3.6.1 Palvelun lisääminen ja muuttaminen

Uuden palvelun lisääminen tapahtuu ”POST/service”-rajapintaa käyttämällä. Pyynnön rungossa kerrotaan tarvittavat tiedot siitä, kuinka käyttäjille pystyy saamaan tiedot.

```
{
    "service_name": "cards",
    "service_url": "https://example.com/",
    "response_key": "results"
}
```

Esimerkkikoodi 8. Esimerkki uudesta palvelupyynnöstä

Yksinkertaisessa palvelun haussa tarvitsee antaa palvelulle jokin nimi sekä URL, josta tiedot haetaan. ”response_key” on valinnainen kenttä, jonka avulla pystytään valitsemaan vain haluttu lista, kuten esimerkiksi kaikki pankkikortit.

```
{
    "service_name": "card-details",
    "service_data_dependency": {
        "service_name": "cards",
        "field_name": "card_id"
    },
    "service_url": "https://www.example.com/{}",
    "response_key": null
}
```

Esimerkkikoodi 9. Esimerkki uudesta palvelusta, jolla on yhteys toiseen palveluun.

Jos palvelulla on tarvetta käyttää jonkin muun palvelun tietokenttää, on se mahdollista. ”service_data_dependency” -objektille tulee antaa toisen palvelun nimi ja tietokenttä, jota tulisi käyttää pyynnössä. Näissä tilanteissa URL-tiedossa on merkittynä ”{}”, joka vaihdetaan annettuun kenttään. Esimerkiksi koodiesimerkissä 9 URL ”https://www.example.com/{}", muuttuu ”https://www.example.com/[card_id]" pyyntöjä luodessa.

Jo ennestään tallennettuja palveluita voi myös muokata. Jos pyynnössä mainitun palvelun nimi on jo olemassa, sen tiedot korvataan uusilla. Tämän avulla pystytään helposti korjaamaan vääriä tietoja palveluista.

3.6.2 Käyttäjien tallennus

Käyttäjien manuaalinen tallennus tapahtuu ”POST/users”-rajapintaa käyttämällä. Kuten käyttäjien haussa, tallennus vaatii kaksi argumenttia: maan ja segmentin.

```
{
    "service": "cards",
```

```
    "userIds": ["1234"]
}
```

Esimerkkikoodi 10. Esimerkki käyttäjien tietojen tallennuspyynnön rungosta

Pyynnön rungossa kerrotaan tieto, mitä palvelua halutaan päivittää, sekä käyttäjätunnukset listana. Palvelin palauttaa vastauksen, jos palvelu on löydetty onnistuneesti ja alkaa hakea tietoa palvelusta. Tallennus tapahtuu asynkronisesti, koska tietojen haussa voi kulua paljon aikaa.

3.6.3 Palveluiden haku ja tiedot

Tiedossa olevien palveluiden hakemiseen voidaan käyttää ”GET/service” -pyyntöä, joka palauttaa kaikki palvelut. Palvelujen viimeisimmät muutospäivämäärät voidaan saada ”GET/info”-rajapinnasta. Tämä rajapinta vastaanottaa kolme argumenttia: maan, palvelun ja segmentin.

3.6.4 Palvelun käyttäjien tietojen haku

Käyttäjien haku tapahtuu käyttämällä rajapintaa ”GET /users”. Rajapinta vaatii kolme argumenttia: maan, palvelun ja segmentin.

Palautettava koko voi silti olla suuri, jos käyttäjien tietoja on löytynyt paljon. Esimerkiksi tilien palauttamiseen liittyvä datamäärä voi olla noin 12 megatavua, mikä voi aiheuttaa merkittävää viivettä hitailla yhteyksillä. Palautetun datan kokoa voidaan kuitenkin pienentää pakkaamalla se gzip-formaattiin. Gzip on Jean-loup Gaillyn ja Mark Adlerin luoma pakkausohjelma, jota tukevat kaikki nykyiset selaimet [12].

Spring Bootin avulla voidaan ottaa Gzip-pakkauksen automaattisesti käyttöön muuttamalla sovelluksen ominaisuuksia ja lisäämällä asetuksen "server.tomcat.compression=on" -sovelluksen konfiguraatiotiedostoon. Tämän yhden asetuksen avulla saadaan palvelin pakkaamaan 12 megatavun tiedoston vain 510 kilotavuun.

3.7 Automaattinen tietojen päivitys

Testikäyttäjien tiedot muuttuvat usein, mutta niitä ei haluta hakea uudelleen joka kerta. Testikäyttäjien tietoja tulisi olla mahdollista päivittää manuaalisesti, mutta ihanteellisinta olisi automatisoida mahdollisimman paljon tietojen päivitysprosessia. Automaatio tulisi ajaa vain tiedettäessä testikäyttäjän tietojen muuttuneen. Yksi tällainen tilanne on korttien vanhentuminen.

Kuinka korttien päivitysprosessi voidaan automatisoida? Spring Bootissa on käytössä '@Scheduled' -toiminto, joka mahdollistaa työn ajoittamisen haluttuun aikaan. '@Scheduled' edellyttää cron-työn määrittämistä, jotta työ suoritetaan tiettynä ajankohtana. Cron on ajastuspalvelu Unix-pohjaisille käyttöjärjestelmille, joka mahdollistaa komentojen suorittamisen tiettyinä aikoina [13]. Koska kortit vanhenevat kuukausittain, asetetaan cron-työ ajettavaksi joka kuukausi kuun ensimmäisenä päivänä.

```
@Scheduled(cron = "0 0 0 1 * *")
public void updateOldCardsUsers() {
    updateUsersService.updatedCardsUsers(ValidCountry.DK, ValidSegment.household);
    updateUsersService.updatedCardsUsers(ValidCountry.FI, ValidSegment.household);
    updateUsersService.updatedCardsUsers(ValidCountry.SE, ValidSegment.household);
    updateUsersService.updatedCardsUsers(ValidCountry.NO, ValidSegment.household);

    updateUsersService.updatedCardsUsers(ValidCountry.DK, ValidSegment.corporate);
    updateUsersService.updatedCardsUsers(ValidCountry.FI, ValidSegment.corporate);
    updateUsersService.updatedCardsUsers(ValidCountry.SE, ValidSegment.corporate);
    updateUsersService.updatedCardsUsers(ValidCountry.NO, ValidSegment.corporate);
}
```

Esimerkkikoodi 11. Korttien päivittäminen corn-työn avulla

Scheduled-ajossa haetaan jokaisen maan ja segmentin mahdolliset vanhentuneet testikäyttäjät ja päivitetään näiden tiedot. Esimerkkikoodissa 12 on esitetty, miten etsintä ja päivitys toimivat.

```
public void updatedCardsUsers(ValidCountry country, ValidSegment segment) {
    // Haetaan käyttäjät, joiden korttitiedot on vanhentunut kuukausi sitten
    List<String> usersToUpdate = getOldCardsUsers(country, segment);
    log.info("updating {} - {} cards for users {}", country, segment, usersToUpdate);
    // Päivitetään käyttäjien tiedot
    userUpdateService.updateUsersService("cards", usersToUpdate, country, segment);
}

public List<String> getOldCardsUsers(ValidCountry country, ValidSegment segment) {
    UserDataList data;
    try {
        // Hea käyttäjätiedot Hazelcastista
        data = aService.getUsersServiceDataFromHazelcast("cards", country, seg-
ment).toFuture().get();
    } catch (InterruptedException | ExecutionException e) {
        throw new RuntimeException(e);
    }
    List<String> usersToUpdate = new ArrayList<>();
    // Haetaan päivämäärä, joka on kuukausi sitten
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("MM/yy");
    LocalDateTime now = LocalDateTime.now();
    String date = dtf.format(now.minusMonths(1));

    // Käydään läpi kaikki käyttäjät
    for (User user : data.getUsers()) {
        // Jos käyttäjän tiedoissa on päivitetty kuukausi sitten, lisätään käyttäjä pa-
lautettavaan listaan
        if (user.getData().toString().contains("formatted=" + date)) {
            usersToUpdate.add(user.getUserid());
        }
    }
    return usersToUpdate;
}
```

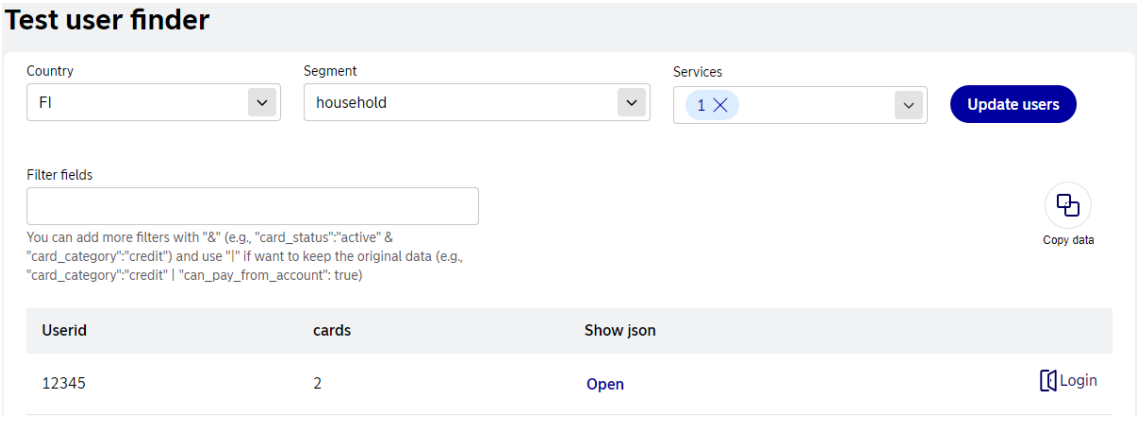
Esimerkkikoodi 12. Vanhojen korttien käyttäjien etsintäfunktio

Työssä haetaan ensin käyttäjätiedot Hazelcastista ja tarkastetaan, onko viime kuussa vanhentuneita kortteja. Mikäli tällaisia kortteja löytyy, lisätään testikäyttäjä listaan, jossa päivitetään näiden testikäyttäjien tiedot.

Manuaalinen päivitys tulee tapahtumaan rajapinnan kautta POST-operaatiolla. Rajapinta vaatii tiedon siitä, mille maalle ja segmentille päivitys tapahtuu. http body tarvitsee tiedon siitä, minkä palvelimen tiedot tulisi päivittää ja mitkä testikäyttäjät päivitetään. Metodin päivitys on asynkroninen, koska päivitystyössä voi kulua todella paljon aikaa, mikäli päivitettäviä käyttäjiä on paljon.

4 Käyttöliittymän kehittäminen

Käyttöliittymän avulla käyttäjä pystyy suodattamaan palveluita haluamallaan tietueilla ja näkemään testikäyttäjien tiedot ennen sisäänkirjautumista. Käyttöliittymä tullaan luomaan Angularin avulla. Angular on Googlen kehittämä web-kehys, jota Nordeassa hyödynnetään web-sovellusten kehityksessä. Tämä mahdollistaa myös Nordean omien yhteiskäyttöön suunniteltujen komponenttien käytön, mikä nopeuttaa kehitysprosessia. Käyttöliittymän tulee olla yksinkertainen käyttää, jotta kaikki pystyisivät hyödyntämään sovellusta, vaikka ei tietäisi ohjelmistosta paljoa.



Kuva 8. Testikäyttäjien etsintänäkymä

4.1 Yleinen näkymä

Käyttöliittymässä käyttäjän tarvitsee valita haluttu maa ja segmentti sekä lisäksi palvelut. Näitä valintoja muuttaessa tulee painaa ”Update users” -painiketta, josta haetaan uudet tiedot näytettäväksi listaan.

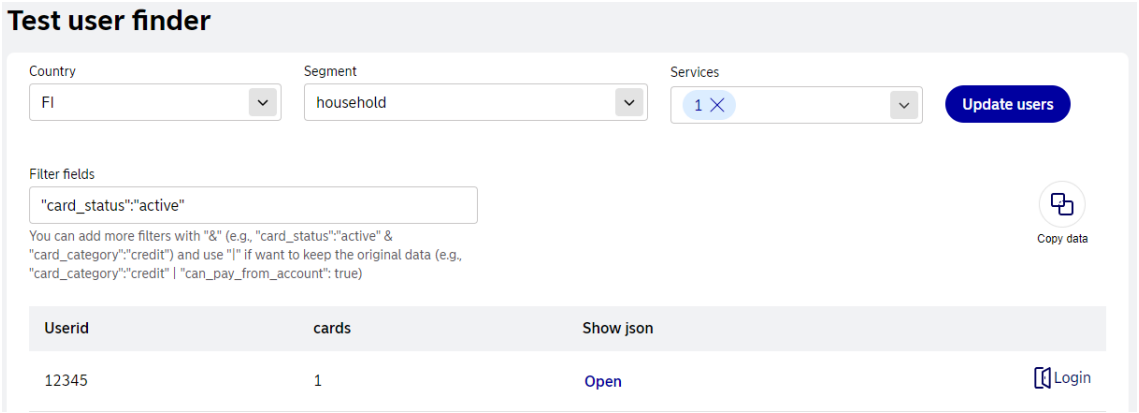
4.2 Käyttäjien suodattaminen

Angularin yksi hyödyistä on putki (pipe), jota pystyy hyödyntämään HTML templatessa ja näin muokkaamaan saatua tietoa suoritusaikana.

```
return filterFirst.map((user) => {
      //Luo uusi objecti, johon tallennetaan suodatetut tiedot
      const data: ServiceInfoMap = {};
      //Käy läpi kaikki valitut palvelut
      for (const dataKey in user.data) {
        //Tarkista onko objectissa kysesistä tekstiä ja ettei suodatin ole negaatio
        if (!JSON.stringify(user.data[dataKey])
              .replace(/\s/g, '')
              .toLowerCase()
              .includes(filterString)
          && !isNotFilter
        ) {
          data[dataKey] = user.data[dataKey];
          continue;
        }
        //Jos suodatin on negaatio, poista ne tiedot, jotka sisältävät suodatinsanan
        if (isNotFilter) {
          const filterStringWithoutExclamation = filterString.replace(/!/g, '');
          data[dataKey] = user.data[dataKey].filter(
            (foundData) =>
              !JSON.stringify(foundData)
                .replace(/\s/g, '')
                .toLowerCase()
                .includes(filterStringWithoutExclamation),
          );
        }
//Jos suodatin ei ole negaatio, poista ne tiedot, jotka eivät sisällä suodatinsanaa
        else {
          data[dataKey] = user.data[dataKey].filter((foundData) =>
            JSON.stringify(foundData)
              .replace(/\s/g, '')
              .toLowerCase()
              .includes(filterString),
          );
        }
      }
      //Palauta käyttäjä, jolla on suodatetut tiedot
      return { userid: user.userid, data } as Testuser;
```

Esimerkkikoodi 13. Käyttäjien suodattaminen Angular-putken avulla.

Esimerkkikoodissa 13 on Angular-putken suodatuskoodi. Yksinkertaisesti suodatintekstistä poistetaan kaikki välilyönnit ja muutetaan kaikki merkit pieniksi. Sama tehdään myös suodatettavalle datalle, joka ensin muutetaan tekstiksi käyttäen JSON.stringify-funktiota. Suodatus käy läpi kaikki testikäyttäjät ja suodattaa tiedot. Käyttäjä voi suodattaa pelkästään tiedot, joissa esiintyy tiettyä tekstiä tai näiden negaatiota.



Kuva 9. Testikäyttäjän etsintä, jossa on suodatettu vain aktiiviset kortit.

Suodatuksessa listaan päivittyy vain ne tiedot, jotka vastaavat suodatintekstiä. Kuvassa 8 testikäyttäjällä 12345 on kaksi korttia, mutta kuvassa 9 niitä on enää yksi. Toinen testikäyttäjän korteista ei siis vastannut suodatustekstiä, jonka vuoksi vain yksi kortti on näytettävissä.

Taulukko 2. Suodatinteksti ja suodatuksen selvennys

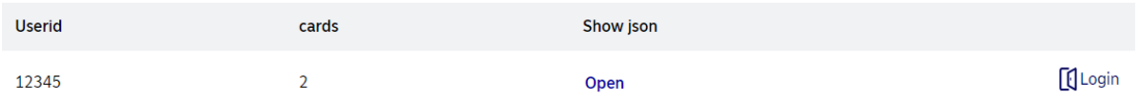
Suodatin	**Palautettavat arvot**
”card_status”:”active”&credit	Kaikki testikäyttäjät, joilla on aktiivinen luottokortti
”card_product”: FI1234|FI3333|FI2333	Kaikki testikäyttäjät, joilla on korttityyppi FI1234, FI3333 ja FI2333

Ja (”&”) -merkin lisäksi suodattimessa voidaan käyttää putkimerkkiä (”|”). Tämä etsii kaikki testikäyttäjän objekteissa olevat tiedot, eikä muokkaa niitä, kuten kuvassa 9 ”&”-merkillä tehty haku. Tällä pystytään löytämään testikäyttäjiä, joilla on mahdollisimman paljon haluttuja tietoja. Tämä nopeuttaa kehityksessä

muutosten varmistamista, koska testikäyttäjiä ei tarvitse vaihtaa useaan kertaan saman toiminnallisuuden varmistukseen.

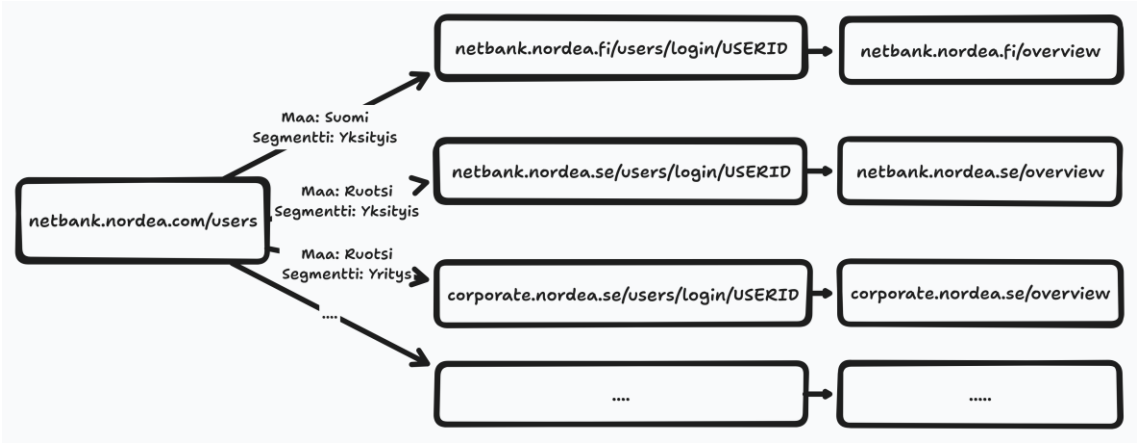
4.3 Muut toiminnallisuudet

Keskitytään aluksi käyttäjäriviin ja niihin tietoihin, mitä siinä näytetään.



Kuva 10. Testikäyttäjän tietorivi, jossa on käyttäjän id, montako korttia löytyi, JSON datan näyttöön käytettävä ”Open”-painike ja ”Login”-painike.

Kuvan käyttäjärivin oikeassa reunassa on ”Login”-painike. Testiympäristöillä on omat sivunsa, joista päästään kirjautumaan sisään. ”Login”-painikkeen ideana on helpottaa kirjautumisprosessia ja nopeuttaa näin testaustyötä.



Kuva 11. Esimerkki kuinka kirjautuminen toimii eri ympäristöihin.

Kuvassa 11 on nähtävissä yksinkertaistettu versio siitä, kuinka kirjautuminen toimii painiketta painamalla. Ensimmäisenä käyttäjä painaa ”Login”-painiketta, joka vie käyttäjän oikealle sivustolle riippuen siitä, mikä maa ja segmentti on valittuna. Sivuston on silti oltava samaa testikäyttäjäsovellusta, jonka avulla päästään kirjautumaan ympäristöön.

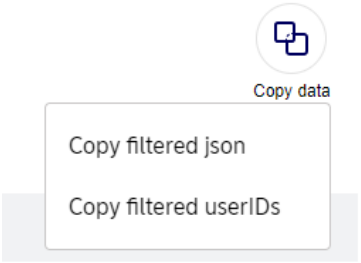
Sivuston URL muuttuu ”/users/login/[userID]”-muotoon. Userid:tä käytetään authentikaatiopalvelun kutsussa, jonka token tallennetaan selaimen muistiin (Session storage). Tallennuksen jälkeen käyttäjä ohjataan ”/overview”-sivulle. Testikäyttäjillä, joilla on useampi sopimus, sopimukset ovat eroteltuna ”:”-merkillä. Jotta oikea token saadaan testikäyttäjälle, autentikaatiopyyntöön pystytään lisäämään tieto siitä, mikä sopimus on kyseessä.

Kuvassa 9 on nähtävissä myös ”Open”-painike. ”Open”-painike avaa ikkunan, jossa on testikäyttäjän oliot. Oliot muuttuvat riippuen suodatuksesta.

Kuva 12. Esimerkki ikkunasta, joka aukeaa "Open"-painiketta painaessa.

Kuvassa 12 nähdään käyttäjän tietoikkunan, jossa näytetään tarkemmin kaikki tiedot, joita palvelu on palauttanut testikäyttäjälle. Kirjautuminen on mahdollista myös tästä ikkunasta ikkunan alhaalla olevan ”Login”-painikkeen avulla.

Suodattaminen yhdellä tekstikentällä toimii, mutta ei ole täydellistä. Jos halutaan kaikki testikäyttäjät, joilla on enemmän kuin 10 korttia, ei tätä ole mahdollista etsiä tekstikentällä. Kuvassa 9 nähdään ”Copy data”-painike tekstikentän vieressä.



Kuva 13. "Copy data" -valikko avattuna

Valikosta voidaan valita kopioitavaksi suodatetut tiedot. Suodatettu tieto kopioidaan leikepöydälle käyttämällä JavaScriptin tarjoamaa ”navigator.clipboard.writeText”-metodia [14]. Esimerkkikoodissa 14 on esitetty, missä muodossa tiedot tallennetaan leikepöydälle.

```
[
  {
    "userid": "12345",
    "data": {
      "cards": [...]
    }
  },
  {
    "userid": "99999",
    "data": {
      "cards": [...]
    }
  }
]
```

Esimerkkikoodi 14. Esimerkki muokatusta tiedosta, jonka käyttäjä pystyy kopioimaan

Esimerkkikoodin 14 avulla pystytään käyttämään mitä tahansa editoria ja suodattamaan testikäyttäjät tarkemmin.

```
[
  {
    "userid": "12345",
    "data": {
      "cards": [{}]
    }
  },
  {
    "userid": "99999",
    "data": {
      "cards": []
    }
  }
].filter(user => user.data.cards.length > 0)
```

Esimerkkikoodissa 15 suodatetaan kaikki testikäyttäjät JavaScriptin avulla, joilla on kortteja. Koodissa suodattuu pois testikäyttäjä 99999, jolla ei ole yhtäkään objektia korteissa.

”Copy filtered users” -painike käyttää samaa JavaScriptin tarjoamaa ”navigator.clipboard.writeText” -metodia kopioidakseen pelkät käyttäjätunnukset. Tämä on tehty helpottaakseen manuaalista tietojen päivitystä, johon tarvitaan lista käyttäjistä. Joissakin tapauksissa saatetaan haluta etsiä tietoja palvelusta vain joiltakin tietyiltä testikäyttäjiltä. Esimerkiksi, mikäli halutaan löytää jokin luottokorttitapahtuma palvelusta, voidaan ensin suodattaa kaikki aktiivisen luottokortin omaavat testikäyttäjät. Tämän jälkeen voidaan palvelimelle antaa lista testikäyttäjistä ja varmistaa näin, ettei turhia pyyntöjä tehdä.

5 Tulokset ja jatkokehitys

Tätä insinöörityötä kirjoittaessa sovellus on ollut käytössä noin kolme viikkoa, ja palaute on ollut positiivista. Uusien toiminnallisuuksien testaaminen on parantunut ja tuotannossa tapahtuvien ongelmien toistaminen testiympäristössä on ollut

mahdollista. Sovelluksen avulla uusien muutosten testaaminen on tehty mahdolliseksi esimerkiksi noin 30 eri korttityyppien kuvia muuttaessa. Muutoksen testaaminen olisi ennen sovellusta ollut lähes mahdotonta testaajien joutuessa kirjautua jokaiselle testikäyttäjälle erikseen tarkastaakseen, onko käyttäjillä kortteja, joiden kuvat olisi pitänyt muuttua. Nyt kuitenkin sovellusta hyödyntämällä testaaja pystyy kirjoittamaan halutun korttityypin suodattimeen ja varmistaa kuvan muuttuneen juuri halutulle kortille.

Aiemmin tiedettiin jokaisesta ympäristöstä löytyvän muutamat testikäyttäjät, joille oli testaamista varten luotu eri korttityyppejä. Sovelluksen avulla löydettyjä testikäyttäjiä on nyt tiedossa noin 150 jokaisessa ympäristössä. Kortit ovat pysyneet myös oikeassa tilassa automaattisen tarkastustyön avulla.

Sovellus on saanut paljon hyvää palautetta eri tiimeiltä. Mobiili- ja palvelinkehittäjät ovat pystyneet luoda parempia testejä halutuille käyttäjille. Testaajat ovat pystyneet varmistaa toiminnon toimivuuden eri käyttäjillä ja eri tuotteilla.

Yksikkötestaus

Testikäyttäjän etsintäohjelman automaattiset testit ovat vähissä, ja tämä heikentää varmistusta siitä, ettei mikään mene rikki muutosten yhteydessä. Jos tiimi haluaa tehdä muutoksia sovellukseen, voi muutosten varmistus olla hankalaa testien puutteen takia.

Automaattinen tietojen päivitys toisille palveluille

Tällä hetkellä pelkästään korttien osalta on automatisoitu vanhentuneiden tietojen tarkastus. Automaattinen päivitys olisi myös hyvä lisätä muille palveluille, jotta manuaalinen tietojen päivityksen tarve vähenisi. Palvelinpyyntöön voitaisiin lisätä kenttä, jossa kerrotaan, millä tietueella on haluttu päivämäärä ja kuinka usein tiedot tulisi päivittää.

Suosikkitestikäyttäjät

Suosikkitestikäyttäjät nopeuttaisivat myös testausta. Testaajat voisivat luoda halutun listan käyttäjistä valmiiksi toiminnon testausta varten ja näin ollen antaa testikäyttäjille tiedot siitä, mihin tarkoitukseen niitä pystyisi käyttämään. Tämän listan pysyisi jakamaan muille, jotka saisivat samat tiedot omaan suosikkilistoihinsa.

Kirjautumisessa uudelleenohjauksen muuttaminen

Kirjautumisen nopeuttaminen oikealle sivulle auttaisi joissain tilanteissa. Tällä hetkellä kaikki käyttäjät viedään etusivulle ”/overview”-kirjautumisen yhteydessä, mutta tätä olisi mahdollisesti hyvä muuttaa halutulle sivulle. Esimerkiksi halutessa testata kortteja, voitaisiin vaihtaa ”/overview” arvoon ”/cards”, joka veisi suoraan korttisivulle kirjautumisen yhteydessä.

6 Yhteenveto

Insinöörityön tavoitteena oli luoda testikäyttäjäsovellus auttamaan Nordean sovellusten testausta. Testikäyttäjät onnistuttiin löytää Splunkin lokiviestien avulla. Sovellukselle tehtiin palvelin Spring Bootin avulla, joka etsi mahdollisia käyttäjiä ja tallensi näiden tiedot Hazelcastiin. Rajapintojen avulla pystytään päivittämään käyttäjien tietoja, lisäämään uusia palveluita ja saamaan halutun palvelun tiedot.

Käyttöliittymä luotiin Angular-kirjaston avulla, jossa pystyy suodattamaan käyttäjiä näiden tietojen perusteella. Käyttöliittymässä käyttäjä pystyy suodattamaan testikäyttäjiä näiden palvelusta palautuvien tietojen avulla. Palveluita pystyy myös valita useamman, jolloin pystytään suodattamaan testikäyttäjät vielä tarkemmin. Käyttöliittymä on myös yksinkertainen, jotta mahdollisimman moni pystyy käyttämään sovellusta. Käyttäjä pystyy valinnaisesti kopioimaan myös kaikki suodatetut tiedot ja suodattamaan itse haluamansa tiedot.

Tämän insinöörityön myötä olen oppinut paljon Spring Bootista sekä palvelupuolen kehityksestä. En ollut aiemmin luonut palvelinta Spring Bootin avulla, joten opeteltavaa oli paljon. Kehitys ei myöskään ollut aina tasaista monen ongelman ilmaantuessa vasta sovelluksen siirryttyä testiympäristöön eikä itse kehitysvaiheessa. Tämä hankaloitti ongelmien löytämistä ja korjaamista. Onnekkaasti Angular-kirjasto oli minulle tuttu jo entuudestaan, joten itse käyttöliittymän kehitys sujui moitteettomasti. Sovellus on työtä kirjoittaessa ollut noin kolme viikkoa saatavilla ja sen on kerännyt muilta kehittäjiltä sekä testaajilta positiivista palautetta. Sovelluksen myötä kehityksessä on säästynyt aikaa paljon ja testausta on pystytty suorittamaan laajemmalta alueelta.

Lähdeluettelo

[1] Nordea, ”Nordea,” 2024. Verkkoaineisto. https://www.nordea.fi/henkiloasiakkaat/tule-asiakkaaksi/asiakkaamme-ovat-paaosin-pohjoismaissa.html. Luettu 20.01.2024.

[2] Bitbucket, ”A brief overview of Bitbucket,” 2024. Verkkoaineisto. https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket. Luettu 20.01.2024.

[3] Splunk, ”Splunk,” 2024. Verkkoaineisto. https://www.splunk.com/en_us/about-splunk.html. Luettu 01.02.2024.

[4] Jenkins, ”Jenkins,” 2024. Verkkoaineisto. https://www.jenkins.io/. Luettu 01.02.2024.

[5] Spring Boot, ”Spring,” 2024. Verkkoaineisto. https://spring.io/projects/spring-boot. Luettu 01.02.2024.

[6] Hazelcast, ”Hazelcast,” 2024. Verkkoaineisto. https://docs.hazelcast.com/imdg/4.2/. Luettu 05.02.2024.

[7] Postman, ”Postman,” 2024. Verkkoaineisto. https://www.postman.com/company/about-postman/. Luettu 05.02.2024.

[8] Angular, ”Angular,” 2024. Verkkoaineisto. https://angular.io/guide/what-is-angular. Luettu 10.02.2024.

[9] Splunk how to connect SKD, ”Splunk documentation,” 2024. Verkkoaineisto. https://dev.splunk.com/enterprise/docs/devtools/java/sdk-java/howtousesdkjava/howtoconnectsdkjava#Log-in-using-your-username-and-password. Luettu 11.02.2024.

[10] Splunk , ”How to display search results,” 2024. Verkkoaineisto. https://dev.splunk.com/enterprise/docs/devtools/java/sdk-java/howtousesdkjava/howtodisplaysearchsdkjava/#To-display-results-using-a-results-reader. Luettu 13.02.2024.

[11] Hazelcast, ”Hazelcast TTL,” 2024. Verkkoaineisto. https://docs.hazelcast.com/cloud/map-configurations#time-to-live. Luettu 15.02.2024.

[12] Gzip, ”Gzip,” 2024. Verkkoaineisto. https://www.gzip.org/. Luettu 16.02.2024.

[13] Wikipedia, ”Cron,” 2024. Verkkoaineisto. https://en.wikipedia.org/wiki/Cron. Luettu 16.02.2024.

[14] Mozilla web docs, ”Clipboard: writeText() method,” 2024. Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/API/Clipboard/writeText. Luettu 20.02.2024.