jamk

Asiakasvirtojen mallintaminen kyberharjoitusympäristössä

Kalle-Eemeli Riuttanen

Opinnäytetyö, AMK
Toukokuu 2023
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Riuttanen, Kalle-Eemeli

Asiakasvirtojen mallintaminen kyberharjoitusympäristössä

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu, 2023, 54 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikka. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Jyväskylän ammattikorkeakoulun yhteydessä toimiva JYVSECTEC oli toteuttajana Elintarvikeketjun kyberturvallisuus -hankkeessa , jonka tavoitteena oli rakentaa elintarvikearvoketjun digitaalista kokonaisuutta mallintava kyberharjoitusympäristö sekä testata sen toiminta pilottiharjoituksessa. Osaksi tätä kokonaisuutta oli tarve suunnitella ja toteuttaa järjestelmäkokonaisuus, joka simuloi yksittäisen kaupparyhmän ruokatavarakauppojen päivittäisiä asiakasvirtoja. Tätä järjestelmäkokonaisuutta lähdettiin suunnittelemaan ja toteuttamaan käyttäen konstruktiivista tutkimusotetta.

Toteutetun palvelun tuli pystyä toimimaan JYVSECTECin kyberharjoitusympäristössä, RGCE:ssä. Realistic Global Cyber Environment (RGCE) on JYVSECTECin kyberharjoitusympäristö, joka mallintaa globaalia internettiä täysin eristetyssä virtuaaliympäristössä. RGCE:n rakenteet, palvelut sekä toiminnallisuudet ovat tehty mahdollisimman lähelle oikeaa internettiä.

Järjestelmäkokonaisuus päädyttiin toteuttamaan kahdella erillisellä järjestelmällä, jotka molemmat toteutettiin Docker-pohjaisina web-palveluina. Ensimmäinen näistä järjestelmistä mallinsi yksittäistä kaupan kassaa ja sen tehtävänä oli luoda ostostapahtumia, joiden tuli sisältää realistisesti mallinnetut tuottet sekä asiakkaan henkilötiedot. Toinen järjestelmä puolestaan toimi keskitettynä varastona ostotapahtumissa käytettäville tuotteille, jotka saapuivat sille säännöllisin väliajoin HTTP-pyyntöinä.

Kehitettyä järjestelmäkokonaisuutta arvioitiin käyttäen kolme tasoista markkinatestiä. Työn tavoitteeksi asetettiin tämän markkinatestin ensimmäinen taso, jonka läpäisy vaati konstruktion käyttöönoton kohdeorganisaation sisällä, koska valmis tuotos oli tarkoitus ottaa käyttöön toimeksiantaja yrityksen sisällä eikä sitä ollut tarkoitus viedä organisaation ulkopuolelle. Työn todettiin läpäisseen tämän tason, koska järjestelmäkokonaisuus otettiin JYVSECTECin sisäiseen käyttöön Elintarvikeketjun kyberturvallisuus -hankkeen pilottiharjoituksessa.

Opinnäytetyön ensisijainen tavoite oli tuottaa järjestelmäkokonaisuus, joka täyttäisi toimeksiantajan asettamat vaatimukset konstruktiivista tutkimusmenetelmää käyttäen. Tämän tavoitteen katsottiin täyttyneen ja järjestelmäkokonaisuudelle onnistuttiin myös tunnistamaan mahdollisia kehitysideoita, joita voidaan JYVSECTECin toimesta kehittää tulevaisuudessa tarvittaessa.

Avainsanat (asiasanat)

JYVSECTEC, kyberturvallisuusharjoitus, kyberharjoitus, Docker, TypeScript, Ansible

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Riuttanen, Kalle-Eemeli

Modeling customer flows in a cyber training environment

Jyväskylä: JAMK University of Applied Sciences, May 2023, 54 pages.

Information and Communications. Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

JYVSECTEC, which operates as a part of Jyväskylä University of Applied Sciences, was the implementer of the Food Chain Cyber Resilience project, the goal of which was to build a cyber training environment that models various systems and processes of the food production and distribution sector. The capability of this environment was to be tested in a pilot exercise. As a part of this project there was a need for a service that simulates the daily customer flows of grocery stores. This service was planned and implemented using a constructive research approach.

The implemented service had to be able to function within JYVSECTEC's cyber training environment, RGCE. Realistic Global Cyber Environment (RGCE) is JYVSECTEC's cyber training environment that models the global internet in a completely isolated virtual environment. RGCE's structures, services and functionalities are made to mirror the real Internet as closely as possible.

The service was designed as two separate systems, both of which were implemented as Docker-based web services. The first of these systems modeled an individual cash register and its task was to create purchase events throughout the day. These events had to contain realistically modeled products and the customer's personal information. The other system acted as a centralized warehouse for the products used in these purchase events. These products arrived at regular intervals as HTTP requests.

The finished product was evaluated using a three-level market test. The goal for this product was to pass the first level of this test. Passing of the first level required the introduction of the construction within the target organization. The product was found to have passed this level because the service was used internally by JYVSECTEC in the pilot exercise of the Food Chain Cyber Resilience project.

The primary goal of the thesis was to produce a service that would meet the requirements set by JYVSECTEC using a constructive research method. This goal was considered to have been met, and possible development ideas for the service were also identified. Based on these ideas JYVSECTEC may continue development of the product in the future if necessary.

Keywords/tags (subjects)

JYVSECTEC, cyber, security, exercise, Docker, TypeScript, Ansible

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet ja termit

API	Application Programming Interface - Rajapinta joka koostuu useista määritellyistä säännöistä, jotka mahdollistavat applikaatioiden välisen kommunikaation (What is an API n.d).
CI/CD	Continous Integration / Continous Delivery - Metodi jonka tavoitteena on automatisoida useita ohjelmistokehityksen vaiheita, kuten testaamista sekä käyttöönottoa (What is CI/CD 2022).
DNS	Domain Name System - Tietokanta, jossa säilytetään internetin verkkotunnuksia sekä käännetään ne IP-osoitteiksi (Lutkevitch 2021).
DOM	Document Object Model - Ohjelmointirajapinta web-dokumenteille, joka mahdollistaa dokumentin rakenteen, tyylin sekä sisällön muokkaamisen ohjelmallisesti (Introduction to the DOM n.d).
EKK	Elintarvikeketjun kyberturvallisuus - Elintarvikeketjun kyberturvallisuuden kehittämisen hanke, jossa mallinnetaan elintarvikearvoketjun digitaalista kokonaisuutta (Elintarvikeketjun kyberturvallisuus n.d, Projektin kuvaus).
ERP	Enterprise Resource Planning - Organisaatioiden käyttämä järjestelmä, jota käytetään päivittäiseen liiketoiminnan hallinnointiin (What is ERP n.d).
Express	Express on kevyt ja joustava web-ohjelmistokehys, joka tarjoaa minimalistisen, mutta vankan joukon ominaisuuksia verkkosovellusten kehittämiseen. Monet suosituimmista web-ohjelmistokehyksistä pohjautuvat Expressiin. (Express n.d.)

HTTP	Hypertext Transfer Protocol - HTTP on sovelluskerroksen protokolla, joka on kehitetty web-palvelimien ja selainten väliseen kommunikointiin (HTTP n.d).
JSON	JavaScript Object Notation - Datan välitykseen käytettävä kevyt tekstipohjainen formaatti, joka koostuu pääasiassa avain-arvo -pareista (Introducing JSON n.d).
ORM	Object Relational Mapping - Ohjelmisto jonka tarkoitus on sovittaa yhteen tietokantoja sekä olio-ohjelmoinnissa käytettäviä data rakenteita (Ellingwood n.d).
REST	Representational State Transfer - Joukko arkkitehtonisia rajoituksia, joiden avulla voidaan määritellä rajapintoja (What is a REST API n.d).
RGCE	Realistic Global Cyber Range - JYVSECTECin harjoitustoiminnassa käytettävä kyberharjoitusympäristö (JYVSECTEC CYBER RANGE – RGCE and solutions 2018).
SSH	Secure Shell - Salattu verkkoprotokolla, jonka avulla käyttäjä voi hallita esimerkiksi etäpalvelimia. (Domantas 2023).
SQL	Structured Query Language - Relaatiotietokantojen tiedonkäsittelyyn käytettävä ohjelmointikieli (What is SQL (Structured Query Language) n.d).
TCP	Transmission Control Protocol - TCP on standardi/protokolla, joka mahdollistaa aplikaatioiden sekä laitteiden välisen kommunikaation verkon yli. Se on suunniteltu dataa sisältävien pakettien lähettämiseen sekä näiden pakettien onnistuneen siirtämisen varmistamiseen. (What is Transmission Control Protocol TCP/IP n.d.)

UUID	Universal Unique Identifier - 128-bittinen arvo jota käytetään erinäisten entiteettien yksilöimiseen esimerkiksi tietokannoissa (Gillis 2021).
YAML	YAML Ain't Markup Language - YAML on datan sarjallistus (engl. serialization) -kieli. Sen yleisin käyttötarkoitus on konfiguraatiotiedostojen kirjoittaminen. (What is YAML 2023)

1 Johdanto

Maailmanlaajuinen digitalisoituminen on ollut näkyvä trendi jo vuosikymmenien ajan, mutta edelleen sen leviäminen jatkuu ja kiihtyy entistä kovempaan tahtiin. Jokaisen henkilön arkipäivää ovat kasvavassa määrin erilaiset digitaaliset hyödykkeet, niin viihteen parissa kuin työtä tehdessä. Oli kyseessä sosiaalinen media, suoratoistopalvelut tai taianomainen chatbotti, joka osaa vastata elämän syvimpiinkin kysymyksiin, on selvää, että moderni ihminen integroituu yhä enemmän digitaaliseen maailmaan tavalla tai toisella.

Luonnollisesti digitalisaatio vaikuttaa myös korkeammalla tasolla. Niin julkisen- kuin yksityisenkin sektorin yritykset ovat jatkuvassa kehityksen kierteessä digitaalisten järjestelmiensä kanssa. Menestyäkseen nykypäivän digitaalisessa maailmassa on välttämätöntä jatkuvasti omaksua uusinta teknologiaa ja sulautua yhä monimutkaisempiin järjestelmiin. Digitalisaatio tuo paljon positiivisia puolia yhteiskuntamme päivittäiseen toimintaan, mutta se tuo mukanaan myös lukuisia uhkia ja potentiaalisia riskejä.

Kyberuhat kehittyvät muun teknologian kehityksen rinnalla kovaa vauhtia, ja mitä enemmän eri toimialat digitalisoivat omaa toimintaansa, sitä enemmän erinäisillä uhkatoimijoilla on mahdollisia kohteita, joihin iskeä. Tämä ilmiö on selvästi havaittavissa pelkästään päivittäisten uutisten seuraamisella. Päivittäin uutisoidaan tietomurroista tai palvelunestohyökkäyksistä, jotka kohdistuvat niin pieniin kuin isoihinkin yrityksiin ja palveluntarjoajiin. Nykypäivänä kyberturvallisuus onkin merkittävä ja ajankohtainen puheenaihe, joka on viime vuosina noussut esiin myös julkisessa keskustelussa.

1.1 Nykytilanne

World Economic Forum julkaisi tammikuussa 2023 raportin kuluneen vuoden kyberturvallisuuden kehityksestä. Raportin tavoitteena oli kartoittaa yrityksien johtohahmojen sekä kyberturvallisuusasiantuntijoiden näkökulmia ajankohtaisista kyberongelmista ja niiden vaikutuksista yrityksiin ympäri maailman. Tässä raportissa tuli esimerkiksi ilmi, että maailman nykyinen geopoliittinen tilanne on vaikuttanut asenteisiin kyberturvallisuuden osalta melko vahvasti. Raportin mukaan 67 prosenttia yrityksien johtohenkilöistä kertoo geopoliittisen epävakauden vaikuttaneen heidän yrityksensä kyberstrategiaan jopa huomattavasti. Suurimmat

riskit nähtiin olevan liiketoiminnan jatkuvuuteen sekä julkiseen kuvaan tai maineeseen vaikuttaminen, ja 43 prosenttia yrityksien johtohenkilöistä uskoivat todennäköiseksi sen, että seuraavan kahden vuoden aikana heidän organisaationsa tulee olemaan jonkin tason kyberhyökkäyksen vaikutuksen alaisena. (Global CyberSecurity Outlook 2023.)

1.2 Kyberharjoitukset

Kyberuhkien lisääntyminen sekä kyberturvallisuus tietoisuuden leviäminen laajasti ovat vaikuttaneet monien palveluntarjoajien varautumiseen mahdollisiin uhkiin ja hyökkäyksiin. Kun puhutaan kyberhyökkäyksiin varautumisesta, harva toimenpide on yhtä käytännönläheinen ja opettavainen kuin kyberharjoitus. Kyberharjoituksissa pyritään mallintamaan realistisia uhkaskenaarioita hallitussa ympäristössä ja testaamaan harjoittelevien organisaatioiden käytänteiden tehokkuutta.

Onnistuneen harjoituksen edellytyksenä on kuitenkin se, että harjoitteluympäristö kokonaisuutena mahdollistaa suunniteltujen uhkaskenaarioiden toteuttamisen. Lisäksi harjoitusympäristön palveluiden tulee mallintaa harjoittelevien organisaatioiden todellisia järjestelmiä mahdollisimman tarkasti. Näiden järjestelmien normaalista toiminnasta harjoitusympäristössä pitää pystyä rakentamaan selkeä kokonaiskuva, jotta uhkaskenaarioiden aiheuttamat poikkeustilat voidaan havaita. Ympäristön järjestelmien välillä kulkevan datan tulee myös olla oikean näköistä, sillä harjoittelevat henkilöt tietävät tarkasti, millaista tietoa heidän järjestelmissään normaalisti liikkuu. Tämän vuoksi poikkeavat datat saattavat aiheuttaa hämmennystä ja hankaloittaa harjoituksen kulkua.

2 Tutkimusasetelma

2.1 Toimeksiantaja

Opinnäytetyön tilaaja JYVSECTEC on Jyväskylän ammattikorkeakoulun yhteydessä toimiva kyberturvallisuuden tutkimus-, kehitys- ja koulutuskeskus. JYVSECTECin asiantuntemukseen kuuluvat muun muassa kyberturvallisuus, tietoturvapoikkeamiin vastaaminen (engl. incident response), nousevat teknologiat (engl. emerging technologies) sekä IT-teknologiat. (Specializing in cyber security expertise n.d.)

2.2 Tavoitteet

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa järjestelmäkokonaisuus, joka simuloi yksittäisen kaupparyhmän ruokatavarakauppojen päivittäisiä asiakasvirtoja sekä tuotemenekkiä keskeisenä osana suurempaa kyberharjoitusympäristöä. Kyseinen harjoitusympäristö kehitettiin osana Elintarvikeketjun kyberturvallisuus -hanketta, jonka tavoitteena oli rakentaa elintarvikearvoketjun digitaalista kokonaisuutta mallintava kyberharjoitusympäristö sekä testata sen toiminta pilottiharjoituksessa (Elintarvikeketjun kyberturvallisuus n.d, Projektin kuvaus).

2.3 Tutkimuskysymykset

Edellisen luvun perusteella voidaan määritellä opinnäytetyön keskeiseksi tutkimuskysymykseksi: "Kuinka toteuttaa järjestelmäkokonaisuus, joka täyttää toimeksiantajan asettamat vaatimukset?" Tämä tutkimuskysymys kiteyttää opinnäytetyön keskimmäisen tavoitteen, joka on toteuttaa vaatimustenmukainen järjestelmäkokonaisuus ja tuottaa lisäarvoa toimeksiantajalle.

Järjestelmäkokonaisuuden suunnittelussa on tärkeää ottaa huomioon myös teknologiavalinnat, mikä johtaa luontevasti kysymykseen: "Mitä teknologioita tulisi käyttää toteutettaessa kyseistä järjestelmäkokonaisuutta?" Teknologiavalintojen huolellinen harkitseminen auttaa varmistamaan, että järjestelmäkokonaisuus saavuttaa sille asetetut tavoitteet ja vaatimukset tehokkaasti ja kestävästi.

Luvussa 1.2 korostettiin, että kyberharjoitusympäristössä liikkuvan datan tulisi olla mahdollisimman realistista. Tämän perusteella voidaan määritellä viimeiseksi tutkimuskysymykseksi: "Kuinka saadaan järjestelmäkokonaisuudessa liikkuva data vastaamaan oikeaa muotoaan?” Tämän tavoitteen saavuttaminen on tärkeää, jotta harjoitusympäristö voi tarjota mahdollisimman autenttisen ja hyödyllisen oppimiskokemuksen osallistujille.

2.4 Tutkimusmenetelmä

2.4.1 Kuvaus

Koska opinnäytetyön tavoitteena oli tuottaa konkreettinen ratkaisu toimeksiantajan käytännön ongelmaan, valittiin työn tutkimusmenetelmäksi konstruktiivinen tutkimus. Konstruktiivinen

tutkimus on hyvin käytännönläheinen tutkimusmenetelmä, joka sisältää suunnittelua, mallintamista, toteutusta sekä testausta. Konstruktiivisessa tutkimuksessa pyritään saamaan tuotokseksi käytännössä hyödynnettävä rakenne, ja on tärkeää, että saavutettu ratkaisu osoittautuu toimivaksi. (Ojasalo, Moilanen & Ritalahti 2015, 65–66.)

Konstruktiivisessa tutkimuksessa on tyypillistä, että käytännön toimijat ovat aktiivisesti mukana toivotun ratkaisun laatimisessa. Konstruktiivinen lähestymistapa myös korostaa tutkimuksen hyödyntäjien ja toteuttajien välistä vuorovaikutusta. (Mts. 65–66.) Vaikka opinnäytetyön teknisen toteutuksen suhteen annettiinkin käytännössä täysi vapaus, toimeksiantaja oli vahvasti mukana järjestelmäkokonaisuuden vaatimusmäärittelyssä ja ajan tasalla kehitystyön edistymisestä.

Ojasalo ja muut (2015, 68) esittävät kehitetyn ratkaisun toimivuuden arviointiin kolme tasoista markkinatestiä, jolla tutkimuksen tuloksia arvioidaan käytännössä. Ensimmäisen tason läpäisy vaatii, että toteutettu ratkaisu toimii toivotusti kohdeorganisaatiossa. Toisen tason läpäisemiseksi ratkaisu tulisi ottaa käyttöön useammassa organisaatiossa. Kolmas ja viimeinen taso edellyttää parempaa menestystä organisaatioilta, jotka ovat ottaneet ratkaisun käyttöön verrattaessa saman toimialan organisaatioihin, jotka eivät ole ottaneet ratkaisua käyttöön. (Mts. 68.) Tämän opinnäytetyön tuotos oli tarkoitus ottaa käyttöön toimeksiantaja yrityksen sisällä, eikä sitä tarkoitus viedä organisaation ulkopuolelle. Siksi tavoitteeksi asetettiin markkinatestin ensimmäisen tason läpäisy.

2.4.2 Riskit

Yksi tyypillisimmistä riskeistä konstruktiivisissa tutkimuksissa on se, että kohdeorganisaation tai toimeksiantajan sitoutuminen projektiin ei kestä (Lukka 2001, Potentiaalisia riskejä). Tämä riski oli kuitenkin käytännössä eliminoitu, sillä työlle oli asetettu selkeät tavoitteet sekä aikataulut toimeksiantajan hankkeen puitteissa.

Kohdeorganisaatiolle relevantti huolenaihe on myös mahdollisten liikesalaisuuksien menettäminen (mt. Potentiaalisia riskejä). Vaikkakin työ liittyi hyvin läheisesti esimerkiksi kohdeorganisaation kyberharjoitustoimintaan, se oli kuitenkin rajattu huolellisesti koskemaan vain kehityksen kohteena olevaa järjestelmäkokonaisuutta. Mahdolliset viittaukset muihin ympäristön osiin käytiin läpi toteuttajan sekä toimeksiantajan kesken tämän riskin minimoimiseksi.

3 Toteutuksen suunnittelu

3.1 Palvelun vaatimukset ja ominaisuudet

Luvussa 2.2 mainittiin, että järjestelmäkokonaisuus tuli toimimaan osana suurempaa kyberharjoitusympäristöä. Elintarvikeketjun kyberturvallisuus -hanke (EKK) kattaakin kaikki elintarvikeketjun osa-alueet mukaan lukien alkutuotannon, elintarvikkeiden jalostajat, logistiikan toimijat sekä kaupan alat. Näin ollen kehitettävän palvelun tulikin siis integroitua osaksi tätä ketjua, mutta kuitenkin olla itsenäinen muuten riippumaton kokonaisuus.

Palvelun toteutuksessa käytettävät teknologiat ja arkkitehtuuriratkaisut olivat suurelta osin valinnanvaraisia. Kuitenkin kaikkien palvelun osien tuli olla verkkopohjaisia järjestelmiä, jotta ne voitaisiin integroida saumattomasti muuhun harjoitusympäristöön. Tämä mahdollistaisi helpon yhteensopivuuden ja vuorovaikutuksen muiden järjestelmien kanssa.

Asiakasvirtojen ja tuotemenekin simuloinnissa keskeiset käsitteet voidaan rajata tuotteisiin ja asiakkaisiin. Nämä kaksi muodostavat ostostapahtumat, jotka puolestaan mahdollistavat kaupan päivittäisen tapahtumavirran kuvaamisen ja analysoinnin. Ostotapahtumien avulla voitaisiin tarkastella esimerkiksi suosituimpia tuotteita sekä asiakkaiden ostokäyttäytymistä, mutta ennen kaikkea kokonaismyyntiä ajan kuluessa.

3.1.1 Skaalautuvuus

Järjestelmäkokonaisuutta tuli olla mahdollista skaalata horisontaalisesti vaivattomasti. Tavoitteena oli, että useampi tällainen kokonaisuus voitaisiin pystyttää esittämään useita kaupparyhmiä, joista jokaisella olisi *x* määrä kauppoja ja jokaisella kaupalla voisi olla *y* määrä kassoja. Näin saataisiin vaihtelevuutta kauppojen sekä ryhmien kokoihin tarvittaessa ja mahdollistettaisiin harjoitusympäristön mukautuminen erilaisiin skenaarioihin.

3.1.2 Tuotteet

Vaikka EKK-hankkeessa pyritäänkin ensisijaisesti mallintamaan digitaalisia järjestelmiä, mallinnetaan harjoitusympäristössä myös fyysisiä tuotteita digitaalisesti. Käytännössä tämä tarkoittaa sitä, että jokainen tuotepakkaus joka ympäristössä liikkuu, on yksi itsenäinen entiteetti,

joka pitää pystyä säilyttämään sellaisenaan. Yksittäinen tuotepakkaus sisältää realistisesti mallinnetut pakkaustiedot sekä myös tuotteen fyysiset ominaisuudet, kuten esimerkiksi ravinnearvot (ks. liite 1). Näiden tuotteiden suunniteltiin liikkuvan säännöllisin väliajoin jalostajilta kaupoille, joten järjestelmäkokonaisuuden piti pystyä vastaanottamaan kauppakohtaisia lähetyksiä sekä säilyttämään niissä tuleva data tietokannassa. Nämä lähetykset toteutettiin HTTP (Hypertext Transfer Protocol)-pyyntöinä, jotka sisälsivät tuotteet JSON-formaatissa (JavaScript Object Notation).

Tärkeä osa harjoitusympäristöä oli mahdollistaa koko elintarvikeketjun täysi jäljitettävyys tuotekohtaisesti. Jokaisen yksittäisen pakkauksen kulkeminen ketjussa tuli olla jäljitettävissä, ja siksi aiemmin mainitut pakkaustiedot sisälsivät muun muassa tuote-erä tiedot. Näiden tietojen avulla tuotteen elinkaari olisi mahdollista selvittää aina raaka-ainetasolle asti.

3.1.3 Kuitit

Oleellinen osa palvelun toimintaa oli myös kuittien generointi tapahtuneista ostotapahtumista. Näiden kuittien piti sisältää vähintäänkin tieto kaikista ostetuista tuotteista hintoineen sekä tiedot siitä mistä kaupasta kyseinen kuitti oli peräisin, sillä kuitit välitettiin eteenpäin ERP–järjestelmälle, jossa niiden avulla voitiin visualisoida kaupparyhmän eri kauppojen päivittäistä toimintaa. ERP–järjestelmästä ja sen roolista harjoitusympäristössä kerrotaan lisää luvussa 3.3.5. Kauppojen tiedot kuten nimi ja osoite saatiin JYVSECTECin Stork-palvelusta, jota kuvataan luvussa 3.3.3. Kuitteihin tarvittavat hintatiedot puolestaan voitiin hakea Tuotetieto-palvelusta, josta kerrotaan lisää luvussa 3.3.4.

Luvussa 1.2 käsiteltiin, miten kyberharjoitusympäristön normaalista toiminnasta tulisi luoda osallistujille selkeä kokonaiskuva. ERP-järjestelmässä kuittien avulla toteutettava visualisointi edistää tämän kokonaiskuvan rakentamista, sillä se auttaa hahmottamaan ja ymmärtämään harjoitusympäristön toimintaa.

3.1.4 Asiakkaat

Ostotapahtumien tuli sisältää simuloituja henkilöitä, joilla tuli olla realistisesti mallinnetut henkilötiedot. Henkilötietoihin lukeutuivat etu- ja sukunimi, osoite, sähköpostiosoite sekä

puhelinnumero ja niitä hyödynnettiin muun muassa luvussa 3.1.2 mainitun jäljitettävyyden mahdollistamisessa. Henkilötietojen lisäksi jokaisella henkilöllä tuli olla myös maksukortti, jonka tiedoilla oli mahdollista suorittaa korttimaksuja. Myös tämä palvelun ominaisuus vaati integroitumisen edellisessä luvussa mainittuun Stork-palveluun.

3.1.5 Maksuliikenne

Luvussa 3.1.4 mainittiin, että jokaisella simuloidulla tuli olla henkilötietojen lisäksi myös maksukortti. Tämän maksukortin tietoja käyttäen luotiin jokaisesta ostotapahtumasta korttimaksu hyödyntämällä harjoitusympäristössä jo olemassa olevaa pankki-infrastruktuuria sekä sen palveluita. Tähän infrastruktuurin integroituminen vaati kuitenkin muutoksia olemassa olevaan järjestelmään, jolla maksutapahtumia oli aikaisemmin luotu.

Maksuliikenteen rooli EKK-harjoitusympäristössä on hyvin samankaltainen kuin aikaisemmin mainitun ERP-järjestelmässä tapahtuvan visualisoinnin. Sen pääasiallinen tarkoitus on rakentaa kokonaiskuvaa ympäristön normaalista toiminnasta sekä toimia myös yhtenä indikaattorina mahdollisten uhkaskenaarioiden tapahtuessa.

3.1.6 Kyberharjoitusympäristö – RGCE

Järjestelmäkokonaisuuden tuli pystyä toimimaan JYVSECTECin kyberharjoitusympäristössä, RGCE:ssä.

Realistic Global Cyber Environment (RGCE) on JYVSECTECin kyberharjoitusympäristö, joka mallintaa globaalia internettiä täysin eristetyssä virtuaaliympäristössä. RGCE:n rakenteet, palvelut sekä toiminnallisuudet ovat tehty mahdollisimman lähelle oikeaa internettiä ja sen ominaisuuksiin lukeutuvat muun muassa realistinen nimipalvelujärjestelmä (DNS) arkkitehtuuri, julkisten avainten hallintajärjestelmä sertifikaateille sekä TOR-verkko. Lisäksi RGCE sisältää monia julkisia palveluita kuten uutissivustoja, keskustelupalstoja, sosiaalisen median alustoja sekä videoiden- ja kuvien jakamispalveluja. RGCE mahdollistaa myös automatisoidun realistisen käyttäjä- sekä verkkoliikenteen simuloinnin, jota voidaan hallita keskitetysti. Tämä ominaisuus mahdollistaa myös bottiverkkojen luonnin esimerkiksi erilaisten verkkohyökkäyksien toteuttamiseksi. (JYVSECTEC CYBER RANGE – RGCE and solutions 2018, 2-3)

RGCE:tä laajennetaan myös jatkuvasti tiettyjä harjoituskokonaisuuksia varten suunnitelluilla järjestelmillä ja järjestelmäkokonaisuuksilla, kuten esimerkiksi terveydenhuollon kyberharjoitusympäristöllä, joka sisältää muun muassa potilastietojärjestelmän, todenmukaisesti mallinnetut tietueet sekä integroidut kansalliset palvelut (JYVSECTEC terveydenhuollon kyberharjoittelun mahdollistajana n.d).

Koska RGCE on täysin eristetty oikeasta internetistä, eivät siellä toimivat palvelut voi olla millään lailla riippuvaisia mistään resurssista, joka pitäisi hakea esimerkiksi jostain julkisesta rajapinnasta. Tämä tuli siis myös ottaa huomioon palvelua kehitettäessä.

3.2 Teknologiat

Tässä luvussa esitellään tekniseen toteutukseen valitut teknologiat sekä perustellaan niiden valinta syyt. Huomion arvoista on se, että koko EKK-hankkeen kehityksessä pyrittiin noudattamaan yhtenäistä teknologiakokonaisuutta, jotta eri järjestelmien väliset integraatiot olisivat mahdollisimman helppoja ja kehitystyö sujuisi järjestelmästä toiseen kivuttomasti. Näin ollen suuri osa näistä teknologioista päätettiin jo aikaisemmin, mutta tästä huolimatta tässä luvussa perustellaan kyseisten teknologioiden käyttö tämän järjestelmäkokonaisuuden kontekstissa sekä mainitaan mahdolliset vaihtoehtoiset teknologiat tarpeen mukaan.

3.2.1 TypeScript

TypeScript on Microsoftin kehittämä vahvasti tyypitetty, olioperustainen (engl. object oriented) ja koottu (engl. compiled) ohjelmointikieli. Vaikka TypeScriptistä voi puhua ohjelmointikielenä, se on oikeastaan ennemminkin JavaScriptin laajennus, joka sisältää kaikki JavaScriptin ominaisuudet lisäten siihen kuitenkin monia ominaisuuksia, kuten esimerkiksi staattiset tyypitykset. Koska TypeScript on koottu ohjelmointikieli, se on luonnollisesti koottava ennen suorittamista. Tämän koonnin lopputulos on puhdasta JavaScriptiä, joten TypeScript toimii kaikilla alustoilla, joissa on tuki JavaScriptille. (TypeScript – Overview n.d.)

TypeScriptin suurin etu JavaScriptiin nähden ovat sen tarjoamat staattiset tyypitykset. Otetaan esimerkiksi yksinkertainen JavaScript-funktio, joka ottaa parametrina user-muuttujan. Oletetaan,

että kyseinen muuttuja on objekti, ja funktio palauttaa sen firstname-atribuutin arvon pienellä alkukirjaimella:

```
function getFirstnameLowercaseJS (user) {
    return user.firstname.toLowerCase()
}
```

Tässä on sama funktio uudelleenkirjoitettuna TypeScriptillä ja varustettuna tyypeillä:

```
interface User {
    firstname: string
}

function getFirstnameLowecaseTS (user: User): string {
    return user.firstname.toLowerCase()
}
```

Molemmat esimerkit ovat täysin päteviä, mutta ero tulee esiin, kun kyseistä funktiota yritetään kutsua virheellisen tyyppisellä argumentilla. TypeScript havaitsee virheen välittömästi ja ilmoittaa käyttäjälle selkeästi, mikä ongelma on:

```
Argument of type 'string' is not assignable to parameter of type 'User'. ts(2345)
View Problem (Alt+F8)    No quick fixes available
getFirstnameLowecaseTS("Hello World")
```

JavaScript puolestaan ei huomaa virhettä, koska se ei ota huomioon funktion vaatiman argumentin tyyppiä:

```
getFirstnameLowercaseJS("Hello World")
```

Kuitenkin, kun tätä koodia suoritetaan esimerkiksi selaimen komentoriviltä, se luonnollisesti aiheuttaa virheen:

```
> function getFirstnameLowercaseJS (user) {
      return user.firstname.toLowerCase()
  }

  getFirstnameLowercaseJS("Hello World")
✖ ▶Uncaught TypeError: Cannot read properties of undefined (reading 'toLowerCase')
      at getFirstnameLowercaseJS (<anonymous>:2:27)
      at <anonymous>:5:1
```

Tämä onkin TypeScriptin suurimpia etuja JavaScriptiin verrattuna ja perimmäinen syy sille, miksi TypeScript alun perin kehitettiin. JavaSriptin käyttötarkoituksien laajeneminen frontendistä backendiin ja aina vain monimutkaisempiin ohjelmistokokonaisuuksiin myös palvelinpuolella toi esiin suuria ongelmia koodin ylläpidon ja uudelleenkäytettävyyden saralla. (TypeScript – Overview n.d.)

Tälle opinnäytetyölle ohjelmointikieltä valittaessa oli tiedossa, että kehitettävä järjestelmä tulisi sisältämään hyvin todennäköisesti useita erilaisia data malleja sekä rakenteita, joiden pitäisi olla täysin yhteensopivia EKK-harjoitusympäristön muiden järjestelmien kanssa. Lisäksi koodin luettavuus oli tärkeä prioriteetti muun muassa jatkokehityksen kannalta. Negatiivisena puolena TypeScriptissä verrattuna JavaScriptiin voi nähdä sen, että koodin tunnollinen tyypittäminen tuo käyttäjälle lisätyötä, mutta pitkässä juoksussa tuo lisätyö maksaa itsensä takaisin moninkertaisesti. Näiden syiden vuoksi TypeScript valikoitui lopulta työn ohjelmointikieleksi.

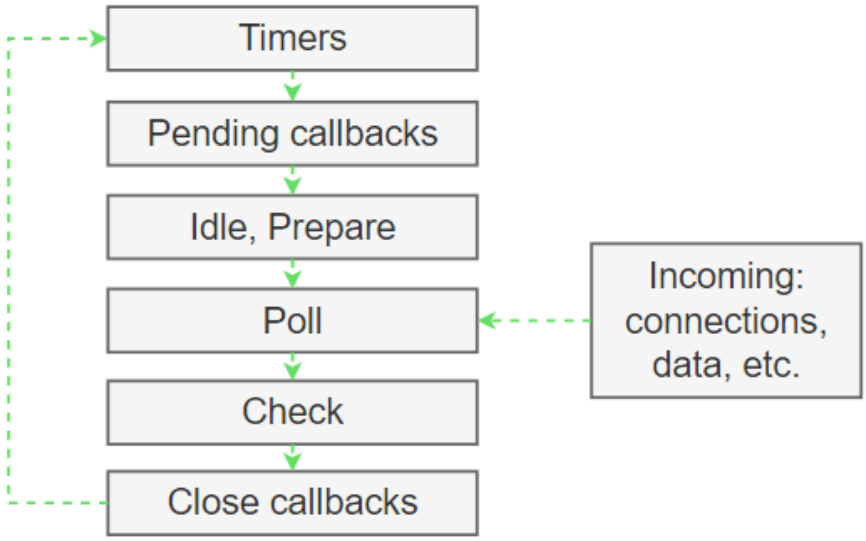
3.2.2 Node.js

Node.js on avoimen lähdekoodin JavaScript suoritusympäristö (engl. runtime). Se on asynkroninen ja tapahtumasilmukkaan (engl. event loop) perustuva sekä nykypäivänä yleisemmästä samanaikaisuus (engl. concurrency) mallista poiketen, jossa hyödynnetään käyttöjärjestelmän säikeitä, Node.js pyörii yhdessä prosessissa. Lisäksi Node.js sekä sen kirjastot ovat suurilta osin rakennettu ei-blokkaavien mallien mukaan. Tämä mahdollistaa tuhansien yhdenaikaisten yhteyksien käsittelyn yksittäisellä palvelimella ilman huolta siitä, että tarvitsisi huolehtia useiden samanaikaisten säikeiden hallinnoimisesta. (Introduction to Node.js n.d;About Node.js n.d.)

Node.js:n keskiössä oleva tapahtumasilmukka voidaan jakaa karkeasti kuuteen eri vaiheeseen:

1. **Timers**: suoritetaan takaisinkutsufunktiot (engl. callback) jotka on asetettu *setTimeout()* tai *setInterval()* funktioilla
2. **Pending callbacks**: suoritetaan takaisinkutsufunktiot, jotka on viivästetty edellisessä silmukan iteraatiossa
3. **Idle, Prepare**: sisäinen vaihe
4. **Poll**: haetaan uudet I/O tapahtumat (engl. event) sekä suoritetaan I/O liittyvät takaisinkutsufunktiot
5. **Check:** suoritetaan *setImmediate()* funktiolla asetetut takaisinkutsufunktiot
6. **Close callbacks:** sulkevat takaisinkutsufunktiot

Tämä jatkuva silmukka mahdollistaa Node.js:n suorittaa ei-blokkaavia I/O (input / output) -toimintoja purkamalla erinäisiä toimintoja järjestelmän kernelille aina kuin vain mahdollista. (The Node.js Event Loop n.d.)



Kuvio 1. Node.js Event Loop (The Node.js Event Loop n.d, muokattu)

Node.js:n yhden säikeen arkkitehtuuri sekä suorituskykyinen tapahtuma silmukka tarjoavat hyvät edellytykset järjestelmien skaalautuvuudelle. Tämä olikin yksi isoimmista eduista, jonka Node.js pystyi tarjoamaan tälle opinnäytetyölle, sillä skaalautuvuus (ks. luku 3.1.1) oli yksi tärkeimmistä vaatimuksista, jotka työlle oli asetettu. Node.js:n myötä pystyttäisiin tarvittaessa myös hyödyntämään lukuisia lisämoduuleja, jotka ovat saatavilla Node Package Managerin (npm) avulla.

3.2.3 Hapi.js

Hapi.js on avoimen lähdekoodin web-ohjelmistokehys (engl. framework), jolla luodaan yleisesti API (Application Programming Interface) -rajapintoja, HTTP-välityspalvelimia sekä websivustoja. Hapi rakennettiin alun perin Expressin päälle, mutta myöhemmin se erkaantui Expressistä täysin. (Kayere 2020.)

Hapi on ominaisuuksiltaan hyvin monipuolinen ohjelmistokehys, joka on rakennettu käyttäjäystävällisyys etusijalla. Sen ominaisuuksiin lukeutuvat muun muassa sisäänrakennettu auktorisointi- ja autentikointirajapinta, koodin eheyden takaaminen versionhallinnan puolella sekä palvelimen kuorman suojaus oletuksena rajoittamalla tulevien pyyntöjen tietosisällön (engl. payload) kokoa ja aikakatkaisemalla roikkuvat pyynnöt (engl. timeout). (The Simple, Secure Framework Developers Trust n.d.)

Hapi valittiin käytettäväksi backend-ohjelmistokehykseksi ensisijaisesti sen vuoksi, että se oli käytössä EKK-hankkeen muissa järjestelmissä jo aikaisemmin ja sen käytänteet olivat tutut sekä kokemukset Hapin käytöstä kehitystiimin sisällä olivat yleisesti ottaen positiivisia. Alun perin valinta perustui Hapin laajuuteen ja varmuuteen, sillä tiedossa oli, että hankkeen kehitys tulee sisältämään useita erilaisia järjestelmäkokonaisuuksia ja näin ollen valitun ohjelmistokehyksen tulisikin olla ominaisuuksiltaan mahdollisimman kattava.

3.2.4 Vue.js

Koska järjestelmäkokonaisuuden ensisijainen tehtävä on toimia ikään kuin taka-alalla simulaattorina muiden järjestelmien tukena, ei sen vaatimuksiin suoranaisesti kuulunut minkäänlaista käyttöliittymää. Mutta koska kyseessä on verkkopohjainen toteutus, oli kuitenkin luontevaa jättää tälle mahdollisuus myös jatkokehitystä silmällä pitäen. Frontend kehityksen kannalta yleisimpiä ratkaisuja ovat erilaiset frontend-ohjelmistokehykset. Nämä ohjelmistokehykset pyrkivät ratkaisemaan yleisimpiä ongelmia frontend kehityksessä, kuten esimerkiksi koodin ylläpidettävyys, kokonaisuuksien pilkkominen pienempiin uudelleenkäytettäviin komponentteihin sekä erinäisten elementtien standardisointi käytettävyyden helpottamiseksi (Pekarsky 2020).

Vue on avoimen lähdekoodin JavaScript-ohjelmistokehys, jonka on kehittänyt Evan You, entinen Googlen työntekijä, joka pyrki luomaan kevyemmän mutta ominaisuuksiltaan samankaltaisen vaihtoehdon Angular-ohjelmistokehykselle (Cromwell 2016). Vue hyödyntää komponenttien kirjoittamiseen malline (engl. template) syntaksia sekä tarkkailee automaattisesti JavaScriptin tilamuutoksia (engl. state changes) ja päivittää DOMia (Document Object Model) näiden muutosten tapahtuessa. Vue-komponenttien rakenne perustuu yleensä Single-File Component -malliin, jossa kaikki yksittäisen komponentin osat on koottu yhteen *.vue*-päätteiseen tiedostoon. (Introduction n.d.)

```
<script>
export default {
  data() {
    return {
      count: 0
    }
  }
}
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

Kuvio 2. Vue.js esimerkki komponentti

Vue valittiin osaksi järjestelmäkokonaisuuden teknologioita pääasiassa sen keveyden sekä ominaisuuksien laajuuden vuoksi. Isona osatekijänä valinnassa oli myös Vuen lisäosaksi kehitetty Vuetify-komponenttikirjasto, joka sisältää laajan valikoiman valmiiksi tehtyjä Googlen Material Design -spesifikaation mukaisia komponentteja (What is Vuetify n.d). Myös Vuen suosio edesauttoi sen valintaa sillä sen myötä Vuesta on tarjolla paljon dokumentaatiota ja muita resursseja kehitystyön tueksi.

Vue 3 on Vuen nykyinen ja viimeisin pääversio. Toteutuksessa päädyttiin kuitenkin käyttämään Vue2-kirjastoa, koska kehityksen alkaessa Vue3 oli vielä betavaiheessa ja muun muassa sen tuki

ulkoisille kirjastoille oli merkittävästi rajoitetumpi. Vue3 ei esimerkiksi tukenut aiemmin mainittua Vuetify-komponenttikirjastoa lainkaan.

Kuten jo aikaisemmin mainittiin ei järjestelmäkokonaisuuden alkuperäisiin vaatimuksiin kuulunut minkäänlaista käyttöliittymää. Tästä huolimatta päädyttiin sellainen kuitenkin myöhemmässä vaiheessa kehittämään, sillä käyttöliittymän todettiin helpottavan järjestelmien toiminnan hahmottamista kyberharjoituksen aikana.

3.2.5 Tietokanta

Tietokantaratkaisua mietittäessä oli vaihtoehtoina kaksi erilaista lähestymistapaa: relationaaliset ja ei-relationaaliset tietokannat. Relaatiotietokannat käyttävät tiedon säilyttämiseen tauluja, jotka voivat jakaa tietoa keskenään muodostaen relaation. Nämä taulut koostuvat niiden rakenteen määrittelevistä sarakkeista sekä varsinaisen datan sisältävistä riveistä. Jokainen rivi taas sisältää pääavaimen, jota käytettäessä toisessa taulussa vierasavaimena voidaan luoda yhteyksiä taulujen välille. Relaatiotietokantojen suurimpia etuja ovat muun muassa relaatioiden takaama tiedon eheys sekä nopeat transaktiot. (Relational vs. Non-relational Databases n.d.) Ei-relationaaliset (NoSQL) tietokannat eivät implementoi samanlaista taulurakennetta kuin relaatiotietokannat ja niiden sisältämän datan ei tarvitse olla yhtä tarkasti määriteltyä sekä yhdenmukaista. NoSQL-tietokannat voidaan jakaa datatyypin mukaan neljään eri malliin:

1. **Document database**: Dataa säilytetään esimerkiksi JSON-objekteina.
2. **Kev-value database**: Jokainen artikkeli sisältää avaimen ja sitä vastaavan arvon.
3. **Wide-column store**: Sisältää tauluja ja rivejä, mutta sarakkeet ovat dynaamisia.
4. **Graph database**: Dataa säilytetään noodeissa sekä särmissä (engl. nodes and edges).

Ei-relaationalisten tietokantojen suurimpiin etuihin lukeutuvat muun muassa niiden horisontaalinen skaalautuvuus, helppokäyttöisyys sekä joustavat datamallit. Nämä tietokannat ovat hyvä valinta esimerkiksi silloin, kun säilytettävää dataa on paljon ja se ei ole normalisoitua. (What is NoSQL n.d.)

Tiedon eheyden säilyminen oli äärimmäisen tärkeää, koska esimerkiksi mahdolliset duplikaatit voisivat aiheuttaa ongelmia muissa ympäristön järjestelmissä. Transaktioiden nopeudella taas voidaan vähentää riskiä siitä, että tietokantakyselyistä muodostuisi mahdollinen pullonkaula

järjestelmää skaalatessa. Suunnitteluvaiheessa tiedostettiin myös, että säilytettävää dataa saattaisi olla paljon, mutta se kuitenkin tulisi olemaan normalisoitua. Näiden edellä mainittujen seikkojen perusteella päädyttiinkin lopulta relaatiotietokanta-pohjaiseen ratkaisuun.

Ensisijaisesti työssä päädyttiin käyttämään PostgreSQL-tietokantaa. PostgreSQL (Postgres) on avoimen lähdekoodin tietokantajärjestelmä, joka pohjautuu SQL-ohjelmointikieleen ja joka voidaan yleisesti mieltää relaatiotietokannaksi. Kuitenkin monista muista relaatiotietokannoista poiketen PostgreSQL tukee sekä relationaalisia että ei-relationaalisia tietotyyppejä. (What is PostgreSQL n.d.) PostgreSQL valikoitui tietokantaratkaisuksi sen suorituskyvykkyyden ja joustavuuden vuoksi. Myöhemmässä vaiheessa kehityksen edetessä huomattiin kuitenkin, että Postgresin lisäksi haluttaisiin vaihtoehdoksi myös jokin kevyempikin tietokantaratkaisu. Postgressiä käytettäessä jokainen järjestelmä, joka tarvitsisi tietokannan, vaatisi vähintäänkin yhden ylimääräisen Docker- kontin (ks. luku 3.2.9). Tämä puolestaan olisi ongelma siinä vaiheessa, kun järjestelmiä alettaisiin skaalata horisontaalisesti. Näin ollen päädyttiin vaihtoehtoiseksi tietokantaratkaisuksi valita SQLite -tietokantamoottori (engl. database engine).

Postgresin tavoin SQLite on myös relaatiotietokanta, mutta muista tietokannoista eroten se säilyttää koko tietokantaa yhdessä tiedostossa. Tämä tekee siitä erinomaisen vaihtoehdon kevyemmille tietokannoille, mutta yksinkertaisuutensa vuoksi SQLite on toiminnallisuuksiltaan ja ominaisuuksiltaan rajoitetumpi kuin muut relaatiotietokannat. SQLite ei esimerkiksi validoi ollenkaan tietokantaan kirjoitettavan datan tyyppiä ja yhden tiedoston rakenteesta johtuen useamman käyttäjän ei ole mahdollista muokata tietokantaa yhdenaikaisesti. (What is SQLite n.d.) Näistä puutteista huolimatta päädyttiin SQLiteen, sillä sen käyttökohteet tulisivat olemaan pääasiassa järjestelmiä, joiden tietokannat olisivat yksinkertaisia ja joissa keveys olisi ehdoton prioriteetti.

3.2.6 ORM / Sequelize

ORM (Object Relational Mapping) yksinkertaistaa applikaation ja relaatiotietokannan välistä vuorovaikutusta luomalla uuden kerroksen (engl. layer) näiden kahden välille käyttämällä erilaisia deskriptoreita. Nämä deskriptorit ovat käytännössä applikaation puolella objekteja, jotka vastaavat tietokannan tauluja ja joiden avulla kehittäjät voivat suorittaa erinäisiä operaatioita

tietokantaan. Esimerkiksi datan päivittäminen, luominen, lukeminen sekä poistaminen voidaan suorittaa ilman, että täytyisi kirjoittaa kyseiset operaatiot valmiiksi SQL-kielellä. (Awati 2023.)

Sequelize on Node.js:lle kehitetty ORM-työkalu, joka tukee useimpia relaatiotietokantoja. Sen toiminnan pohjana toimivat käyttäjän määrittelemät mallit, jotka sisältävät tiedon siitä, minkälaista entiteettiä tai tietokannan taulua kyseinen malli edustaa. Tähän tietoon lukeutuvat esimerkiksi taulun nimi ja sen sarakkeiden nimet sekä tietotyypit. (Sequelize n.d;Model Basics n.d.)

Luvussa 3.2.5 kerrottiin, kuinka tietokantaratkaisua mietittäessä päädyttiin lopulta käyttämään kahta eri relaatiotietokantaa: PostgreSQL sekä SQLite. Sequelizen valinta perustuikin pitkälti siihen, että se sisälsi tuen näille molemmille tietokannoille ja näiden kahden välillä vaihtaminen oli tarvittaessa todella helppoa yksinkertaisella konfiguraation muutoksella. Lisäksi työssä hyödynnettiin myös npm-rekisterissä (ks. luku 3.2.2) tarjolla olevaa sequelize-typescript -moduulia, joka teki muun muassa mallien määrittämisestä typescript-ystävällisemmän:

```
1   import { Table, Column, Model, DataTypes } from 'sequelize-typescript'
2
3   @Table
4   class Person extends Model {
5       @PrimaryKey
6       @Column(DataTypes.UUID)
7       id!: string
8
9       @Column(DataTypes.TEXT)
10      firstname!: string
11
12      @Column(DataTypes.DATE)
13      birthday!: Date
14  }
```

Kuvio 3. Esimerkki sequelize-typescript tietokantamallista

3.2.7 HAProxy (High Availability Proxy)

HAProxy on avoimen lähdekoodin välityspalvelin (engl. reverse-proxy), joka mahdollistaa kuormituksen tasapainottamisen (engl. load balancing) TCP- sekä HTTP-palvelimille (Description

n.d). HAProxya käyttämällä voidaan muun muassa parantaa palveluiden suorituskykyä sekä luotettavuutta jakamalla tulevaa verkkoliikennettä useammalle palvelimelle. (Anicas 2022)

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend web-frontend
    bind *:80
    default_backend web-backend

backend web-backend
    balance roundrobin
    server web-server1 192.168.0.10:8080 check
    server web-server2 192.168.0.11:8080 check
```

Kuvio 4. HAProxy konfiguraatiotiedosto

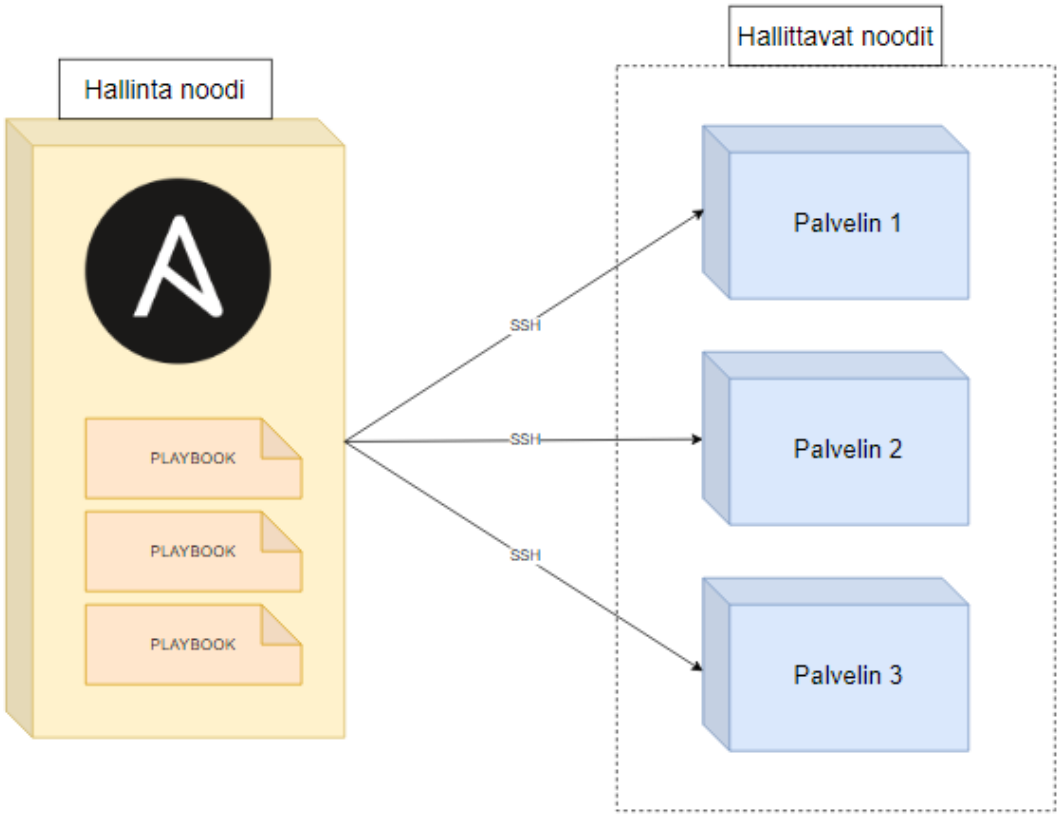
Kuviossa 4 on esitetty konfiguraatiotiedosto yksinkertaiselle HTTP-kuormantasausratkaisulle, jossa on kaksi web-palvelinta (*web-server1* ja *web-server2*). Backend-määrittelyssä (*backend web-backend*) tasapainottaja (*balance roundrobin*) jakaa saapuvat pyynnöt tasaisesti näiden kahden palvelimien kesken. Frontend määrittelyssä (*frontend web-frontend*) HAProxy asetetaan kuuntelemaan kaikkia IP-osoitteita portissa 80 (*bind *:80*) ja kaikki saapuvat pyynnöt ohjataan oletuksena web-backendiin (*default_backend web-backend*).

Kuten luvussa 3.1 mainittiin, järjestelmien tuli olla verkkopohjaisia palveluja ja tästä syystä HAProxy valittiin osaksi toteutusta verkkoliikenteen ohjausta varten. HAProxyn oli myös todettu aikaisemmin EKK-hankkeen kehitystyön aikana helpottavan palveluiden konfigurointia käyttöönoton yhteydessä.

3.2.8 Ansible

Ansible on Pythonilla kirjoitettu avoimen lähdekoodin komentorivin automaatiojärjestelmä. Sitä käytetään järjestelmien konfigurointiin ja applikaatioiden käyttöönottoon sekä päivittämiseen. Ansible käyttää OpenSSH-ohjelmistokokonaisuutta, joka perustuu SSH-protokollaan ja se on suunniteltu mahdollisimman helppokäyttöiseksi käyttöönoton nopeuttamiseksi. (How Ansible works n.d.)

Ansible perustuu rakenteeseen, joka sisältää hallinta noodeja sekä hallittavia noodeja. Ansiblea suoritetaan hallinta noodeilta, joilta käsin käyttäjä voi ajaa esimerkiksi Ansiblen playbook-tiedostoja. Playbook-tiedostot ovat YAML (YAML Ain't Markup Language) -merkintäkielellä (engl. data serialization language) kirjoitettuja ja ne voivat sisältää esimerkiksi konfiguraatioita sekä erinäisiä prosesseja, jotka käyttäjä haluaa suorittaa (Ansible playbooks n.d). Hallittavat noodit taas ovat palvelimia, joille esimerkiksi käyttöönotettavat applikaatiot halutaan pystyttää. Näille palvelimille pusketaan Ansible-moduuleita, jotka suoritetaan hallinta noodilta käyttämällä SSH:ta. Kun moduulit on suoritettu, ne poistetaan palvelimelta. (How Ansible works n.d.)



Kuvio 5. Ansiblen rakenne

Ansiblea päädyttiin käyttämään osana tätä työtä varsinkin kehitystyön aikaisen käyttöönoton ja testaamisen helpottamiseksi. Kehitystyön aikana kehitettäviä palveluita pystytettiin eri vaiheissa erilliseen testausympäristöön, jonka puitteet vastasivat pitkälti varsinaista RGCE:n ympäristöä. Tämä testausympäristö sisälsi myös muita EKK-harjoitusympäristön järjestelmiä, joita vasten kehitettäviä palveluita pystyttiin testaamaan. Jokaiselle kehitettävälle palvelulle luotiinkin oma Ansible-playbook, jonka avulla kyseinen palvelu pystytettiin testausympäristöön.

3.2.9 Docker

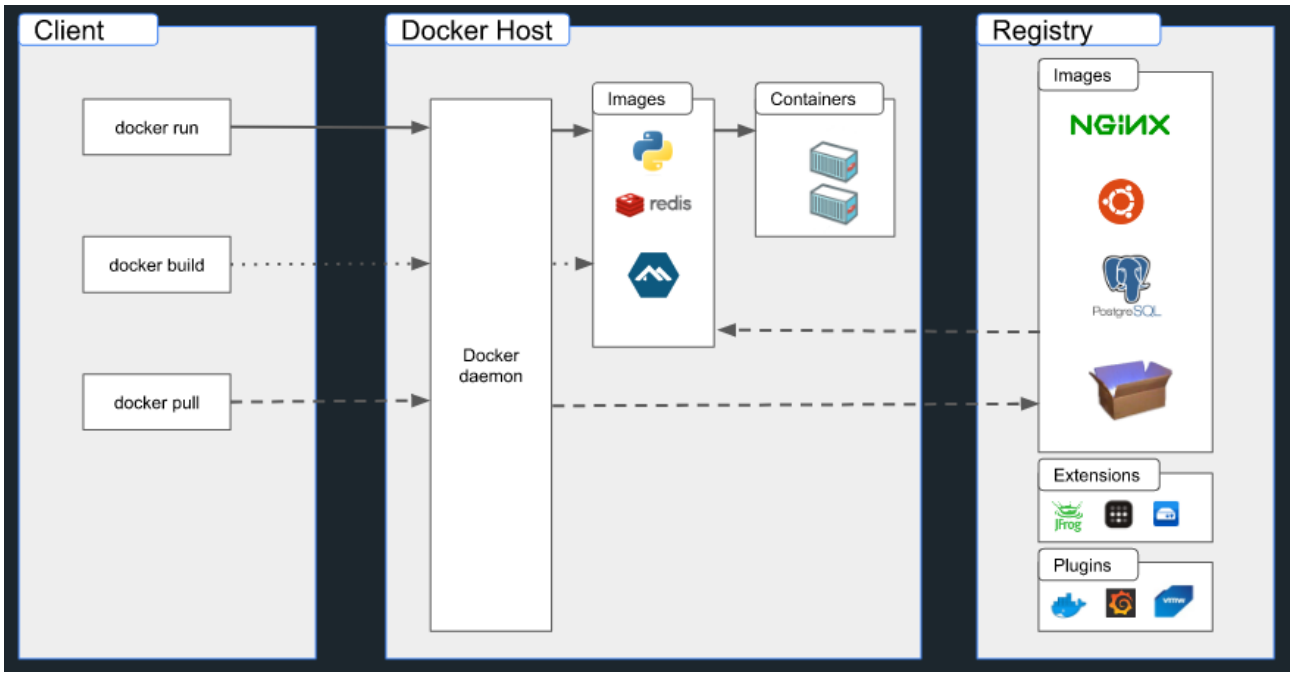
Docker on Go-ohjelmointikielellä toteutettu avoimen lähdekoodin alusta/ympäristö, jonka avulla voidaan pystyttää palveluja käyttämällä kontteja. Docker-kontit ovat virtuaaliympäristöjä, jotka sisältävät kaiken tarvittavan palvelun pyörittämiseen kuten esimerkiksi palvelun lähdekoodin, järjestelmäkirjastot sekä asetukset. Kontitetut palvelut toimivat samanlailla niin Linux kuin Windows ympäristöissäkin, sillä kontit eristävät palvelun muusta ympäristöstä. Käyttäjä voi hallita kontteja Docker-clientin avulla antamalla konteille erilaisia komentoja, joilla voidaan esimerkiksi käynnistää, pysäyttää tai poistaa kontteja. (Docker overview n.d;Use containers to Build, Share and Run your applications n.d.)

```
docker run: Käynnistää kontin
docker stop: Pysäyttää kontin
docker build: Rakentaa kontin
```

Kontit luodaan levykuvien (engl. image) avulla. Nämä levykuvat sisältävät kaiken tarvittavan kontin pystyttämiseen ja ne usein pohjautuvat muihin levykuviin. Esimerkiksi web-palvelua varten tehdyn levykuvan pohjana voisi toimia Ubuntu-käyttöjärjestelmän levykuva, jonka päälle on asennettu Apache-palvelin. Tämän jälkeen siihen lisättäisiin vielä käyttäjän palvelu sekä sen konfiguraatio, jonka jälkeen levykuvasta voidaan tehdä valmis kontti. Levykuvia voidaan säilyttää Dockerin julkisessa Docker Hub -rekisterissä, jonne käyttäjät voivat julkaista omia levykuviaan ja josta näitä levykuvia voi vapaasti hakea omaan käyttöön. Käyttäjien on myös mahdollista luoda omia yksityisiä rekistereitä niin halutessaan. (Docker overview n.d.)

```
docker pull: Hakee levykuvan rekisteristä
docker push: Lataa levykuvan rekisteriin
```

Dockerin arkkitehtuuri perustuu Docker-clientin ja Docker-taustaprosessin vuorovaikutukseen. Docker-client on käyttäjän kosketuspinta Dockeriin, kun taas taustaprosessi hoitaa konttien hallinnoinnin. Client ja taustaprosessi voivat olla samalla isäntäkoneella tai client voi olla yhteydessä taustaprosessiin myös etänä ja ne kommunikoivat käyttäen REST (Representational state transfer) -ohjelmointirajapintaa, UNIX-pistokkeita (engl. sockets) tai verkkorajapintaa. (Mt.)



Kuvio 6. Docker infrastruktuuri (Docker overview n.d)

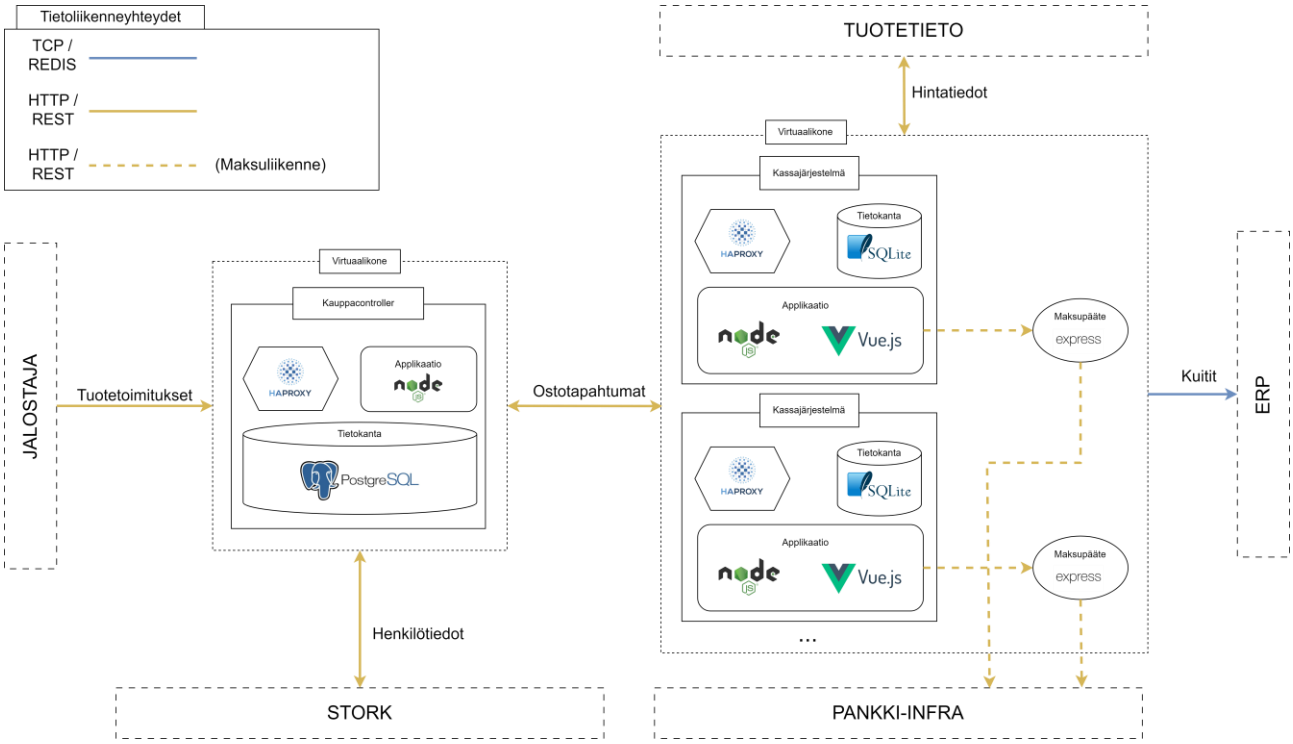
Docker tarjoaa lisäksi myös Compose-työkalun, jonka avulla voidaan konfiguroida ja käynnistää useita kontteja kerralla. Composea käytettäessä käyttäjä konfiguroi palvelunsa tarvitsemat kontit yhdessä YAML-tiedostossa, jonka jälkeen nämä kontit luodaan ja käynnistetään komentoriviltä yhdellä komennolla. (Docker Compose overview n.d.)

Docker valittiin käytettäväksi toteutukseen ensisijaisesti sen keveyden ja helppokäyttöisyyden ansiosta. Suurena osatekijänä oli myös konttien mahdollistama erittäin tehokas horisontaalinen skaalautuvuus, sillä kuten aikaisemmin luvussa 3.1 mainittiin tulisi järjestelmäkokonaisuuden

toteutuksen olla sellainen, että se voitaisiin helposti monistaa tarvittaessa. Myös Dockerin suosio on iso etu, koska tämän myötä Dockerista on saatavilla paljon dokumentaatiota sekä muita hyödyllisiä resursseja kehitystyön tueksi. Compose-työkalu puolestaan tulisi olemaan hyödyllinen palveluiden käyttöönotossa, sillä tiedossa oli, että jokainen palvelu saattaisi itse applikaation lisäksi sisältää mahdollisesti myös erillisen tietokannan sekä välityspalvelimen, joita tultaisiin myös käyttämään Docker-kontteina.

3.3 Palvelun kuvaus

Järjestelmäkokonaisuus päädyttiin toteuttamaan kuvion 7 mukaisella infrastruktuurilla.



Kuvio 7. Palvelun infrastruktuuri

3.3.1 Kauppacontroller

Kuviossa 7 vasemmalla keskellä kuvattuna oleva Kauppacontroller-järjestelmä toimii ostostapahtumien generaattorina sekä keskitettynä varastona kaikkien kauppojen tuotteille. Jokainen Kassajärjestelmä rekisteröi itsensä Kauppacontrollerille käynnistyessään HTTP-pyynnöllä, joka sisältää sille konfiguroidun kaupan nimen, jolloin Kauppacontroller hakee tietokannastaan

kyseisen kaupan tiedot ja palauttaa ne Kassajärjestelmälle. Kaikkien kauppojen tiedot haetaan palvelun käynnistyksen yhteydessä automaattisesti Stork-palvelusta.

Rekisteröinnin jälkeen Kassajärjestelmät kyselevät säännöllisin väliajoin ostotapahtumia Kauppacontrollerilta, joka palauttaa Kassajärjestelmille tapahtuman tuotteet sekä henkilön. Nämä kyselyt toteutetaan myös HTTP-pyyntöinä, joissa lähetetään mukana Kassajärjestelmälle rekisteröinnin yhteydessä palautettu kaupan id. Tuotteet valitaan satunnaisesti kyseisen kaupan tuotteista tietokannasta tämän id:n avulla ja henkilön tiedot, mukaan lukien maksukortti, puolestaan haetaan Storkista.

Tuotteet saapuvat kuvatulla tavalla luvun 3.1.2 mukaisesti HTTP-pyyntöinä JSON-muodossa Kauppacontrollerille kerran päivässä. Saapuvien pyyntöjen yhteydessä tiedot luetaan tietokantaan. Lisäksi pyyntöihin sisältyy tieto siitä, mille kaupalle kyseiset tuotteet tulee varastoida.

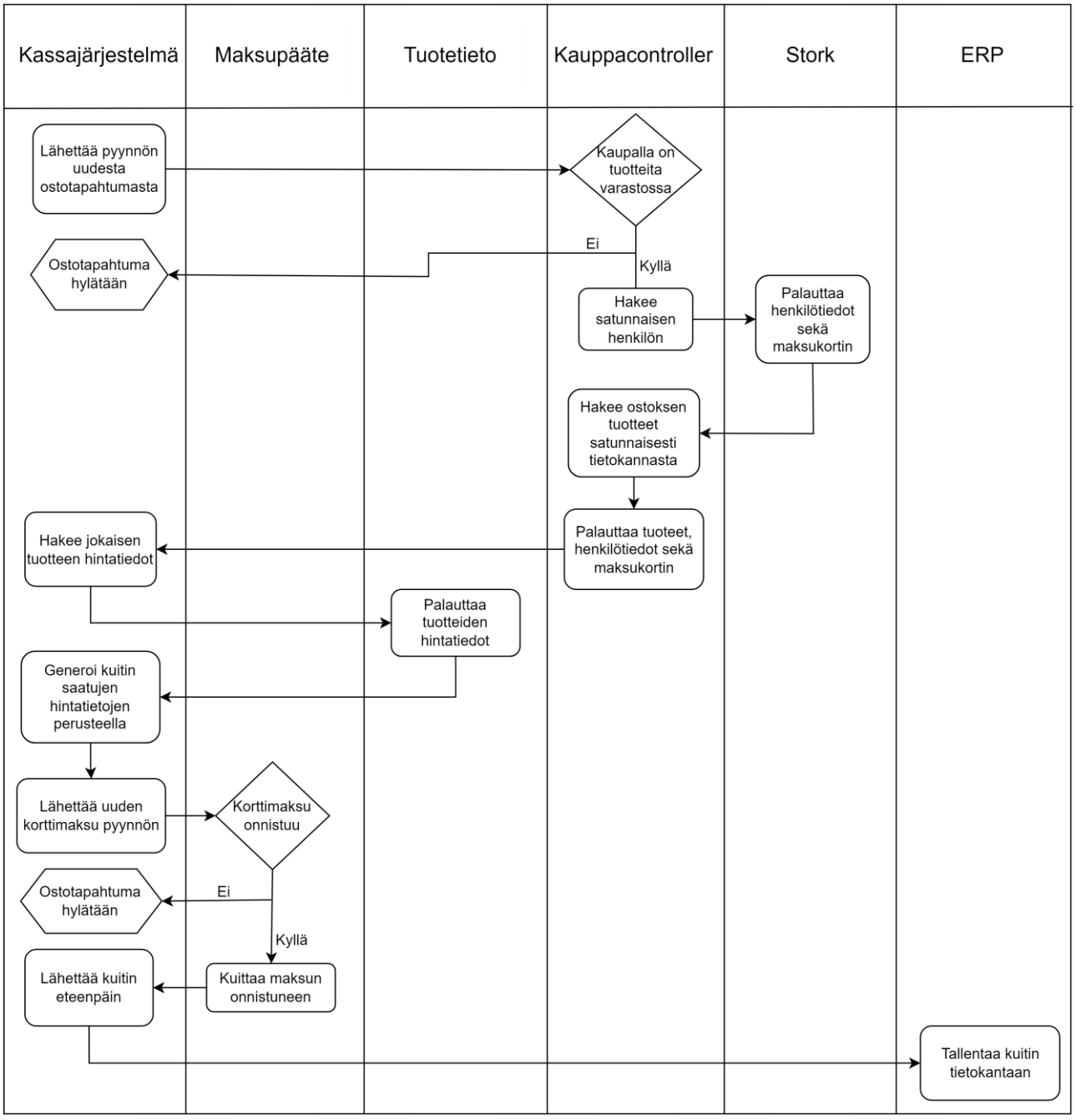
3.3.2 Kassajärjestelmä

Kassajärjestelmä-palvelu puolestaan mallintaa yksittäistä kaupan kassaa ja sen tehtävänä on generoida päivittäistä toimintaa. Näin ollen varsinaista kauppa entiteettiä ei järjestelmäkokonaisuudessa ole ollenkaan, vaan jokainen kauppa koostuu yhdestä tai useammasta Kassajärjestelmästä. Kassajärjestelmille annetaan ympäristömuuttujana kaupan nimi, jolla sidotaan sen ostotapahtumat tiettyyn kauppaan. Tämä ratkaisu mahdollistaa vaatimuksien mukaisen kauppojen skaalauksen, sillä lisäämällä tietylle kaupalle useampia Kassajärjestelmiä, voidaan luontevasti kasvattaa kaupan myyntiä.

Luvussa 3.1.3 kerrottiin, kuinka jokaisesta ostotapahtumasta tulee luoda kuitti, joka lähetetään eteenpäin ERP-järjestelmälle. Näiden kuittien generointiin tarvittavat hintatiedot haetaan tuotekohtaisesti Tuotetieto-palvelusta. Valmiit kuitit taas välitetään ERP-järjestelmälle erillisen Redis-palvelimen kautta. Redis on tietokoneen muistissa säilöttävä avain-arvo-tietokanta, jota voidaan käyttää tiedon säilyttämisen lisäksi muun muassa viestien välittäjänä (engl. broker) (Redis n.d).

Keskitetty tietokantaratkaisu tekee yksittäisestä Kassajärjestelmästä kevyemmän kokonaisuuden, sillä jokainen Kassajärjestelmä ei tarvitse omaa tietokantaa tuotteiden säilyttämiseen. Tämän

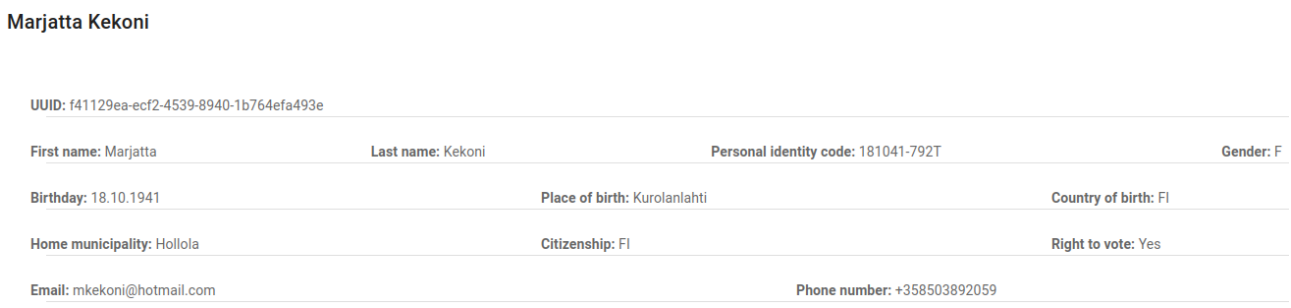
lisäksi myös luvussa 3.1.2 mainitut tuotetoimituksia mallintavat HTTP-pyynnöt jalostajaympäristöstä voidaan nyt ohjata keskitetysti yhteen paikkaan sen sijaan, että niitä pitäisi lähettää usealle eri palvelimelle. Tämä taas on EKK-harjoitusympäristön kokonaiskuvan kannalta huomattavasti selkeämpi ratkaisu.



Kuvio 8. Prosessikaavio ostotapahtumasta

3.3.3 Stork

Stork on JYVSECTECin sisäiseen käyttöön kehitetty entiteetti generaattori palvelu, jonka tehtävä on luoda ja säilyttää realistisesti mallinnettuja entiteettejä, kuten esimerkiksi yrityksiä sekä henkilöitä. Muut harjoitusympäristöjen palvelut voivatkin siis hyödyntää Storkista saatavilla olevia tietoja tarpeen mukaan. Koska nämä tiedot ovat keskitetty yhteen palveluun, on niiden hallinnoiminen helppoa ja näin taataan myös se, että niitä käyttävien järjestelmien väleillä ei tule ristiriitoja tiedon eheyden suhteen. Stork on toteutettu web-palveluna, josta haluttuja tietoja voidaan hakea REST-ohjelmointirajapintaa käyttämällä. Tämän järjestelmäkokonaisuuden osalta hyödynnettiin Storkin henkilötietorajapintoja, sillä vaatimuksissa mainittiin, että jokaisella ostotapahtuman asiakkaalla tulee olla henkilötiedot sekä maksukortti (ks. liite 2). Lisäksi Storkista haettiin myös kauppojen tiedot.



Kuvio 9. Storkissa generoitu esimerkkihenkilö

3.3.4 Tuotetieto

Tuotetieto on EKK-harjoitusympäristön osaksi kehitetty palvelu, joka sisältää kaikkien harjoitusympäristössä liikkuvien elintarvikkeiden tiedot. Jokaisesta elintarvikkeesta löytyvät muun muassa realistisesti mallinnetut ravintoarvot, pakkaustiedot sekä hintatiedot (ks. liite 1). Tuotetieto on toteutettu web-palveluna, josta haluttuja tietoja voidaan kysellä hyödyntämällä sen REST-ohjelmointirajapintaa.

3.3.5 ERP

ERP (Enterprise resource planning) on ohjelmistokokonaisuus, jota organisaatiot käyttävät päivittäisten liiketoimiensa hallintaan. ERP-järjestelmä voi sisältää esimerkiksi kirjanpidon,

hankinnat, projektin- ja riskinhallinnan sekä toimitusketjun toimintojen hallinnan. (What is ERP n.d.)

EKK-harjoitusympäristö sisältää useamman mallinnetun ERP kokonaisuuden ja muun muassa jokaisella kaupparyhmällä on oma ERP-järjestelmänsä. Tämän työn kannalta oleellisin osa-alue näitä järjestelmiä on mahdollisuus tarkastella jokaisen kaupparyhmän päivittäistä toimintaa ostotapahtumien muodossa.

3.3.6 Maksupääte

Maksupääte on web-palvelu, joka toimii linkkinä Kassajärjestelmän sekä RGCE:n pankki-infrastruktuurin välillä. Tätä järjestelmää ei tarvinnut kehittää kokonaan alusta, vaan se toteutettiin muokkaamalla olemassa olevaa palvelua, joka oli aikaisemmin kehitetty JYVSECTECin toimesta.

Tämä palvelu oli alun perin täysin autonominen järjestelmä, joka loi satunnaisia maksutapahtumia ilman ulkoista syötettä. Ostotapahtumia varten tähän järjestelmään lisättiin REST-ohjelmointirajapinta, johon luotiin reitti uusien maksutapahtumien tekemiselle. Tämä reitti vastaanotti maksun summan sekä maksukortin, jolla maksu tulisi suorittaa.

4 Toteutus

Toteutuksen tuotoksia säilytettiin toimeksiantajan Gitlab-versionhallintapalvelussa. Kassajärjestelmälle sekä Kauppacontrollerille luotiin tähän palveluun omat ryhmät, joiden molempien sisälle tehtiin kolme erillistä repositoriota:

- **frontend**: Sisältää frontendin lähdekoodin ja konfiguraation.
- **backend**: Sisältää backendin lähdekoodin ja konfiguraation.
- **aio**: Koonti repositorio (ks. luku 4.4).

Näiden lisäksi Kassajärjestelmä-ryhmän alle luotiin vielä repositorio, joka sisältäisi muokatun Maksupääte-palvelun lähdekoodin sekä tarvittavan konfiguraation.

Tällä menettelyllä varmistettiin se, että toimeksiantajan olisi mahdollista seurata toteutuksen etenemistä esteettä. Gitlab sisältää myös monipuolisen CI/CD-järjestelmän, jota hyödynnettiin muun muassa Docker levykuvien automaattiseen rakentamiseen aina, kun repositorioihin puskettiin uusia muutoksia. Näin luodut levykuvat pystyttiin myös säilyttämään keskitetysti Gitlabissa ja ne olivat tarvittaessa helposti saatavilla esimerkiksi testaamista varten.

4.1 Docker-kontit

Jokainen repositorio sisälsi lähdekoodin lisäksi myös yksilöidyn Dockerfile-tiedoston, jolla määritellään repositoriosta luodun levykuvan rakenne ja sisältö. Dockerfile on teksitiedosto, joka sisältää kaikki komennot, jotka käyttäjä voisi ajaa komentorivillä kyseisen levykuvan kokoamiseksi (Dockerfile reference n.d).

```
1   # build stage
2   FROM node:16 as build-stage
3   WORKDIR /usr/src/app
4   COPY . /usr/src/app
5   RUN npm install
6   RUN npm run build
7
8   # production stage
9   FROM node:16-alpine as production-stage
10  WORKDIR /usr/src/app
11  COPY --from=build-stage /usr/src/app/dist /usr/src/app
12  COPY --from=build-stage /usr/src/app/package.json /usr/src/app
13  COPY --from=build-stage /usr/src/app/certs /usr/src/app/certs
14  COPY --from=build-stage /usr/src/app/static /usr/src/app/static
15  RUN npm install --only=production
16  EXPOSE 3000
17  CMD [ "node", "./index.js" ]
```

Kuvio 10. Kassajärjestelmän backend-repositorion Dockerfile-tiedosto

Kuviossa 10 on kuvattu Kassajärjestelmän backend-repositorion Dockerfile-tiedoston sisältö kaikkine vaiheineen. Tiedosto on jaettu kahteen kokonaisuuteen, joista ensimmäinen (*build stage*)

kokoaa lähdekoodin pohjalta varsinaisen applikaation ja jälkimmäinen (*production stage*) luo levykuvan, josta lopulta luodaan kontti, jota käytetään kootun applikaation ajamiseen.

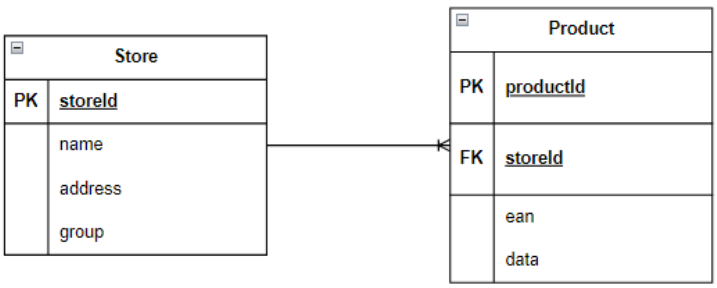
Koontivaihe aloitetaan FROM-ohjeella, joka määrittää mihin olemassa olevaan levykuvaan tämä uusi levykuva pohjautuu. Tässä tapauksessa käytettävä node:16 -levykuva sisältää esimerkiksi Linux-käyttöjärjestelmän sekä Node-suoritusympäristön. WORKDIR-ohjeella kerrotaan missä tiedostopolussa seuraavaksi tulevat ohjeet ja komennot tulisi ajaa. COPY-ohjeella kopioidaan projektin lähdekoodi konttiin, jonka jälkeen RUN-ohjetta käyttämällä ajetaan seuraavat komennot:

- **npm install**: Asentaa kaikki applikaation tarvitsemat lisämoduulit (engl. dependency).
- **npm run build**: Ajaa ennalta määritellyn build-skriptin, joka kokoaa TypeScript-lähdekoodista valmiin JavaScript-applikaation.

Tuotantovaiheeseen kopioidaan koontivaiheessa valmiiksi koottu applikaatio, asennetaan jälleen tarvittavat lisämoduulit, ja lopuksi CMD-ohjeella määritetään komento, joka ajetaan, kun tästä levykuvasta luotu docker kontti käynnistetään. Tässä määritelty *node index.js* suorittaa index.js -tiedoston käyttäen *node*-komentoa, joka puolestaan käynnistää kassajärjestelmä applikaation pohjana toimivan Hapi-palvelimen.

4.2 Kauppacontroller

Kauppacontrollerin toteutuksessa ensimmäinen prioriteetti oli tietokannan rakenteen määrittely sekä tietokantamallien rakentaminen. Luvussa 3.3 kuvattiin Kauppacontrollerin toimivan keskitettynä varastona useamman kaupan kaikille tuotteille. Tämän perusteella pystyttiin määrittämään kaksi erillistä taulua tietokantaan, joista ensimmäinen sisältäisi tuotteet ja toinen kaupat.



Kuvio 11. Kauppacontrollerin tietokannan taulut

Kuviossa 11 on esitetty näiden kahden edellä mainitun taulun sisältö sekä niiden välinen relaatio. Kuviossa vasemmanpuoleinen Store-niminen taulu pitää sisällään kaikkien Stork-palvelusta haettujen kauppojen tiedot. Taulun pääavaimena käytetään generoitua UUID-merkkijonoa. UUID (Universal Unique Identifier) on 128-bittinen arvo, jota käytetään entiteettien yksilöimiseen ja tällä hetkellä jokainen uusi generoitu UUID-arvo on erittäin todennäköisesti uniikki muihin UUID-arvoihin verrattuna ainakin vuoteen 3400 asti (Gillis 2021). Pääavaimen lisäksi taulusta löytyvät myös kaupan nimi, osoitetiedot sekä kaupparyhmän tunnus, jotka kaikki generoidaan Storkissa. Luvussa 3.3 kuvattiin Kassajärjestelmien rekisteröitymisen tapahtuvan kaupan nimeä käyttämällä ja kaupan nimen avulla voidaankin yksittäisen kaupan tiedot hakea tietokannasta.

Kuviossa 11 oikeanpuoleinen Product-niminen taulu puolestaan sisältää kaikki tuotteet. Kuten luvussa 3.1.2 kerrottiin, tulevat kaikki tuotteet Kauppacontrollerille HTTP-pyyntöinä JSON-formaatissa. Tarkemmin otettuna nämä pyynnöt sisältävät listan JSON-objekteja, jotka kaikki kuvastavat yksittäisiä tuotepakkauksia. Näin ollen jokaisesta listan objektista luodaankin yksi uusi rivi Product-tauluun. Taulun pääavaimena käytetään jälleen UUID:tä, jonka lisäksi se sisältää myös ean- sekä data-kentän. Ean-kentässä on jokaisen tuotteen EAN (European Article Number) -koodi ja data-kentässä puolestaan säilytetään kaikkia luvussa 3.1.2 mainittuja tietoja yksittäisestä tuotteesta JSON-objektina.

Näiden kahden edellä mainitun taulun välille on määritelty yhden suhde moneen -yhteys, joka käytännössä tarkoitta sitä, että yhdellä kaupalla voi olla useita tuotteita, mutta yksi tuote voi kuulua vain yhdelle kaupalle. Tämä yhteys on toteutettu Product-taulussa olevan storeId-vierasavaimen avulla, joka mahdollistaa esimerkiksi yksittäisen kaupan tuotteiden hakemisen.

4.2.1 Ostotapahtumien tuotteet

Ostostapahtumien sisältämät tuotteet haettiin kuvion 8 mukaisesti Kauppacontrollerin tietokannasta. Mutta jotta yksittäisen tapahtuman sisältö ei kuitenkaan olisi täysin satunnaista, tehtiin tietokantahaut tiettyjen parametrien rajoissa.

Aluksi määritettiin kaksi muuttujaa, joilla rajoitettiin tuotteiden määrää tapahtumassa. Ensimmäinen näistä muuttujista on *singleProductLimit*, jonka arvo on kokonaisluku, joka kertoo kuinka monta kappaletta yksittäistä tuotenimikettä voi enintään olla yhdessä ostotapahtumassa.

SingleProductLimit-muuttujan oletusarvoksi asetettiin 5. Toisen muuttujan arvo puolestaan kertoo kokonaislukuna, kuinka monta eri tuotenimikettä yksi ostotapahtuma voi enintään sisältää. Tämä muuttuja on nimeltään *maxAmountOfProduts* ja sen oletusarvoksi asetettiin 10. Näitä muuttujia hyödyntämällä asetettiin yksittäisen ostotapahtuman raja-arvot.

Varsinaiset tietokantakyselyt suoritetaan kahdessa osassa. Ensimmäiseksi haettiin ostotapahtuman tuotenimikkeet:

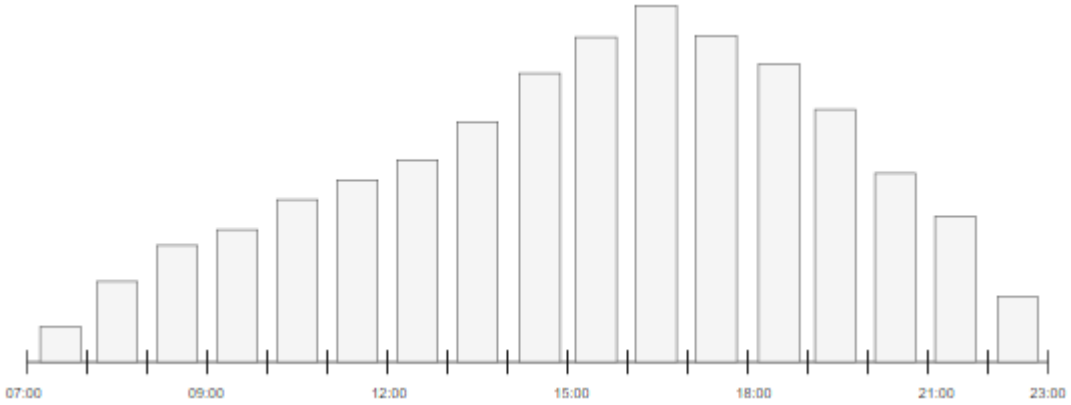
```
SELECT * FROM (SELECT DISTINCT "ean" FROM "Products" /
WHERE "storeId" = <STORE_ID>) AS "distinct" /
ORDER BY RANDOM() LIMIT <maxAmountOfProducts>;
```

Tämä kysely palauttaa satunnaisen listan tuotteita, joilla kaikilla on eri EAN-koodi. Nämä saadut tuotteet käydään seuraavaksi silmukassa läpi ja haetaan tietokannasta varsinaiset tuotteet, jotka palautetaan ostostapahtumaan:

```
SELECT * FROM "Products" WHERE "storeId" = <STORE_ID> /
AND "ean" = <EAN> LIMIT <singleProductLimit>;
```

4.3 Kassajärjestelmä

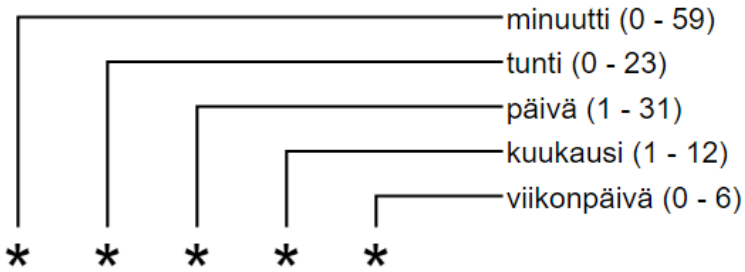
Kassajärjestelmän kohdalla tärkein ominaisuus oli ostotapahtumien luominen, joka käytännössä tarkoitti HTTP-pyyntöjen tekemistä Kauppacontrollerin REST-ohjelmointirajapintaan säännöllisin väliajoin. Kuitenkin jotta kassojen päivittäinen toiminta olisi mahdollisimman realistista, tarvitsi näiden pyyntöjen ajoitukseen kiinnittää tarkemmin huomiota. Ensinnäkin ostotapahtumia ei tulisi tapahtua kauppojen aukioloaikojen ulkopuolella, ja ne eivät myöskään voisi olla täysin satunnaisia tai täydellisen säännöllisiä.



Kuvio 12. Ostotapahtumien määrän vaihtelu kellonajasta riippuen

Kuviossa 12 kuvataan kuinka ostotapahtumien määrän tulisi vaihdella päivän aikana kellonajasta riippuen. Ostotapahtumien määrän pitäisi kasvaa aamusta kohti noin kello neljää, jolloin asiakasvirrat ovat vilkkaimmillaan, jonka jälkeen määrän tulisi taas laskea iltaa lähestyttäessä. Tämänkaltainen vaihtelu mahdollistaisi sen, että kun päivittäisiä ostostapahtumia visualisoitaisiin myöhemmin ERP-järjestelmässä (ks. luku 3.3.5), näyttäisivät ne huomattavasti realistisimmilta.

Ostotapahtumien rajaaminen aukioloajan tuntien sisään päädyttiin toteuttaa hyödyntämällä npm-rekisterissä (ks. luku 3.2.2) tarjolla olevaa cron-moduulia. Tämä moduuli mahdollistaa funktioiden ajoittamisen cron-syntaksia käyttämällä.



Kuvio 13. Cron-syntaksin rakenne

Kassajärjestelmien aukioloajaksi rajattiin 07:00 – 23:00, joka voitiin esittää seuraavanlaisella cron-määritelmällä (engl. expression):

```
0 7-23 * * *
```

Ostotapahtumien tunnittaista määrän vaihtelua varten määritettiin ensin muuttuja *purchaseAmount* , joka esitti kokonaislukuna ostotapahtumien määrää kaikkein kiireisimpinä tunteina. Tämä muuttuja on mahdollista määrittää erikseen ympäristömuuttujalla, mutta oletuksena sen arvoksi asetetaan 100. Tämän lisäksi määritettiin myös lista (engl. array) kertoimia, jotka vastasivat jokaista aukiolotuntia (7-23):

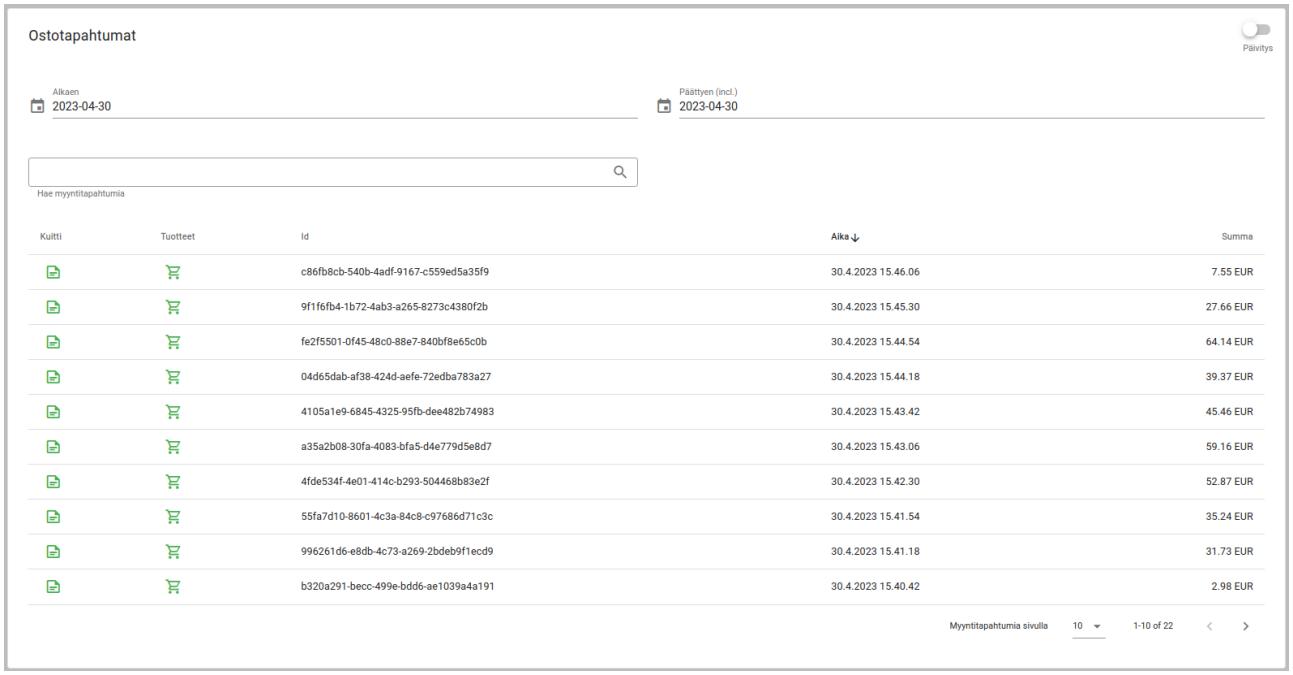
```
[
    0.1     // 7-8
    0.2,    // 8-9
    0.3,    // 9-10
    ...
    1,      // 16-17
    1,      // 17-18
    ...
    0.5,    // 20-21
    0.3,    // 21-22
    0.1     // 22-23
]
```

Tätä listaa ja aikaisemmin mainittua muuttujaa hyödyntäen luotiin funktio, jossa tarkistetaan ensin tämänhetkinen kellonaika JavaScriptin *Date.now()* -funktion avulla, jonka jälkeen haetaan listasta saatua kellonaikaa aikaa vastaava kerroin. Kun tällä kertoimella kerrotaan *purchaseAmount*-muuttujan arvoa, saadaan kyseisen tunnin ostostapahtumien määrä. Tämän jälkeen ajastetaan JavaScriptin *setInterval()* -funktiota käyttäen saatu määrä ostostapahtumia. Yksittäisen ostotapahtuman prosessia vaiheineen on kuvattu kuviossa 8.

Kaikki luodut ostotapahtumat tallennettiin myös Kassajärjestelmän omaan SQLite-tietokantaan, josta niitä voitaisiin hakea käyttöliittymään visualisointitarkoituksiin. Tilan säästämiseksi tietokantaan lisättiin myös ominaisuus, jonka avulla voitiin määrittää, kuinka pitkään ostotapahtumat säilytetään niiden luomisen jälkeen. Tämä mahdollistaa vanhojen ja tarpeettomien ostotapahtumien automaattisen poistamisen tietokannasta, mikä auttaa optimoimaan tilankäyttöä.

4.3.1 Käyttöliittymä

Kassajärjestelmän käyttöliittymästä kehitettiin mahdollisimman kevyt ja sen ainoa käyttötarkoitus oli tarjota mahdollisuus käyttäjälle tarkastella ostostapahtumien historiaa. Tämä toiminnallisuus toteutettiin yksinkertaisella taulunäkymällä, jossa käyttäjällä on myös mahdollisuus rajata ostotapahtumia joko päivämäärällä tai hakea tiettyjä tapahtumia id:n avulla. Näkymän rakentamiseen käytettiin pääasiassa luvussa 3.2.4 mainitun Vuetify-kirjaston komponentteja.

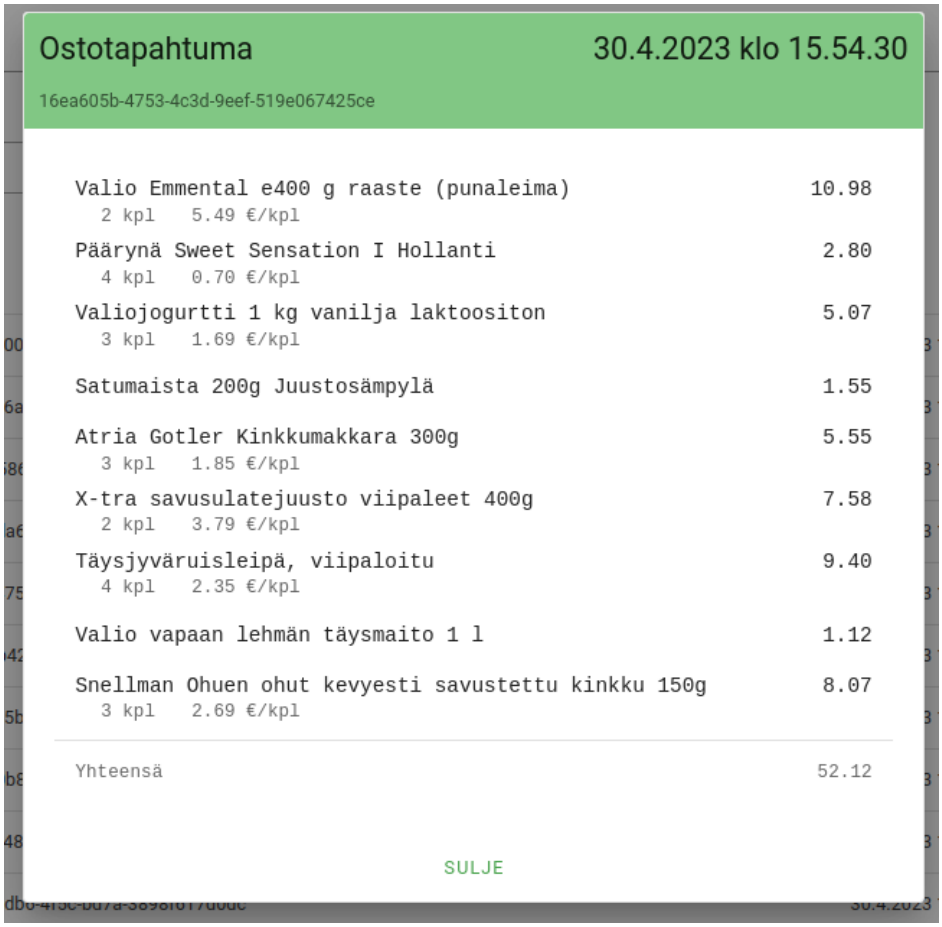


Kuvio 14. Kassajärjestelmän käyttöliittymä

Kuviossa 14 jokaisella taulun rivillä näkyy ostotapahtuman yksilöllinen id, tapahtuman aika sekä kokonaissumma. Oikeassa yläkulmassa olevasta päivitys-napista voidaan käynnistää tai sammuttaa näkymän automaattinen päivitys. Päivityksen ollessa päällä haetaan uudet ostotapahtumat kymmenen sekunnin välein.

Taulun aika-sarakkeen avulla ostotapahtumat voidaan järjestää aikajärjestykseen ja jokaisen rivin oikeassa laidassa näkyy kyseisen tapahtuman tuotteiden hintojen summa. Tuotteet-sarakkeen ikonista aukeaa näkymä, jossa näytetään ostotapahtuman sisältämät fyysiset tuotteet ja muun muassa niiden pakkaustiedot JSON-formaatissa (ks. liite 1). Klikkaamalla rivin vasemmassa

reunassa olevaa kuitti-ikonia puolestaan käyttäjälle avautuu näkymä, jossa esitetään ostotapahtumasta generoidun kuitin tiedot tarkemmin:



Kuvio 15. Yksittäinen ostotapahtuma

Kuviossa 15 voidaan nähdä konkreettisesti aikaisemmin esitellyn ostostapahtumien generoinnin lopputulos sekä jokaisen tuotteen kohdalla näkyvät hintatiedot, jotka on haettu Tuotetieto-palvelusta (ks. luku 3.3.2).

4.4 AIO – All in one

Järjestelmien käyttöönoton helpottamiseksi ja skaalautuvuuden varmistamiseksi pyrittiin Docker-konttien määrä minimoimaan järjestelmäkohtaisesti. Tämän tavoitteen saavuttamiseksi esimerkiksi Kassajärjestelmissä päätettiin käyttää SQLite-tietokantaa PostgreSQL:n sijaan. Huolimatta tästä valinnasta, jokainen Kassajärjestelmä vaati edelleen erilliset frontend- ja backend-kontit.

Luvussa 4.1 esiteltiin Kassajärjestelmän backend-repositorion Dockerfile-tiedosto, joka oli jaettu kahteen erilliseen vaiheeseen käyttäen eri pohjalevykuvia. Vaiheiden välillä tiedostojen siirtämiseen hyödynnettiin COPY-ohjetta. Samanlaista rakennetta käyttäen voidaankin luoda myös aio-levykuva.

```
1   FROM <Kassajärjestelmä_frontend_repositorio> as front
2   FROM <Kassajärjestelmä_backend_repositorio> as back
3   FROM node:16-alpine
4   WORKDIR /usr/src/app
5   COPY --from=back /usr/src/app /usr/src/app
6   COPY --from=front /usr/share/nginx/html /usr/src/app/static
7   RUN npm install --only=production
8   EXPOSE 80 443
9   CMD [ "node", "./index.js" ]
```

Kuvio 16. Kassajärjestelmän aio-repositorion Dockerfile-tiedosto

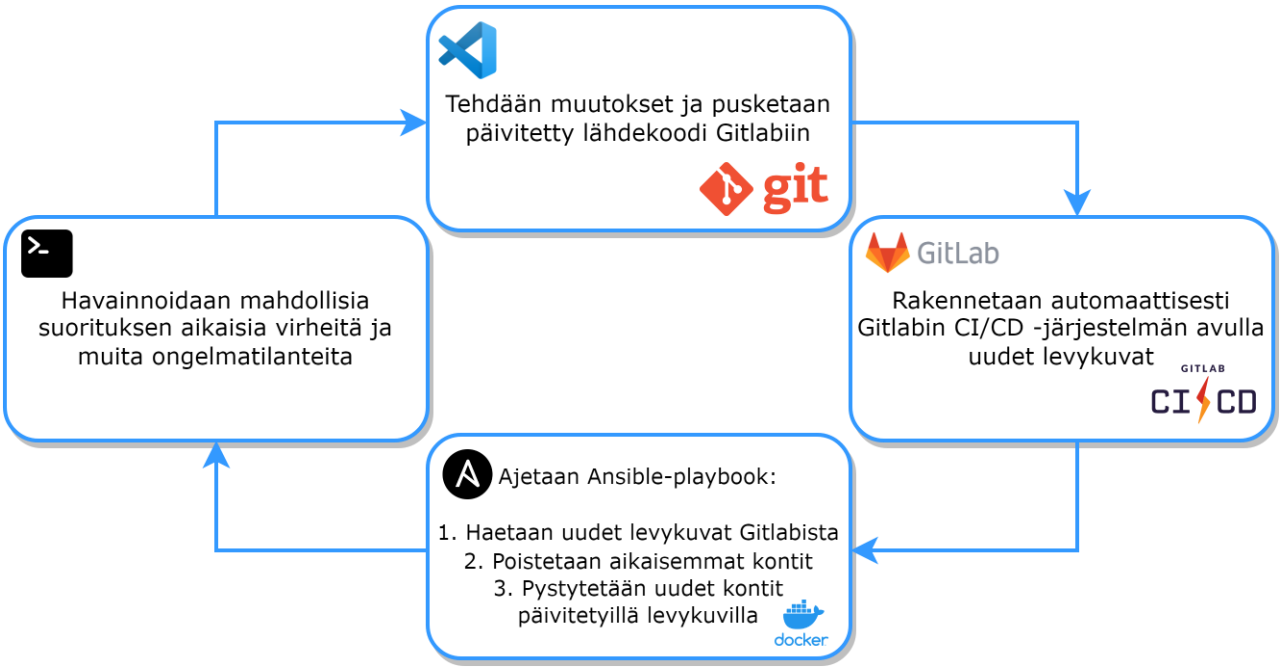
Kuvion 16 Dockerfile-tiedostossa haetaan ensin Gitlabista Kassajärjestelmän frontend-repositoriosta levykuva, joka sisältää kootun Vue-frontendin sekä backend-repositoriosta levykuva, joka puolestaan sisältää kootun Hapi-applikaation. Tämän jälkeen kootut lähdekoodit kopioidaan samaan pohjaan, luoden lopputuloksena yhden levykuvan, joka sisältää kaiken tarvittavan järjestelmän pyörittämiseen.

5 Käyttöönotto

5.1 Testaus

Kuten luvussa 3.2.8 mainittiin, järjestelmäkokonaisuutta testattiin jatkuvasti kehityksen aikan pystyttämällä sen osia Ansiblen avulla testausympäristöön. Tässä ympäristössä ajettiin koko kehityksen ajan vähintään yhtä instanssia molemmista järjestelmistä Docker-kontteina samalla virtuaalikoneella. Luvussa 4 kerrottiin, kuinka järjestelmistä rakennettiin uusia levykuvia kehityksen edetessä Gitlabin CI/CD-järjestelmän avulla. Näin ollen jokaisen isomman päivityksen yhteydessä ajettiin järjestelmille luodut Ansible-playbookit, jotka automaattisesti hakivat aiemmin mainitulle virtuaalikoneelle uudet levykuvat Gitlabista, poistivat aikaisemmat Docker-kontit ja pystyttivät niiden tilalle päivitetyt kontit uusien levykuvien avulla.

Tällä jatkuvalla käyttöönotolla mahdollistettiin järjestelmien tehokas testaaminen koko kehitysprosessin ajan. Koska kyseessä oli täysin autonomisesti pyörivä järjestelmäkokonaisuus, erillisen testausympäristön suurimpia etuja olikin se, että nämä järjestelmät voitiin huoletta jättää ajoon ympärivuorokautisesti myös työskentelyaikojen ulkopuolella. Näin maksimoitiin mahdollisuus törmätä suorituksen aikaisiin virheisiin tai muihin ongelmatilanteisiin.



Kuvio 17. Testauksen aikaisen jatkuvan käyttöönoton vaiheet

Jatkuva käyttöönotto oli osaltaan myös jopa vaatimus kehitystyölle, sillä näiden järjestelmien rinnalla kehitettiin myös muita EKK-harjoitusympäristön järjestelmiä, jotka olivat yhteydessä näihin järjestelmiin ja niiden rajapintoihin.

5.2 Pilottiharjoitus

Järjestelmäkokonaisuuden varsinainen käyttöönotto tapahtui JYVSECTECin järjestämässä Elintarvikeketjun kyberturvallisuus -hankkeen pilottiharjoituksessa, joka pidettiin Jyväskylän ammattikorkeakoulun Lutakon kampuksella 28-29.3.2023. Tätä harjoitusta varten RGCE-harjoitusympäristöön pystytettiin kaksi kaupparyhmäkokonaisuutta, jotka molemmat sisälsivät yhden Kauppacontroller-järjestelmän sekä kolmekymmentä Kassajärjestelmää. Kauppacontroller-järjestelmät asetettiin omille virtuaalikoneilleen ja Kassajärjestelmät jaettiin useammalle

virtuaalikoneelle niin, että yksittäisellä virtuaalikoneella oli neljä Kassajärjestelmää maksupäätteineen. Docker-pohjaisen ratkaisun myötä useampikin järjestelmä olisi tietysti voitu sijoittaa samalla virtuaalikoneelle, mutta tämä jaottelu tehtiin täysin harjoitusteknillisistä syistä johtuen.

Kaikki EKK-harjoitusympäristön järjestelmät pystytettiin RGCE:hen useampi viikko ennen varsinaista harjoitusta, jotta viimeiset testaukset pystyttiin suorittamaan täysin harjoitusta vastaavissa olosuhteissa. Vaikkakin kehityksen aikainen testausympäristö olikin monella tapaa hyvin samanlainen lopullisen RGCE-ympäristön kanssa, suurin ero näiden kahden välillä oli eri järjestelmien välisen dataliikenteen määrä. Tämä ero johtui yksinkertaisesti siitä, että kaikkia EKK-harjoitusympäristön järjestelmäkokonaisuuksia skaalattiin huomattavasti ylöspäin pilottiharjoitukseen verrattuna testausympäristöön. Konkreettinen esimerkki tämän skaalauksen seurauksista huomattiinkin nopeasti Kauppacontroller-järjestelmässä.

Luvussa 3.1.2 kerrottiin kuinka Kauppacontroller-järjestelmä vastaanottaa tuotetoimituksia HTTP-pyyntöinä. Samassa luvussa kuvailtiin myös, kuinka yksittäinen tuote sisältää paljon tietoa, joten näiden pyyntöjen koko saattoikin helposti kasvaa satoihin kilotavuihin testauksen aikana. Luvussa 3.2.3 puolestaan esiteltiin järjestelmien toteutuksessa käytetty Hapi-ohjelmistokehys ja mainittiin, kuinka Hapi rajoittaa tulevien pyyntöjen tietosisällön kokoa osana sen palvelimen kuorman suojausta. Tietosisällön koon raja on oletuksena asetettu yhteen megatavuun ja ympäristön skaalauksen seurauksena siihen törmättiin satunnaisesti tuotetoimituksien yhteydessä. Pyyntöjen tietosisällön koon ylittäessä yhden megatavun pyynnöt epäonnistuvat ja niihin vastataan statuskoodilla 413 (Content Too Large). Tämä ongelma kuitenkin onnistuttiin ratkaisemaan, sillä Hapi tarjoaa mahdollisuuden yliajaa tämän rajoitteen yksittäisen reitin kohdalla muuttamalla kyseisen reitin asetuksista löytyvää *payload.maxBytes* arvoa:

```
options: {
    payload: {
        maxBytes: 2097134 // Default: 1048567
    },
    ...
}
```

Pilottiharjoituksen aikana tarkkailtiin kaikkia EKK-harjoitusympäristön järjestelmiä, jotta niistä voitaisiin tunnistaa mahdollisia kehityskohteita sekä havaita mahdollisia puutteita. Kassajärjestelmien sekä Kauppacontrollerien osalta ei kuitenkaan harjoituksen aikana havaittu välitöntä tarvetta korjauksille tai kehitystoimenpiteille.

6 Jatkokehitys

6.1 Asiakasprofiilit

Luvussa 4.2.1 kuvattiin ostostapahtumien generoinnin prosessia Kauppacontroller-järjestelmässä. Tässä luvussa mainittiin myös, kuinka tapahtuman tuotenimikkeet sekä määrät haettiin tietokannasta satunnaisesti tiettyjen parametrien rajoissa. Tällä menettelyllä saavutettiin määrällisesti järkeviä sekä kohtuullisen realistisia ostotapahtumia suuressa kuvassa, mutta tarkemmin tarkastellessa yksittäisen ostostapahtuman sisältö ei kuitenkaan todennäköisesti näytä kovinkaan normaalilta. Kuvion 15 ostotapahtuma vaikuttaa ensisilmäyksellä uskottavalta, mutta esimerkiksi neljä pussia täysjyväruisleipää lähestyy jo epäuskottavuuden rajaa. Satunnaisen generoinnin vuoksi on kuitenkin täysin mahdollista, että osasta ostotapahtumia tulee täysin epäloogisia ja epärealistisia. Kärjistetty esimerkki tästä voisi olla tapahtuma, joka koostuu pelkästään viidestätoista maitopurkista.

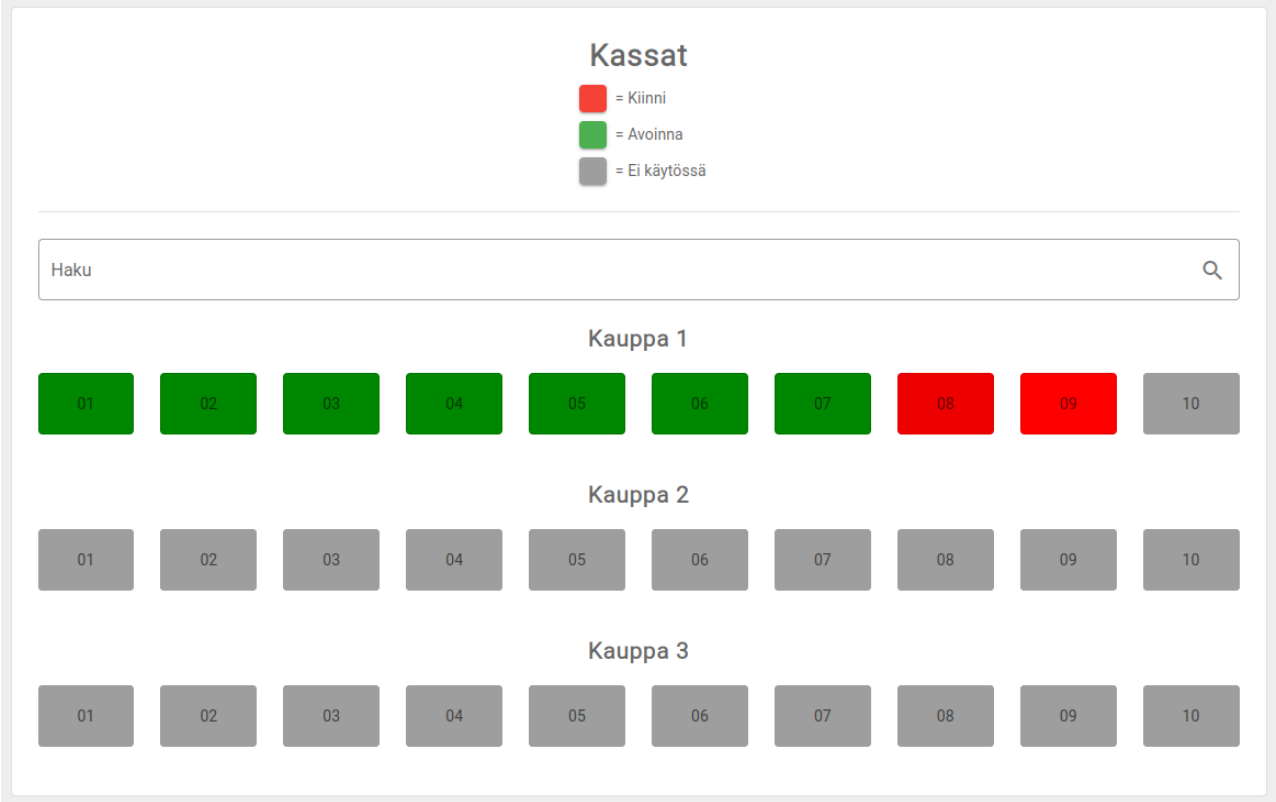
Kehitysideana tämän ongelman ratkaisuksi on Kauppacontrollerin osaksi suunniteltu eräänlaista asiakasprofiilien implementointia. Käytännössä Kauppacontrollerille määritettäisiin alustavasti x määrä profiileja, joissa painotettaisiin tiettyjä tuotenimikkeitä sekä tuotenimikkeiden yhdistelmiä. Kun näiden profiilien avulla sitten luotaisiin ostostapahtumia, nämä tuotteet sekä yhdistelmät valittaisiin huomattavasti todennäköisemmin tapahtumaan. Haettaessa ostostapahtumaan satunnainen henkilö Stork-palvelusta (ks. luku 3.3.3), asetettaisiin tälle henkilölle satunnaisesti joku näistä ennalta määritellyistä profiileista. Näin jokaiselle henkilölle alkaisi kertyä ajan kuluessa säännöllistä ostohistoriaa.

6.2 Kauppacontrollerin käyttöliittymä

Asiakasprofiilien lisäksi Kauppacontrollerin jatkokehitykseen suunniteltiin myös Kassajärjestelmien harjoituksien aikaiseen hallinnointiin käytettävää käyttöliittymää. Tässä käyttöliittymässä käyttäjän

olisi mahdollisuus nähdä jokainen rekisteröitynyt Kassajärjestelmä ja niiden tämänhetkinen tila. Käyttöliittymässä olisi mahdollista vähintäänkin sammuttaa ja käynnistää Kassajärjestelmiä sekä esimerkiksi muuttaa ostotapahtumien ajoittamisen parametreja.

Tämä ominaisuus ehdittiin opinnäytetyön aikarajojen puitteissa toteuttaa prototyyppitasolle asti, mutta jotta sitä voitaisiin varsinaisesti hyödyntää harjoituksessa, tulee sitä kuitenkin vielä kehittää pidemmälle. Itse käyttöliittymän lisäksi myös Kassajärjestelmien rekisteröinti prosessiin piti tehdä muutoksia. Nyt jokaisen rekisteröinnin yhteydessä yksittäinen Kassajärjestelmä saa kaupan tietojen lisäksi myös yksilöidyn numeron, jonka avulla Kauppacontroller voi erottaa saman kaupan Kassajärjestelmät toisistaan. Lisäksi jokaisen Kassajärjestelmän pitää myös ylläpitää omaa tilaansa esimerkiksi tietokannassa.



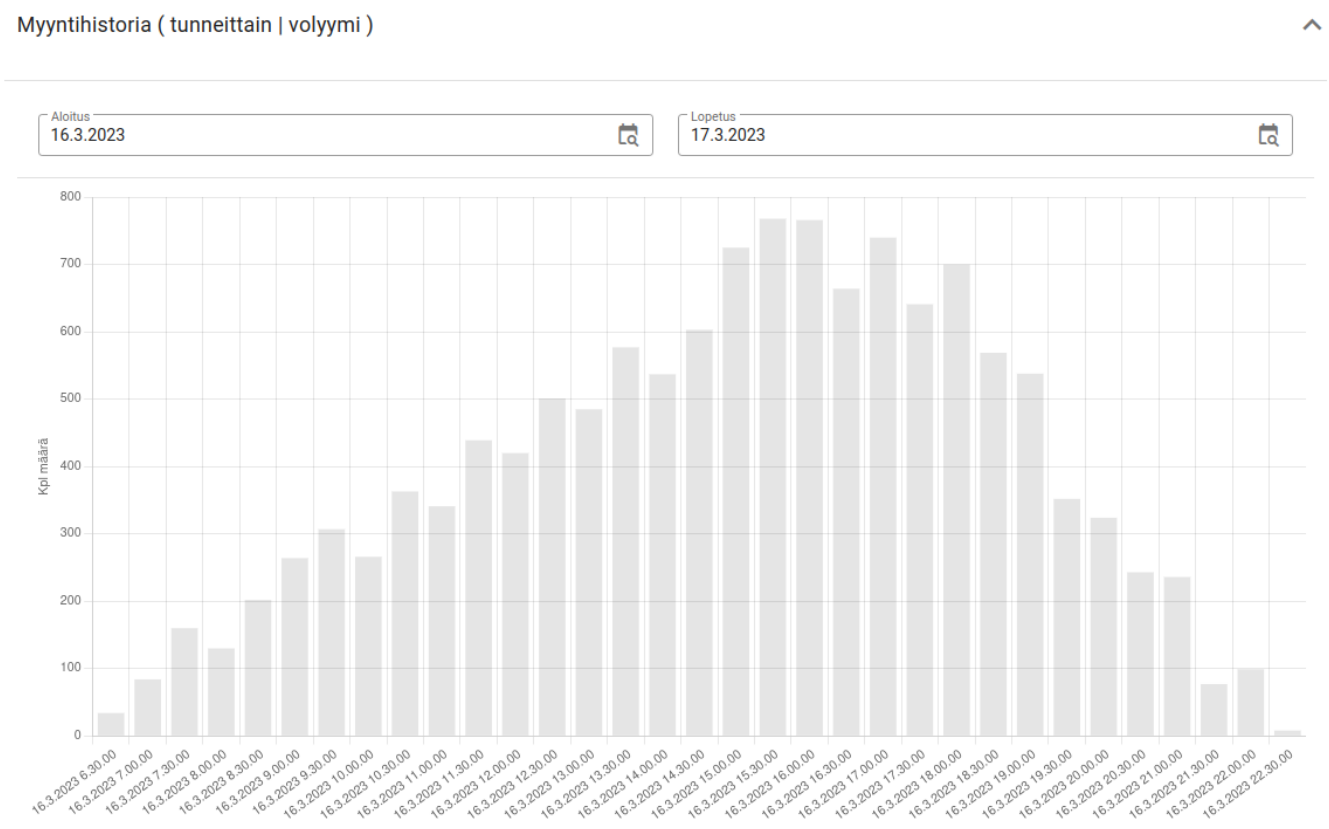
Kuvio 18. Kauppacontrollerin käyttöliittymän prototyyppi

7 Tulokset

Opinnäytetyön tuotoksena suunniteltiin sekä toteutettiin järjestelmäkokonaisuus, joka simuloi elintarvikekauppojen päivittäistä toimintaa ostotapahtumien muodossa osana Elintarvikeketjun

kyberturvallisuus -hankkeessa kehitettyä kyberharjoitusympäristöä. Järjestelmäkokonaisuus koostui kahdesta erillisestä järjestelmästä, jotka molemmat toteutettiin Docker-pohjaisina web-palveluina luvussa 3.2 esiteltyjä teknologioita hyödyntäen. Nämä kaksi järjestelmää olivat Kauppacontroller, joka toimi keskitettynä varastona ostostapahtumissa käytettäville tuotteille ja Kassajärjestelmä, jonka tehtävänä oli tapahtumien luominen ja ajoittaminen. Järjestelmäkokonaisuuden vaatimuksia sekä ominaisuuksia kuvattiin luvussa 3.1.

Järjestelmäkokonaisuuden keskeisin ominaisuus oli ostostapahtumien tuottaminen. Yksittäisen ostotapahtuman tuli sisältää realistisesti mallinnetut tuotteet sekä henkilötiedot ja näitä ostotapahtumia piti luoda automaattisesti päivän kuluessa. Luvussa 3.1.4 kerrottiin, kuinka jokaisesta ostotapahtumasta tulisi myös generoida kuitti, jota hyödynnettäisiin myöhemmin ERP-järjestelmässä visualisoinnissa. Tätä visualisointia varten ostotapahtumat pyrittiinkin jaksottamaan mahdollisimman realistisesti (ks. luku 4.3).



Kuvio 19. Kuvakaappaus ERP-järjestelmän käyttöliittymästä

Kuvio 19 on kuvakaappaus aikaisemmin mainitussa testausympäristössä olleen ERP-järjestelmän käyttöliittymän komponentista, joka visualisoi yksittäisen kaupan myyntihistoriaa rajatulla aikavälillä puolen tunnin välein. Kuten kuvakaappauksen ylälaidasta näkyy, aikaväliksi on asetettu yksi päivä. Tämän graafin piirtämiseen käytettävä data ovat Kassajärjestelmältä tulleet kuitit. Verrattaessa tätä graafia kuvioon 12, jossa oli esitetty kuinka ostostapahtumien tulisi vaihdella kellonajasta riippuen, voidaan todeta, että luvussa 4.3 esitellyllä toteutustavalla on päästy halutunlaiseen lopputulokseen tapahtumien ajoittamisen osalta. Selkeyden vuoksi mainittakoon vielä, että ERP-järjestelmän graafissa yksittäinen palkki ei edusta ostostapahtumien määrää, vaan niiden sisältämien tuotteiden yhteenlaskettuja kappalemääriä.

Järjestelmäkokonaisuuden sisältämien tietojen realistisuus saavutettiin onnistuneesti integroitumalla Tuotetieto-palveluun tuotetietojen ja Stork-palveluun henkilötietojen osalta. Maksukorttia lukuun ottamatta muut henkilötiedot jäivät näiden järjestelmien osalta piiloon, mutta esimerkiksi luvussa 6.1 esiteltyjen asiakasprofiilien implementaatiolle on jo niiden johdosta luotu hyvä pohja. Tuotetietojen vaikutus taas on hyvin nähtävillä Kassajärjestelmän käyttöliittymässä (ks. kuvio 15) sekä ERP-järjestelmässä (ks. liitteet 3 ja 4).

Vaatimuksien mukainen skaalautuvuus järjestelmille mahdollistettiin Dockerin ja luvussa 3.3 esiteltyjen ratkaisujen avulla. Näiden ratkaisujen toimivuus pystyttiin todentamaan EKK-hankkeen pilottiharjoituksessa, jossa järjestelmien määrä moninkertaistettiin verrattuna testausympäristöön.

8 Pohdinta

8.1 Työn tavoitteet

Opinnäytetyön ensisijainen tavoite oli tuottaa konkreettinen ratkaisu toimeksiantajan esittämään käytännön ongelmaan konstruktiivista tutkimusmenetelmää käyttäen. Luvussa 2.4.1 kerrottiin myös, kuinka konstruktiivisessa tutkimuksessa on oleellista, että työn tuotokseksi saadaan käytännössä hyödynnettävä rakenne, joka voidaan myös todeta toimivaksi. Edellisessä luvussa esiteltyjen tulosten, sekä luvussa 5.2 mainitun EKK-hankkeen pilottiharjoituksen perusteella voidaan tämä tavoite mieltää saavutetuksi.

Luvussa 2 esitettiin konstruktiivisen tutkimuksen tuloksien arviointiin kolme tasoista markkinatestiä. Samassa luvussa mainittiin myös, kuinka tälle työlle asetettiin tavoitteeksi tämän markkinatestin ensimmäinen taso, jonka läpäisy vaatisi konstruktion käyttöönoton kohdeorganisaation sisällä. Työn voidaankin todeta läpäisseen tämän tason, koska järjestelmäkokonaisuus otettiin JYVSECTECin sisäiseen käyttöön aiemmin mainitussa pilottiharjoituksessa.

Toimeksiantaja oli tyytyväinen työn lopputulokseen ja sen toteutustapaan kokonaisuutena. Erityisesti toteutetun järjestelmäkokonaisuuden suoriutuminen pilottiharjoituksessa sekä tunnistetut jatkokehitysideat olivat toimeksiantajan näkökulmasta arvokkaita saavutuksia.

8.2 Teknologiavalinnat

Luvussa 3.2 esiteltiin ja perusteltiin työn tekniseen toteutukseen valitut teknologiat. Teknologiavalintojen voidaan yleisesti ottaen todeta olleen onnistuneita, sillä järjestelmäkokonaisuus vaatimuksineen pystyttiin toteuttamaan niiden avulla. Näiden teknologioiden valintaan kuitenkin vaikutti paljon EKK-hankkeen kehityksessä aikaisemmin määritelty teknologiakokonaisuus. Opinnäytetyön teknisen toteutuksen aikarajojen puitteissa ei vaihtoehtoisia teknologioita myöskään ehditty testata ja vertailla kovinkaan syvällisesti.

Hapi.js:n valinta toteutuksen backend-ohjelmistokehykseksi pohjautui pitkälti aiemman kehitystyön kokemuksiin ja sen tuttuihin käytänteisiin. Kuitenkin tämän työn järjestelmien toteutuksessa iso osa sen tarjoamista eduista, joita esiteltiin osittain luvussa 3.2.3, jäivät hyödyntämättä. Koska kehitetyt järjestelmät olivat käytännössä täysin autonomisia eivätkä vaatineet minkäänlaista käyttäjien hallintaa, pysyivät esimerkiksi Hapin tarjoamat autentikointirajapinnat kokonaan käyttämättöminä. Toisaalta taas osa näistä ominaisuuksista aiheutti jopa ongelmia, kuten esimerkiksi luvussa 5.2 mainitut HTTP-pyyntöjen tietosisällön koon rajoitukset. Näin ollen onkin täysin mahdollista, että esimerkiksi Express olisi voinut olla parempi ratkaisu tämän työn toteutuksen kannalta, sillä se on ohjelmistokehyksenä huomattavasti minimaalisempi eikä olisi sisältänyt samanlailla tämän työn suhteen turhia ominaisuuksia. Express on minimaalisuutensa vuoksi myös kevyempi kuin Hapi, jonka ansiosta se olisi voinut täyttää palvelun skaalautuvuus vaatimuksen paremmin. Lisäksi Express on Hapia suositumpi ja kypsempi ohjelmistokehys, joka taas voitaisiin tulkita eduksi jatkokehitystä silmällä pitäen.

Erityisen onnistuneeksi teknologiavalinnaksi puolestaan voidaan nostaa Dockerin sekä Ansiblen yhdistelmä. Nämä kaksi teknologiaa mahdollistivat tehokkaasti työn kehityksen aikaisen jatkuvan käyttöönoton testausta varten sekä tarjosivat hyvät edellytykset myös pilottiharjoitusta edeltävälle käyttöönotolle. Docker eliminoi täysin huolen lopullisen käyttöympäristön yhteensopivuudesta järjestelmien kanssa, jonka ansiosta varsinainen kehitystyö pystyttiin keskittämään täysin haluttujen ominaisuuksien rakentamiseen järjestelmien sisällä. Ansible puolestaan säästi paljon arvokasta kehitysaikaa automatisoimalla suuren osan käyttöönoton eri vaiheista järjestelmäkohtaisten playbookkien avulla.

8.3 Yhteenveto

Kokonaisuutena opinnäytetyön voidaan sanoa onnistuneen tavoitteiden mukaisesti. Konstruktiivisesta tutkimusotetta käyttäen saatiin luotua järjestelmäkokonaisuus, joka omalta osaltaan suoriutui odotuksien mukaisesti Elintarvikeketjun kyberturvallisuus -hankkeen pilottiharjoituksessa ja jota JYVSECTEC voi tulevaisuudessa hyödyntää omassa kyberharjoitustoiminnassaan. Järjestelmäkokonaisuudelle onnistuttiin myös tunnistamaan mahdollisia kehitysideoita, joita voidaan JYVSECTECin toimesta kehittää tarvittaessa.

Lähteet

Anicas M. 2022. An Introduction to HAProxy and Load Balancing Concepts. Artikkeli DigitalOcean verkkosivustolla. Viitattu 11.5.2023. https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

About Node.js. N.d. Node.js-verkkosivusto. Viitattu 5.3.2023. https://nodejs.org/en/about/

Ansible playbooks. N.d. Dokumentaatio Ansible-verkkosivustolla. Viitattu 22.4.2023. https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html

Awati, R. 2023. Object-relational mapping (ORM). Artikkeli Theserverside-verkkosivustolla. Viitattu 1.5.2023. https://www.theserverside.com/definition/object-relational-mapping-ORM

Cromwell, V. 2016. Evan You. Artikkeli Vue.js:n kehittäjästä. Viitattu 7.4.2023. https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/

Description. N.d. HAProxy-verkkosivusto. Viitattu 9.4.2023. https://www.haproxy.org/#desc

Docker Compose overview. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 23.4.2023. https://docs.docker.com/compose/

Docker overview. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 7.4.2023. https://docs.docker.com/get-started/overview/

Dockerfile reference. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 22.4.2023. https://docs.docker.com/engine/reference/builder/

Domantas, G. 2023. What is SSH: Understanding Encryption, Ports and Connection. Artikkeli Hostinger-verkkosivustolla. Viitattu 4.5.2023. https://www.hostinger.com/tutorials/ssh-tutorial-how-does-ssh-work

Elintarvikeketjun kyberturvallisuus, Projektin kuvaus. N.d. JAMKin verkkosivusto. Viitattu 4.3.2023. https://www.jamk.fi/fi/tutkimus-ja-kehitys/tki-projektit/elintarvikeketjun-kyberturvallisuus

Ellingwood, J. N.d. Artikkeli Prisma.io verkkosivustolla. Viitattu 4.5.2023. https://www.prisma.io/dataguide/types/relational/what-is-an-orm

Express. N.d. ExpressJS-verkkosivusto. Viitattu 22.5.2023. https://expressjs.com/

Gillis, A. 2021. UUID (Universal Unique Identifier). Artikkeli Techtarget-verkkosivustolla. Viitattu 28.4.2023. https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier

Global Cybersecurity Outlook 2023. 2023. World Economic Forumin verkkosivusto. Viitattu 4.3.2023. https://initiatives.weforum.org/global-cyber-outlook/home.

How Ansible works. N.d. Ansible-verkkosivusto. Viitattu 22.4.2023. https://www.ansible.com/overview/how-ansible-works

HTTP. N.d. Mozilla-verkkosivusto. Viitattu 4.5.2023. https://developer.mozilla.org/en-US/docs/Web/HTTP

Introduction. N.d. Vuejs-verkkosivusto. Viitattu 7.4.2023. https://vuejs.org/guide/introduction.html

Introducing JSON. N.d. JSON-verkkosivusto. Viitattu 4.5.2023. https://www.json.org/json-en.html

Introduction to Node.js. N.d. Node.js-verkkosivusto. Viitattu 5.3.2023. https://nodejs.dev/en/learn/

Introduction to the DOM. N.d. Mozilla-verkkosivusto. Viitattu 14.5.2023. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

JYVSECTEC CYBER RANGE – RGCE and solutions. 2018. JYVSECTECin verkkosivusto. Viitattu 4.3.2023. https://jyvsectec.fi/cyber-range/.

JYVSECTEC terveydenhuollon kyberharjoittelun mahdollistajana. N.d. JYVSECTECin verkkosivusto. Viitattu 4.3.2023. https://jyvsectec.fi/fin/terveydenhuolto/terveydenhuollon-harjoitukset/

Lukka, K. 2001. Konstruktiinen tutkimusote. Artikkeli konstruktiivisesta tutkimuksesta Metodix-verkkosivustolla. Viitattu 4.3.2023. https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Lutkevich B. 2021. Domain name system (DNS). Artikkeli Techtarget-verkkosivustolla. Viitattu 4.5.2023. https://www.techtarget.com/searchnetworking/definition/domain-name-system

Model Basics. N.d. Dokumentaatio Sequelize-verkkosivustolla. Viitattu 1.5.2023. https://sequelize.org/docs/v6/core-concepts/model-basics/

Ojasalo, K., Moilanen, T. & Ritalahti, J. 2015. Kehittämistyön menetelmät. Uudenlaista osaamista liiketoimintaan. 4. uud. p. Helsinki: Sanoma Pro Oy. Viitattu 4.3.2023. https://janet.finna.fi/Record/jamk.993548674806251, Ellibs.

Pekarsky, M. 2020. Does your web app need a front-end framework? Blogikirjoitus Stackoverflow-verkkosivustolla. Viitattu 7.4.2023. https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/

Redis. N.d. AWS-verkkosivusto. Viitattu 5.5.2023. https://aws.amazon.com/redis/

Relational vs. Non-relational Databases. N.d. MongoDB-verkkosivusto. Viitattu 7.4.2023. https://www.mongodb.com/compare/relational-vs-non-relational-databases

Sequelize v6. N.d. Dokumentaatio Sequelize-verkkosivustolla. Viitattu 1.5.2023. https://sequelize.org/docs/v6/

Specializing in cyber security expertise. N.d. JYVSECTECin verkkosivusto. Viitattu 4.3.2023. https://jyvsectec.fi/about/.

The Node.js Event Loop. N.d. Node.js-verkkosivusto. Viitattu 5.3.2023. https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

TypeScript – Overview. N.d. Tutorialspoint-verkkosivusto. Viitattu 4.3.2023. https://www.tutorialspoint.com/typescript/typescript_overview.htm.

Use containers to Build, Share and Run your applications. N.d. Dokumentaatio Docker-verkkosivustolla. Viitattu 7.4.2023. https://www.docker.com/resources/what-container/

What is an API. N.d. IBM-verkkosivusto. Viitattu 4.5.2023. https://www.ibm.com/topics/api

What is CI/CD. 2022. RedHat-verkkosivusto. Viitattu 4.5.2023. https://www.redhat.com/en/topics/devops/what-is-ci-cd

What is ERP. N.d. Oracle-verkkosivusto. Viitattu 9.4.2023. https://www.oracle.com/erp/what-is-erp/

What is NoSQL. N.d. MongoDB-verkkosivusto. Viitattu 7.4.2023. https://www.mongodb.com/nosql-explained

What is PostgreSQL. N.d. IBM-verkkosivusto. Viitattu 7.4.2023. https://www.ibm.com/topics/postgresql

What is REST API. N.d. RedHat-verkkosivusto. Viitattu 9.5.2023. https://www.redhat.com/en/topics/api/what-is-a-rest-api

What is SQLite. N.d. Codeacademy-verkkosivusto. Viitattu 23.4.2023. https://www.codecademy.com/article/what-is-sqlite

What is SQL (Structured Query Language). N.d. AWS-verkkosivusto. Viitattu 4.5.2023. https://aws.amazon.com/what-is/sql/

What is Transmission Control Protocol TCP/IP. N.d. Fortinet-verkkosivusto. Viitattu 4.5.2023. https://www.fortinet.com/resources/cyberglossary/tcp-ip

What is Vuetify. N.d. Vuetifyjs-verkkosivusto. Viitattu 7.4.2023. https://vuetifyjs.com/en/introduction/why-vuetify/#what-is-vuetify3f

What is YAML. 2023. RedHat-verkkosivusto. Viitattu 4.5.2023. https://www.redhat.com/en/topics/automation/what-is-yaml

Liitteet

Liite 1. EKK-harjoitusympäristön yksittäisen tuotteen tietosisältö


```
{
    "label": {
        "ean": "6408180733260",
        "name": "Vaasan Ruispalat Ohut Herkku 195g 6 kpl täysjyväruisleipä",
        "lotId ": "L077007126630 ",
        "price ": {
            "vat": {
                "rate": 14,
                "class": 3
            },
            "value": 1.25,
            "currency": "€",
            "priceUnit ": "KPL "
        },
        "origin ": "FI ",
        "quantity ": {···
        },
        "brandName ": "Vaasan ",
        "shelfLife ": 21,
        "properties ": {
            "frozen ": false,
            "organic ": null,
            "eNumbers ": null,
            "vitamins ": null,
            "allergens ": [···
            ],
            "nutrients ": [···
            ],
            "preservation ": null,
            "containsAlcohol ": false,
            "isAgeLimitedByAlcohol ": false
        },
        "description ": "Ohut täysjyväruisleipä sitkeään rukiin himoon!Leivottu 100 % kotimaisesta viljasta.Kuitua 13 % .",
        "manufacturer ": "wheat ",
        "filenameImage ": "6408180733260. jpg ",
        "filenameThumb ": "6408180733260 - thumb.jpg ",
        "hierarchyPath ": [···
        ],
        "contactDetails ": {···
        },
        "expirationDate ": "2023 - 05 - 18 T20: 32: 40.485 Z ",
        "productionDate ": "2023 - 04 - 27 T20: 32: 40.485 Z ",
        "refBatchAmount ": 0.195,
        "sequenceNumber ": "228 / 260 "
    },
    "content ": {
        "properties ": {
            "nutrients ": [
                {
                    "name ": "Energiaa ",
                    "value ": "1213 kJ / 290 kcal "
                }, {
                    "name ": "Rasvaa ",
                    "value ": "1,7 g "
                }, {
                    "name ": "Rasvaa,josta tyydyttyneitä rasvoja ",
                    "value ": "0,4 g "
                }, {
                    "name ": "Hiilihydraattia ",
                    "value ": "53 g "
                }, {
                    "name ": "Hiilihydraattia,josta sokereita ",
                    "value ": "1,1 g "
                }, {
                    "name ": "Proteiinia ",
                    "value ": "9,3 g "
                }, {
                    "name ": "Suolaa ",
                    "value ": "1,1 g "
                }
            ]
        },
        "batchAmount ": 0.195
    }
}
```

Liite 2. Storkissa generoidun henkilön maksukortit

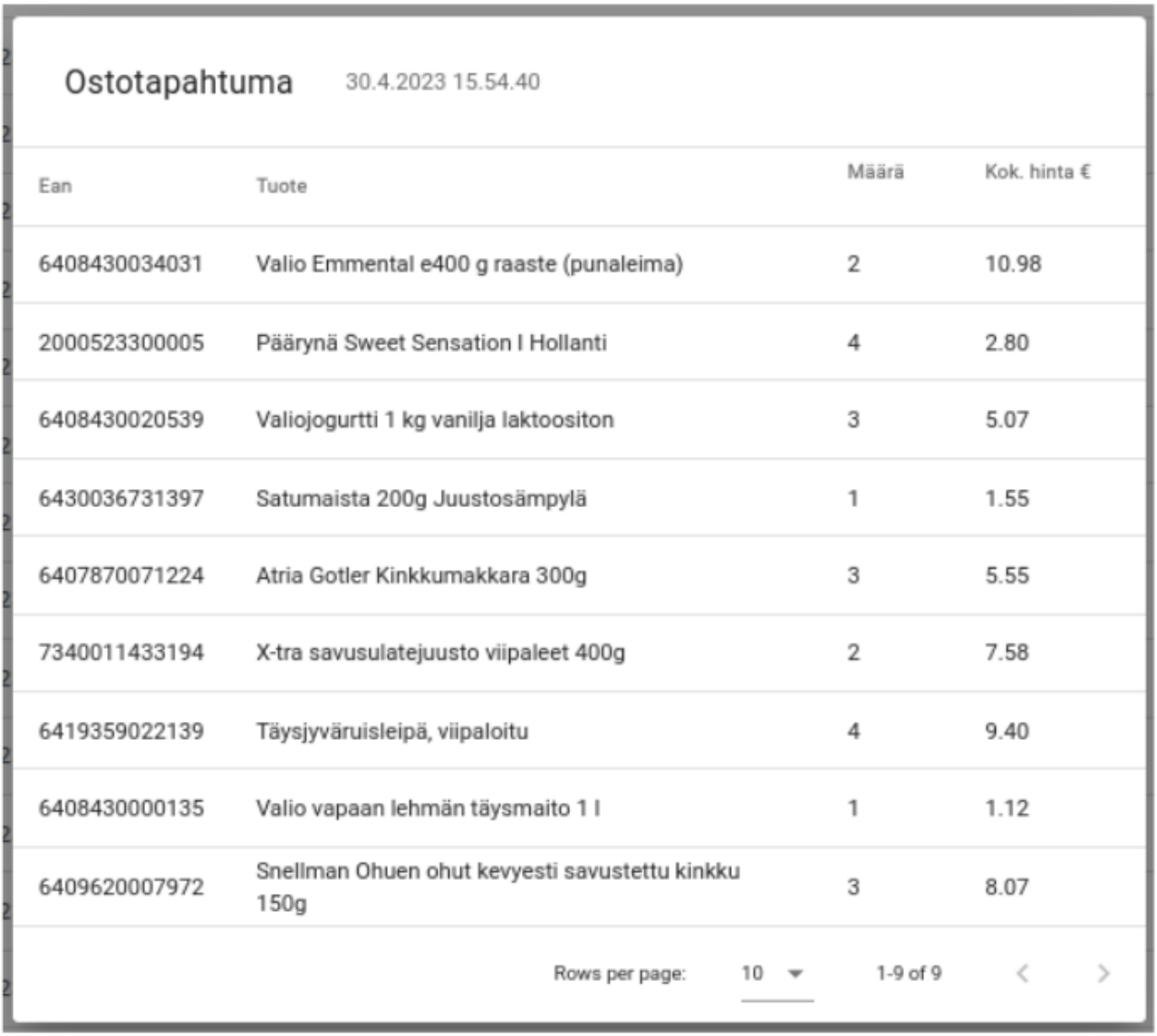
Payment cards +

Card number	Network	Type	Exp date	CVV
43187082895746660	VISA	CREDIT	08/2027	321
43187167158459356	VISA	DEBIT	05/2024	601

Liite 3. Kuvion 14 Ostotapahtumat ERP-järjestelmässä

30.4.2023 15.47.28	41bbee98-59b2-40ba-9732-70ba2ab5b58f	41.78	
30.4.2023 15.46.52	613c89be-4c1c-412e-8573-09e9a0d42e86	31.80	
30.4.2023 15.46.16	c86fb8cb-540b-4adf-9167-c559ed5a35f9	7.55	
30.4.2023 15.45.40	9f1f6fb4-1b72-4ab3-a265-8273c4380f2b	27.66	
30.4.2023 15.45.04	fe2f5501-0f45-48c0-88e7-840bf8e65c0b	64.14	
30.4.2023 15.44.28	04d65dab-af38-424d-aefe-72edba783a27	39.37	
30.4.2023 15.43.52	4105a1e9-6845-4325-95fb-dee482b74983	45.46	
30.4.2023 15.43.16	a35a2b08-30fa-4083-bfa5-d4e779d5e8d7	59.16	
30.4.2023 15.42.40	4fde534f-4e01-414c-b293-504468b83e2f	52.87	
30.4.2023 15.42.04	55fa7d10-8601-4c3a-84c8-c97686d71c3c	35.24	
30.4.2023 15.41.28	996261d6-e8db-4c73-a269-2bdeb9f1ecd9	31.73	
30.4.2023 15.40.52	b320a291-becc-499e-bdd6-ae1039a4a191	2.98	
30.4.2023 15.40.16	a293beb6-d24c-4033-ba75-d63d6bc8e115	53.74	
30.4.2023 15.39.40	a67eb2ce-62af-49ae-9332-edaed964bf4a	5.25	

Liite 4. Kuvion 15 Ostostapahtuma ERP-järjestelmässä



Ostotapahtuma 30.4.2023 15.54.40

Ean	Tuote	Määrä	Kok. hinta €
6408430034031	Valio Emmental e400 g raaste (punaleima)	2	10.98
2000523300005	Päärynä Sweet Sensation I Hollanti	4	2.80
6408430020539	Valiojogurtti 1 kg vanilja laktoositon	3	5.07
6430036731397	Satumaista 200g Juustosämpylä	1	1.55
6407870071224	Atria Gotler Kinkkumakkara 300g	3	5.55
7340011433194	X-tra savusulatejuusto viipaleet 400g	2	7.58
6419359022139	Täysjyväruisleipä, viipaloitu	4	9.40
6408430000135	Valio vapaan lehmän täysmaito 1 l	1	1.12
6409620007972	Snellman Ohuen ohut kevyesti savustettu kinkku 150g	3	8.07

Rows per page: 10 1-9 of 9