Android-ohjelman suunnittelu ja määrittely sekä toteutus käyttäen konstruktiivista suunnittelua

Oulun yliopisto
Tieto- ja sähkötekniikan tiedekunta
/M3S
Gradu
Teemu Törrö
01.03.2021

Tiivistelmä

Työn tavoitteena oli tutkia, sopiiko konstruktiivinen tutkimusmenetelmä Android-ohjelma suunnitteluun ja määrittelyyn sekä toteutukseen. Motiivina työn tekemiseen oli löytää mahdollinen toteutuskeino uudelle järjestelmälle. Olemassa olevalle järjestelmälle tarvittiin erillinen korvaava järjestelmä, jotta mobiililaitteen käyttäjällä pääsy informaatioon helpottuisi.

Ensiksi käytiin läpi Android-arkkitehtuuri, tietoturvakäytännöt ja SELinuxin toiminta Android-käyttöjärjestelmässä. Tämän jälkeen tutustuttiin Androidin tietoturvauhkiin ja niiden tutkimukseen. Seuraavaksi esiteltiin mahdolliset Android-sovelluksien ja palvelimen sekä teknologian arkkitehtuuri. Vaatimusmäärittelyssä tutustuttiin perinteisen vesiputousmallin ja ketterien menetelmien vaatimusmäärittelyprosessiin. Seuraavaksi käytiin läpi tutkimusmenetelmät eli tässä tapauksessa konstruktiivinen tutkimus, jonka pohjalta luotiin kuvaus konstruktiivisen tutkimuksen prosessin vaiheista. Sitten esiteltiin ongelmakehys kokonaisuudessaan, ensin demonstroitiin ongelma ja olemassa oleva järjestelmä, mistä päästiin tulevan järjestelmän esittelyyn. Konstruktiivisen tutkimuksen määrittelyvaiheen avuksi sovellettiin ketterien menetelmien käyttäjätapauksia. Lisäksi esiteltiin käyttäjätapauksen kirjoittamista indeksikortille ja määriteltiin esimerkin avulla vaatimukset tulevalle Android-ohjelmalle sekä siihen kuuluvalle palvelin-ohjelmalle ja teknologialle. Itse konstruktio toteutettiin kolmena mahdollisena toteutuksena Android-sovelluksesta. Tämän lisäksi toteutettiin palvelimen ja teknologian konstruktiot. Seuraavaksi konstruktiot evaluoitiin sekä analyyttisesti että testauksen avulla. Lopuksi koottiin tulokset yhteen. Tutkimuksen perusteella konstruktiivisen tutkimuksen iteratiivinen prosessi ja ketterien menetelmien käyttäjäkertomukset tukevat Android-ohjelman suunnittelua ja määrittelyä sekä toteutusta. Konstruktiivista tutkimusta hyödyntäen tässä työssä toteutettiin kolme erilaista toteutusta Android-sovelluksille.

Avainsanat
Android, Tietoturva, Vaatimusmäärittely, Palvelin, Konstruktiivinen tutkimus, Käyttäjäkertomus

Ohjaaja
Dosentti Raija Halonen

Esipuhe

Tämä maisterityö on tehty Oulun yliopiston tieto- ja sähkötekniikan tiedekunnan tietojenkäsittelytieteiden laitokselle. Työn ohjaajana ja valvojana on toiminut Dosentti Raija Halonen. Esitän hänelle kiitokset saamistani neuvoista ja työn eteenpäin viemisestä. Lisäksi esitän kiitokset Pertti Karhapäälle rakentavista kommenteista ja neuvoista.

Kiitokset kuuluvat myös perheelleni ja puolisolleni Sannalle tuesta ja kannustuksesta koko opiskelujeni ajan. Kiitokset myös tyttärilleni Maijulle, Maisalle ja Saanalle.

Teemu Törrö

Oulu, maaliskuu, 01, 2021

Lyhenteet

HAL	Hardware Abstraction Layer
ART	Android Runtime
API	Application Programming Interface
PID	Process ID
ICC	Inter Component Communication
UID	User ID
MAC	Mandatory access control
SELinux	Security-Enhanced Linux
YAFFS	Yet Another Flash File System
PHA	Potential Harmful Applications
DDOS	Distributed Denial of Service
HTML	Hypertext Markup Language
Iframe	inline frame (HTML element)
(C&C)	Command and Control
JSON	JavaScript Object Notation
PWA	Progressive Web Application
URL	Uniform Resource Locator
CSS	Cascading Style Sheets
BSD	Berkley Software Distribution
SSL	Secure Socket Layer
RSS	Realtime Simple syndication
XML	Extensible Markup Language
CRON	Command Run On (Unix scheduler)
ATDD	Acceptance test-driven development
VAPID	Voluntary Application server Identification

Sisällysluettelo

Tiivistelmä 2
Esipuhe 3
Lyhenteet 4
Sisällysluettelo 5
1. Johdanto 7
2. Aikaisempi tutkimus 9
2.1 Android 9
2.1.1 Android arkkitehtuuri 9
2.1.2 SELinux 11
2.1.3 Android-tietoturvauhat ja tutkimus 11
2.2 Vaatimusmäärittely 15
2.3 Järjestelmän arkkitehtuuri 20
2.3.1 Android-sovelluksen arkkitehtuuri 20
2.3.2 Palvelinsovelluksen arkkitehtuuri 23
2.3.3 Teknologian arkkitehtuuri 27
3. Tutkimusmenetelmät 29
4. Ongelmakehys 33
4.1 Tutkimuskysymyksien esittely 33
4.2 Ongelman esittely 33
4.3 Olemassa olevan järjestelmän esittely 34
4.4 Tulevan järjestelmän esittely 34
5. Konstruktion vaatimusmäärittely 35
5.1 Android-sovellus 35
5.2 Palvelinsovellus 37
5.3 Teknologiamäärittely 38
6. Konstruktio 40
6.1 Android-ohjelman konstruktio 40
6.2 Android-ohjelman vaihtoehtoinen konstruktio 42
6.3 Progressive web application (PWA) konstruktio 43
6.4 Palvelinohjelman konstruktio 47
6.4.1 Palvelinohjelman tietokannan päivittäjä 47
6.4.2 Palvelinohjelman Flask-websovellus 49
6.5 Teknologian konstruktio 56
7. Evaluointi 63
7.1 Analyyttinen 63
7.1.1 Android-sovelluksien toteutuksen analysointi 63
7.1.2 Palvelin-sovelluksen toteutuksen analysointi. 64
7.2 Testaus 65
7.2.1 Yksikkötestaus 65
7.2.2 Integraatiotestaus 67
7.2.3 Järjestelmätestaus 68
7.2.4 Hyväksyntätestaus 70
8. Pohdinta 75
9. Yhteenveto 78
Lähteet 79
Liite A. AndroidManifest.xml 84
Liite B. MainActivity.java 85
Liite C. activity_main.xml 87
Liite D. URL-lista 88

1. Johdanto

Työn tavoitteena oli tutkia, sopiiko konstruktiivinen tutkimusmenetelmä Android-ohjelmasuunnitteluun, sen määrittelyyn sekä toteutukseen. Tämän esittämäni tapauksen avulla saadaan olemassa olevan internetsivun tai RSS-syötteen sisältö tietyin reunaehdoin mobiilisovelluksella käytettävään muotoon. Konstruktoidun järjestelmän tulisi toimia mahdollisimman automaattisesti. Järjestelmään kuuluu palvelinsovellus, joka muuttaa olemassa olevan sivun sisällön sellaiseksi, että mobiiliohjelman käyttäjä voi saada haluamansa informaation mahdollisimman helposti omalle päätelaitteelleen sopivaan esitysmuotoon.

Motivaatio järjestelmän kehittämiseen oli se, että nykyiselle järjestelmälle tai tarkemmin internetsivulle oli vaikeata päästä ensimmäistä kertaa, varsinkin mobiililaitteella. Tämän vuoksi suunniteltiin uusi erillinen järjestelmä, jonka avulla käyttäjä sai haluamansa informaation nopeasti ja vaivattomasti silloin, kun hän sitä tarvitsee.

Käytettäväksi järjestelmäksi valittiin Android, koska sen markkinaosuus lähentelee jo 82 % (Gartner Worldwide Sales, 2017). Tästä johtuen mobiilisovellus määriteltiin ensiksi kyseiselle käyttöjärjestelmälle. Toisekseen yritettiin löytää mahdollisimman yleispätevä ratkaisu, jotta muillakin mobiileilla laitteilla sama informaatio olisi saatavilla yhtä helposti.

Palvelimen ohjelmistolle määriteltiin toimintaympäristö. Siitä pyrittiin tekemään kevyt ja tuottamaan ainoastaan kyseisen informaation tarvitsemat palvelut. Kokonaisuuden, ainakin kehitysvaiheessa, tulisi toimia Raspberry Pi[1]- pohjaisella yhden piirilevyn tietokoneella, mutta sen tulisi olla siirrettävissä tarpeen vaatiessa tehokkaammalle palvelinalustalle.

Tutkimuksessa tarkasteltiin Android-käyttöjärjestelmän arkkitehtuuria ja selvitettiin millaisia rajoituksia sekä mahdollisuuksia se tarjoaisi kehitettävän ohjelman suunnittelulle ja toteutukselle. Samalla käytiin läpi mahdolliset tietoturvauhat Android-alustalla toimivalle sovellukselle. Kirjallisuuskatsauksen perusteella ja konstruktiivisen tutkimuksen avulla selvitettiin, miten kyseinen järjestelmä voitaisiin määritellä. Hevner, March, Park ja Ram (2004) esittivät artikkelissaan seitsenkohtaisen ohjenuoran konstruktiiviselle tutkimukselle ja Kasanen, Lukka ja Siitonen (1993) kuusikohtaisen näkemyksensä siitä, mitä tulisi sisällyttää konstruktiiviseen tutkimukseen. Konstruktiivisen tutkimuksen kirjallisuuden perusteella kehitettiin malli, jota apuna käyttäen ongelmakehys ja vaatimusmäärittely tehtiin sekä artefaktin konstruktio.

Tämä kirjallinen työ on jatkoa kandidaattityöhöni (Törrö, 2017), jossa analysoin konstruktiivisen tutkimuksen avulla ongelmakehystä ja vaatimusmäärittelyä. Siinä kehitin tulevan järjestelmän vaatimusmäärittelyyn yksinkertainen mallin, jonka esittelin esimerkin avulla. Tässä työssä jatkettiin konstruktion ja artefaktin toteutus kandityöni pohjalta. Tässä työssä kehitettiin kolme mahdollista Android-ohjelmaa ja sen tarvitsemat palvelut toteutettiin palvelimen osalta sekä saatettiin käytännössä toimimaan itsenäisenä järjestelmänä. Konstruktiossa kehitetty artefakti evaluoitiin analyyttisesti ja testaamalla konstruktion toteutus yksikkö-, integraatio-, järjestelmä- ja hyväksyntätestausta apuna käyttäen. Pohdinnassa tarkasteltiin tämän työn ja aikaisemman tutkimuksen löydöksiä keskenään ja yhteenvedosta löytyy lopputulos.

Työn rakenne on seuraavan kaltainen, luvun 2 kirjallisuuskatsauksessa käydään läpi Android arkkitehtuuri, SELinux ja Androidin tietoturvauhat sekä niihin liittyvä tutkimus.

Lisäksi esiteltiin suunnitellun järjestelmän arkkitehtuuri toteutuksen osalta. Luvussa 3 esitellään konstruktiivinen tutkimus ja niiden pohjalta määritelty prosessikehys. Luvussa 4 esitellään ongelmakehys. Luvussa 5 tarkastellaan vaatimusmäärittelyn kirjallisuuden pohjalta konstruktiivista tutkimusta tukeva malli, jota apuna käyttäen määritellään vaatimukset kehitettävälle järjestelmälle. Luvussa 6 käydään konstruktio niin Android-sovelluksien kuin sen palvelimen ja toteutuksen mahdollistavan teknologian osalta. Luvussa 7 evaluoidaan kehitetty järjestelmä analyyttisesti ja testauksen perusteella. Luvussa 8 on pohdinta ja kirjallisen työn yhteenveto tutkimuksesta on luvussa 9. Työn lopusta löytyy käytetyt lähteet ja sekä liitteenä Webview-sovelluksen toteutus Android päätelaitteelle.

2. Aikaisempi tutkimus

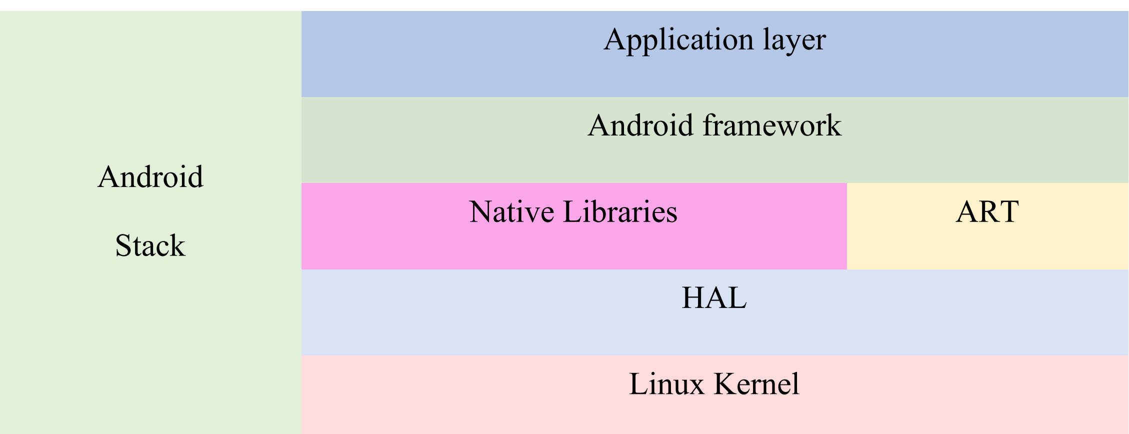
Tässä luvussa käydään läpi aikaisemmin tehtyä tieteellistä tutkimusta Androidista, sen arkkitehtuurista ja tietoturvasta, Android-ohjelmien käyttämistä luparakenteista sekä järjestelmän käyttäjään liittyvästä yksityisyyttä rajoittavista tekijöistä. Vaatimusmäärittelyssä käydään läpi aikaisempi tutkimus vesiputousmallista ketteriin menetelmiin, joitten perusteella esitellään ketterissä menetelmissä käytettävät käyttäjäkertomukset ja indeksikortti. Tämän jälkeen käydään läpi järjestelmä arkkitehtuuria, jonka pohjalta voidaan mahdollisesti kehittää tuleva järjestelmä niin Android-sovelluksen kuin sen palvelinohjelmiston ja sen teknologian osalta.

2.1 Android

Tässä luvussa esitellään ensiksi aikaisempi tutkimus Android arkkitehtuurista. Käydään läpi sen rakenne sekä esitellään SELinux. Lisäksi tarkastellaan Android-tietoturvauhat ja tutkimus.

2.1.1 Android arkkitehtuuri

Androidin arkkitehtuuri (Platform architecture, 2017) eli Android pino (Android stack) koostuu alimmalta tasolta ylöspäin seuraavasti. Ensimmäisenä on Linux kerneli, kuvassa 1 alimmaisena, jossa sijaitsevat suoritin ja muistin hallinta, laiteajurit sekä tiedostojärjestelmä. Toisena kernelin yläpuolella on HAL (hardware abstraction layer), jossa sijaitsevat kirjastot, joiden avulla laitteen sensorit ja lisälaitteet saadaan käyttöön ylemmän tason Java ohjelmointirajapinnalle. (Platform architecture, 2017.)



Kuva 1. Android Pino (Platform architecture, 2017).

Kuvassa 1 Android Runtime koostuu ART virtuaalikoneesta, joka sisältää ARTin, jos käytössä on Android versio 5.0 (API level 21), tai Dalvikin virtuaalikoneen ennen versiota 5.0 (Art and Dalvik, 2017). Jokainen sovellus saa oman virtuaalikoneensa, jossa prosessi luodaan ja ne yksilöidään omalla prosessitunnuksella PID (process id). Samalla tasolla löytyvät myös natiivit C/C++ kirjastot (Native Libraries), joita tarvitaan ART:in ja HAL:lin suorittamisessa. Seuraavalla tasolla (Android Framework) on Java ohjelmointirajapinta (API), joka on alustana kehitettäville Android-ohjelmille. API:n avulla ohjelmistokehittäjä voi käyttää Android-järjestelmän resursseja hyväkseen ohjelmissaan. Ohjelmatasolla (Application layer) on järjestelmän perusohjelmat, jotka

tarjoavat valikoiman valmiina sovelluksiin, kuten sähköposti-, numeronvalitsin-, kamera- ja kalenteriohjelmat. (Platform architecture, 2017.)

Android-sovellus koostuu komponenteista. Enck, Ongtang ja McDaniel (2009) esittelivät Android-sovelluksen komponentit, joita ovat aktiviteetti (activity), palvelu (service), sisällön tarjoaja (content provider) ja viestien välittäjä (broadcaster receiver). Aktiviteetti on ohjelman käyttöliittymä. Jokaisella sovelluksen näkymällä on oma aktiviteettinsa ja niitä voi olla käynnissä yksi kerrallaan. Palvelukomponentti tarjoaa esimerkiksi aktiviteetille keinon suorittaa jokin tehtävä taustalla, kun itse aktiviteetti ei ole aktiivinen. Sisällöntarjoaja on tiedon tallentava ja jakava komponentti. Viestinvälittäjäkomponentti toimii postilaatikon tavoin ja välittää viestejä ohjelmien välillä. (Burns, 2008; Enck ja muut, 2009.)

Komponenttien interaktio tapahtuu intenttien avulla (Burns, 2008; Enck ja muut, 2009). Se on abstrakti kuvaus operaatiosta, joka tullaan suorittamaan. Käytännössä intentti sisältää viestiobjektin, jossa on vastaanottavan komponentin nimi sekä viesti sille. Yksi tärkeimmistä intentin avulla tapahtuvista operaatioista on komponentin käynnistäminen (Intent, 2017). Toinen intentin avulla tapahtuva operaatio on intentin lähetys (intent broadcast). Kolmas operaatio on sisällöntarjoajaan käsiksipääsy (accessing content providers) ja neljäs komponenttien sidonta palveluihin (component bindings to service). (Burns, 2008; Bugiel, Davi, Dmitrienko, Fischer ja Sadeghi, 2011.) Komponenttien välistä kommunikointia kutsutaan yleisnimellä toiminnaksi (action) (Burns, 2008). Intenttien välitystä voi olla kahta erilaista. Ensimmäisellä tavalla kutsuttava komponentti voidaan määritellä ohjelman nimiavaruuden avulla. Toinen tapa on epäsuora kutsuminen, jossa Android-käyttöjärjestelmä valitsee parhaan mahdollisen komponentin suorittamaan kutsutun toiminnan asennettujen ohjelmien ja käyttäjän valintoihin perustuen. (Enck ja muut, 2009.) Epäsuoraa kutsua kutsutaan toimintamerkkijonoksi (action string). Siinä määritellään, millainen komponentti tarvitaan. Toisaalta intenttien kutsua voidaan suodattaa vastaanottavassa komponentissa. Tätä kokonaisuutta voitaneen kutsua sisäiseksi komponenttien kommunikaatioksi ICC (inter component communication). (Enck ja muut, 2009; Octeau ja muut, 2013.)

Bugiel ja kumppanit (2011) esittelivät Androidin tietoturvamekanismit seuraavasti. Ohjelmat ovat omissa hiekkalaatikoissaan ja toimivat siinä rajatussa ympäristössään (Bugiel ja muut, 2011; Enck ja muut, 2009). Tästä syystä jokainen sovellus on eroteltuna yksilöityinä käyttäjäidentiteetillä (UID), jota Linux-kerneli valvoo. Toinen tietoturvamekanismi on ohjelman allekirjoitus. Jokainen kehittäjä allekirjoittaa sovelluksensa ja näin ollen saman allekirjoituksen omaavat sovellukset voivat toimia samassa hiekkalaatikossa. Kolmas mekanismi on Android-lupakehys. Tähän kuuluu valvonta (reference monitor), joka tuottaa pakotetun oikeuksien tarkistuksen MAC (Mandatory access control), jokaiselle komponenttien ICC ja API-kutsulle. Nämä luvat jokainen ohjelma saa käyttöönsä silloin, kun käyttäjä asentaa ohjelman Android-laitteellensa, ja luvan saatuaan jokainen ohjelman komponentti saa samat oikeudet. (Bugiel ja muut, 2011; Enck ja muut, 2009.) Android-luvat (permissions) on luokiteltu neljään eri tasoon normaali, vaarallinen, allekirjoitettu ja allekirjoitettu/järjestelmä (Felt, Chin, Hanna, Song ja Wagner, 2011; Permissions groups, 2017). Feltin ja kumppaneitten (2011) mukaan normaali lupa suojaa API-kutsuja, jotka voivat olla ärsyttäviä käyttäjälle, mutta eivät aiheuta harmia hänelle. Vaaralliset luvat suojaavat harmillisilta API-kutsuilta, kuten rahan käytöltä ja yksityisyyttä koskevien tietojen uteluilta. Allekirjoitettu lupa suojaa vaarallisien oikeuksien käytön, esimerkiksi pakettien asentamisen. Ne ovat yleensä Android-laitteen valmistajan allekirjoittamia lupia. Jos tavallinen Android-ohjelmankehittäjä haluaa käyttää allekirjoitettua lupaa, ei järjestelmä sellaista hänelle

myönnä. Näin ollen ainoastaan ohjelmat, jotka ovat ennalta järjestelmään asennettu, voivat sellaisen luvan omistaa. (Felt ja muut, 2011.)

2.1.2 SELinux

SELinux (Security-Enhanced Linux) on kerneliin lisätty säädös, jonka avulla voidaan kontrolloida tarkemmin, kuinka järjestelmä voi suorittaa prosesseja ja objekteja (Shabtai, Fledel & Elovici, 2010). Tämän pakotetun pääsynhallinnan MAC avulla luodaan yksi lisäys Androidin tietoturvaan. Ennen Android-käyttöjärjestelmänversiota 4.3 oli sovelluksella järjestelmässä toiminnalleen hiekkalaatikko ja oma Linux-käyttäjätunnus (UID), jonka sovellus sai, kun asennus suoritettiin järjestelmään (SELinux, 2017). Tämän lisäksi SELinux tuotiin Android-käyttöjärjestelmään tarkentamaan hiekkalaatikon rajoituksia ja sovellusten oikeuksienhallintaan. Shabtai ja kumppanit (2010) tarkastelivat SELinuxin käyttöönottoa ja implementointia. He kohtasivat muutamaan haasteen, jonka Android ympäristönä aiheutti, mm. tiedostojärjestelmän (yaffs2) tuki SELinuxin vaatimalla automaattiselle tiedostojen tietoturva merkitsemiselle, koska yaffs2:ta puuttuu kyseinen ominaisuus. Saman huomion tekivät myös Smalley ja Craig (2013). Samalla he totesivat, että uudemmassa Android-versiossa on käytössä ext4 tiedostojärjestelmä ja näin ollen myös SELinuxin vaatima tuki automaattiseen tietoturvamerkintään tiedostoille. Toinen haaste oli Shabtain ja kumppaneitten (2010) mukaan SELinux-tuen lisääminen Android-kerneliin. Kolmas haaste liittyi siihen, miten init-prosessille saadaan SELinux-tietoturvamerkintä käyttöön.

Kaikesta huolimatta SELinuxista koituva hyöty verrattuna haittoihin on niin merkittävä, että Shabtain ja kumppaneitten (2010) mukaan tuki SELinuxille Android-järjestelmään kannattaa ottaa mukaan. Näin ollen voidaan määritellä tarkemmin prosessien ja tiedostojen käyttöä, vaikka prosessilla olisi root-oikeudet järjestelmässä. Shabtai kumppaneineen (2010) huomioivat, että juuri näitä root-oikeuden omistavia prosesseja init, mountd, debuggerd, zygote ja installd on suojattava. Vielä Android-versiossa 4.3 SELinux oli sallivassa tilassa edellä mainituille prosesseille, kun versiossa 4.4 tila muuttui pakotetuksi. Android-versiosta 5.0 lähtien SELinux on ollut pakotetussa muodossa käytössä koko käyttöjärjestelmälle (SELinux, 2017).

Smalley ja Craig (2013) testasivat omaa SELinux-implementaatiotaan tietoturvaan root-oikeuksien ja ohjelmien tietoturvan osalta. He löysivät SELinuxin tehokkaaksi tavaksi estää ohjelmien oikeuksien ja tahtomattoman tiedonvaihdon ohjelmien kesken. Näin ollen ohjelmien tiedot pysyvät paremmin niiden hallussa. Tiedetään myös, että SELinuxin avulla voidaan taata Android-laitteeseen asennettavien ohjelmien minimaaliset oikeudet ajaa itseään käyttöjärjestelmässä. Samalla voidaan kontrolloida root-oikeuksien kautta tapahtuvaa tietoturvauhkaa sovelluksien dataan (SELinux, 2017).

2.1.3 Android-tietoturvauhat ja tutkimus

Androidin avoimuus on luonut kiinnostusta tutkijoissa tutkia tietoturvaa ja kehitellä erilaisia parannuksia järjestelmään. Google Android Security Potential Harmful Applications Classification (PHA Classification, 2017) luokittelee erilaisia mahdollisia käyttäjään tai hänen tietoihinsa kohdistuvia uhkia. Näihin kuuluvat seuraavan kaltaiset ohjelmat:

- Takaovelliset -ohjelmat
- Kaupalliset vakoilu -ohjelmat
- DDOS (Distributed Denial of Service) -ohjelmat
- Vihamielinen lataaja -ohjelmat
- Laskuttavat -ohjelmat
- Muihin järjestelmiin vaikuttavat -ohjelmat
- Tietoja kalastelevat -ohjelmat
- Käyttöoikeuksia muuttavat -ohjelmat
- Lunnaita vaativat -ohjelmat
- Root-oikeuksia salassa vaativat -ohjelmat
- Roskapostia ja mainoksia lähettävät -ohjelma
- Vakoiluohjelmat
- Troijaohjelmat

Listalla ensimmäisenä on takaovelliset-ohjelmat, joissa on mahdollisesta suorittaa haitallista ohjelmakoodia käyttäjän sitä tahtomatta. Zhou, W., Zhou, Y., Jiang ja Ning (2012) löysivät tutkimuksessaan ohjelmien markettipaikkoja tutkiessaan uudelleen pakattuja Android-sovelluksia, joihin oli lisättynä ohjelman kylkeen muun muassa takaovia ja mainostulojen uudelleenohjausta hakkereiden tileille.

Toisena listalla tietoturvauhkana on kaupalliset vakoiluohjelmat, joilla voidaan tarkkailla käyttäjän toimia ilman, että kyseinen käyttäjä saa siitä tietoa (PHA Classification, 2017). Enck ja kumppanit (2014) esittelivät Taintdroidin[2] ja kuinka sillä voidaan tarkastella Android-sovelluksen tiedonkulkua Android-alustalla. He löysivät ohjelmia, jotka lähettivät laitteesta yksityisiksi tiedoiksi luokiteltavaa dataa ulkopuoliselle palvelimelle ilman käyttäjän informointia tai lupaa. Sarwar, Mehani, Boreli ja Kaafar (2013) testasivat Taintdroidin havanointimekanismia ja kehittivät tutkimuksessaan keinon, miten esimerkiksi vakoiluohjelman kehittäjät voisivat vältellä ja piilotella tiedonkulkua ohjelmasta ulospäin ilman, että jättäisivät siitä näkyvää jälkeä. Samaan kategoriaan voidaan lisätä tietojenkerääjäohjelmat, jotka tallentavat jotain seuraavista tiedoista ilman käyttäjän suostumusta: Asennetut ohjelmat, käyttäjätilit tai järjestelmän tiedostonimet. Valitettavan useat Android-sovellukset keräävät käyttäjistä edellä mainittuja tietoja, joten Shebaro, Oluwatimi, Midi ja Bertino (2014) kehittivät Identidroidin. Se on kustomoitu versio Android-käyttöjärjestelmästä, ja jonka avulla käyttäjä pystyy paremmin hallitsemaan tai estämään hänestä kerättävän datan määrää.

DDOS ovat ohjelmia, jotka saattavat käyttäjän tietämättään osaksi verkkoa, joka aiheuttaa esimerkiksi http-pyyntöjä verkkopalvelimelle, siten että palvelin kuormittuu ja toiminta hidastuu sekä lopulta kaatuu ylimääräisestä liikenteestä (PHA Classification, 2017). Esimerkiksi Chin, Felt, Greenwood ja Wagner (2011) löysivät tutkimuksessaan ohjelmia, jotka saattoivat altistaa käyttäjää osalliseksi tällaiseen ja samalla ohjeistivat, kuinka ohjelmankehittäjät voisivat estää sen, jos käyttäisivät Chin ja kumppaneitten (2011) kehittämää ComDroid työkalua. Shabtai, Kanonov, Elovici, Glezer ja Weiss (2012) esittelivät myös toisenlaisen DDOS muodon. Siinä laitteen akkua kuormitetaan mahdollisimman paljon. Hyökkäyksen kohteena olevan resursseja kulutetaan siten, että siitä saadaan toimimaton.

Vihamielinen lataaja-ohjelma on ohjelma, jossa itsessään ei ole mitään haitallista koodia, mutta joka sisältää mahdollisuuden ladata toisen haitallisen ohjelman. Esimerkiksi sitä mainostetaan tietoturvapäivityksenä, mutta todellisuudessa ladattuna ja asennettuna ohjelma aiheuttaa ongelmia käyttäjälleen. (PHA Classification, 2017.) Grace, Zhou, Zhang, Zou ja Jiang (2012) esittelevät tämäntyyppisen ohjelman toiminnan.

Vihamielinen ohjelma saattaa latautua samalla, kun ei-vihamielinen ohjelma latautuu. Se voi asentua samalla, kun toinenkin ohjelma asentuu ja näin ollen saada samat oikeudet käyttöönsä. Vaikka alkuperäinen ei-vihamielinen ohjelma poistettaisiin, voi vihamielinen ohjelma jäädä toimintakykyiseksi taustalle. (Grace ja muut, 2012.) Fahl ja kumppanit (2012) kehittävät ja demonstroivat tutkimuksessaan, kuinka toteutetaan tällainen hyökkäys.

Laskuttavat ohjelmat tuottavat käyttäjälleen rahallista menetystä ilman, että hän on suostunut maksamaan niistä tai tilaamaan mitään (PHA Classification, 2017). Tähän kuuluvat soittavat, tekstiviestittelevät ja palvelujen tilaamisia tekevät ohjelmat. Grace kumppaneineen (2012) huomioivat tutkimuksessaan, kuinka tällaisen ohjelman erottaminen niin sanotusti oikein toimivasta laskuttavasta sovelluksesta on haastavaa, koska yleensä metodi, joka johtaa laskuun tarvitsee käyttäjän toimia. Toisaalta ohjelma, joka laskuttaa ilman lupaa, toimii toisin eli taustalla ja ilman käyttäjän suostumusta (Grace ja muut, 2012).

Muihin järjestelmiin vaikuttavat ohjelmat eivät aiheuta Android-järjestelmälle haittoja, vaan voivat toimia ilman ongelmia, mutta kuormittavat esimerkiksi toisia järjestelmiä (PHA Classification, 2017). Esimerkki tällaisesta Android-sovelluksesta raportoitiin Zhang, Hu ja Jin (2017) toimesta. He löysivät Google Play -kaupasta sovelluksia, joihin oli piilotettu inline frame (IFrame) eli upotettu kehys saastuneelle verkkosivulle. Näissä sovelluksissa oli yhteistä se, että ne eivät olleet suoraan haitallisia itse Android-järjestelmälle, vaikka Webview ja javascriptin hyväksyntä voisivat myös aiheuttaa uhan suoraan Android-järjestelmälle. Tässä tapauksessa tietoturvauhka kohdistui Windows järjestelmään (Zhang ja kumppanit, 2017).

Tietoja kalastelevat -ohjelmat yrittävät saada käyttäjältä urkittua pankki- tai käyttäjätunnuksia sekä salasanatyyppistä tietoa (PHA Classification, 2017). Chin kumppaneineen (2011) huomioivat, kuinka esimerkiksi Android-käyttöliittymä ei informoi käyttäjää käynnissä olevasta sovelluksesta, joten käyttäjä voi vahingossa syöttää tietoja väärään ikkunaan. Xu ja Zhu (2012) demonstroivat tutkimuksessaan, kuinka ilmoitusvalikkoa voidaan hyödyntää ja käyttää tietojenkalastamiseen käyttäjältä.

Käyttöoikeuksia muuttavat ohjelmat yrittävät päästä käsiksi sellaiseen tietoon, mihin niille ei ole lupaa myönnetty. Näin ollen ne rikkovat Android-järjestelmän lupakehysmallia, estävät ohjelman poiston tai poistavat turvallisuusmekanismeja käytöstä (PHA Classification, 2017). Grace, Zhou, Jiang ja Sadeghi (2012) tutkivat mainoksien aiheuttamaa turvallisuusuhkaa, ja osoittivat, että monet mainoksien kirjastot vaativat huomattavan paljon tietoa silloin, kun ne on implementoitu Android-sovelluksen sisään. Tutkimuksen löydöksinä Grace ja kumppanit (2012) mainitsivat tungettelevasti yksityistietojen keräämisen, niiden jakamisen mainostajille ja dynaamisen koodin lataamisen turvattomasti, kun mainoksia esitetään käyttäjälle. Käyttöoikeuksien eskalaatio voi tapahtua seuraavan kaltaisen välitysketjun avulla. Ohjelma A käyttää ohjelman B kautta ohjelman C komponenttia, jonka käyttöön ohjelmalla A ei ole käyttöoikeutta suoraan kutsumalla ohjelma C komponenttia. (Davi, Dmitrienko, Sadeghi, & Winandy, 2010.)

Lunnaita vaativat ohjelmat voivat esimerkiksi salata käyttäjän tiedot ja vaativat rahaa, jotta käyttäjä saisi tiedon hallinnan takaisin itselleen (PHA Classification, 2017). Felt, Finifter, Chin, Hanna ja Wagner (2011) totesivat artikkelissaan, että lunnaita vaativat ohjelmat ovat tyypiltään käyttäjän näytön lukitsevia, jolloin laitteen käyttö estyy. Jotta lukituksen voi poistaa, käyttäjä joutuu maksamaan kiristäjälle lunnaat pääsystä laitteeseen. Kehittyneenpää versiota lunnaista vaativista haittaohjelmista edustaa koko

Android-laitteen tai sen tiedostot salaava ohjelma. Andronio Zanero ja Maggi (2015) kehittivät menetelmän, jolla voidaan tutkia Android-sovelluksia ja niiden mahdollisia kryptografisia ominaisuuksia haittaohjelmien varalta. Heidän tutkimuksensa pohjalta määritelmä lunnaita vaativalle ohjelmalle oli, että siihen sisältyvät salaus, lukitus ja uhkailu. Jos kaikki nämä kohdat löydetään ohjelmasta, luokitellaan se lunnaita vaativaksi kiristysohjelmaksi.

Root-oikeuksia salassa vaativa ohjelma muuttaa ohjelman ajo-oikeutta järjestelmässä ilman, että käyttäjä on tietoinen tästä (PHA Classification, 2017). Zhou ja Jiang (2012) löysivät tutkiessaan haittaohjelmia, että jotkin niistä sisälsivät root-oikeuksia salaa asentavia ominaisuuksia. Tällainen ohjelma piilottaa itsensä salaamalla koodinsa ja näin ollen haittaohjelman löytyminen ja tutkiminen Android-laitteesta on haastavaa. Toinen ongelma tällaisessa root-oikeudet omaavassa haittaohjelmassa on, että se pääsee ulos ohjelman omasta hiekkalaatikosta ja näin ollen pystyy suorittamaan käskyjään vapaammin Android-järjestelmässä. (Zhou ja Jiang, 2012.)

Roskapostia tai mainoksia lähettävät-ohjelmat mainostavat käyttäjän laitteessa oleville yhteystiedoille (PHA Classification, 2017). Osa mainostenvälityskirjastoista voitaneen lukea kuuluvan tähän kategoriaan. Grace, Zhou, Jiang ja Sadeghi (2012) huomioivat mainoskirjastojen laajan lupavaltuuksien ja tiedonkeräämisen. He totesivat, että Android-laitteen käyttäjällä ei ole tietoa siitä, mitä mainoskirjastoa ohjelma käyttää ja näin ollen voi aiheuttaa tietoturvauhkia käyttäjälleen.

Vakoiluohjelmat vievät käyttäjältä tämän yksityisiä tietoja sekä tiedostoja ilman lupaa ja suostumusta (PHA Classification, 2017). Vuonna 2011 Chin ja kumppanit tutkivat Android-ohjelmien sisäistä kommunikaatiota ja he totesivat, kuinka ohjelma kehittäjän tulee olla tarkkana, jottei käyttäjälle aiheudu tahatonta vaaraa tietojen karkaamisesta. Wei, Roi ja Ou (2014) löysivät Android-ohjelmista monenlaisia ongelmia, jotka johtuivat osaamattomista ja huolimattomista ohjelmistojenkehittäjistä. Muun muassa salasanojen, OAuth valtuuksien ja salausalgoritmien käytössä löytyi puutteita. Tällöin käyttäjän on vaikeata suojautua tällaisia uhkia vastaan, jos jo niin sanotuissa luotettavien sovelluksien tiedonkäsittelyssä on puutteita. (Wei ja muut, 2014.) Soundcomber-haittaohjelmalla Schlegel, Zhang, Zhou, Intwala, Kapadia, ja Wang (2011) osoittivat, kuinka monimuotoisesti tietoa voidaan kerätä ja lähettää Android-laitteesta. He käyttivät tiedonkeräämiseen, prosessointiin ja välittämiseen Android-alustan sensoreita.

Trojan-ohjelma on tyypiltään esimerkiksi pelin mukana tuleva ohjelma, joka lähettää tekstiviestejä maksullisiin numeroihin ilman käyttäjän lupaa ja suostumusta (PHA Classification, 2017). Schlegel kumppaneineen (2011) kehitti esimerkin vuoksi Soundcomber-haittaohjelman, jolla he pystyivät näyttämään, kuinka kaksi erillistä ja omilla oikeuksillaan olevaa Android-sovellusta voivat yhdessä aiheuttaa potentiaalisen tietoturvaongelman. He pystyivät keräämään käyttäjältä pankki- ja luottokorttitietoja onnistuneesti. Zhou kumppaneineen (2012) tutki Android-sovelluskauppoja ja löysi niistä uudelleenpakattuja ohjelmia. Virallisesta Google Play-kaupasta ladattuja sovelluksia oli uudelleenpakattu ja niihin oli lisättynä haitallisia ominaisuuksia, joihin kuuluivat muun muassa maksullisiin tekstiviestinumeroihin viestien lähettäminen, kirjanmerkkien muokkaaminen Android-laitteessa ja sekä toisten ohjelmien lataaminen ja asentaminen laitteeseen. Lisäksi Android-laitteesta tuli osa suurempaa botnettiä ja siihen voitiin syöttää komentoja komento ja kontrolli (C & C) -palvelimen välityksellä. (Zhou ja muut, 2012.)

Lisäksi yksi merkittävä uhka Android-laitteelle ja sen käyttäjälle tulee Androidin järjestelmäpäivitysten hitaasta saapumisesta päätelaitteelle. Thomas, Beresford ja Rice

(2015) esittelivät tutkimuksessaan, kuinka monimutkainen prosessi kriittisten järjestelmäpäivitysten matka on. Heidän tutkimuksensa mukaan päivitysketjun jarruna toimii laitteiden valmistajat ja heidän hidas ja jopa olematon päivitysten tekeminen. Nopeuttaakseen päivitysten julkaisua päätelaitteille saakka, esiteltiin Android One (2020) projekti. Tällä pyritään varmistamaan laitteiden tietoturva päivitykset kolmeksi vuodeksi ja Android-versio päivitykset vähintään kahdeksi vuodeksi Android-laitteille. Thomas ja kumppanit (2015) tarkastelivat tutkimuksessaan yhtätoista kriittistä Android-haavoittuvuutta, jotka olivat tyypiltään sellaisia, että niiden avulla laitteen lupaoikeudet voitiin muuttaa root-tasolle. Nopeimmin päivitykset tulivat Googlen Nexus-laitteille. Tarkasteluvälillä toiseen laitevalmistajan malliin nähden Nexus oli saanut 49 päivitystä, kun toiseen oli vain tullut yksi päivitys. Android One (2020) asettaa omat rajoituksensa esimerkiksi käyttöliittymään modifiointiin valmistajan taholta, mutta samalla se varmistaa loppukäyttäjälle varmuuden tietoturvapäivitysten saatavuudesta nopeasti ja varmemmin.

Yhteenvetona voidaan todeta, että esitellyt Android-järjestelmälle haitalliset ohjelmatyypit ja niiden mahdollinen estäminen ovat olleet monen tutkijan tarkastelun kohteena. Kokonaisuudessa Android-käyttöjärjestelmänä on ollut avoimuudestaan hyvin otollinen tutkimuskohde. Se on ollut hyvin aktiivisen kehityskohteena koko olemassaolonsa ajan. Tietoturvauhat ja niiden estäminen ovat tulevaisuudessakin hyvin haasteellista ja päivitysten saapumisen Android-päätelaitteille pitäisi valmistajien taata paremmin kuin tällä hetkellä.

2.2 Vaatimusmäärittely

Yksinkertaisimmillaan ja yleistämällä ohjelmistokehitysprojektiin kuuluu kaksi vaihetta, analyysi ja koodaus. Näin totesi Royce (1970), kun hän esitteli vesiputousmallin. Vaikka tarkemmin katsottaessa, hänen esittämänsä malli oli useampivaiheinen prosessi, kun ohjelmistoa kehitetään tai kun sen koko on suurempi kuin organisaation sisäiseen käyttöön tarkoitetulla ohjelmalla. Vesiputousmallissa on viisi erivaihetta, joista ensimmäisenä on vaatimusmäärittely. Toisena vaiheena on suunnittelu. Järjestelmän toteutus eli implementointi tulee kolmantena. Testausvaihe on neljäntenä ja viimeisenä vaiheena on ylläpito. Vesiputousmallissa oletuksena on, että jokaisen vaiheen jälkeen siirrytään seuraavaan vaiheeseen, kunhan vain osa-alue on katselmoitu ja hyväksytty. Täten vaatimusten muutokset ovat kalliita ja vaativat paljon työtä, varsinkin jos ohjelmiston kehitysprosessi on jo pitkällä. (Royce, 1970.)

Vaatimusmäärittely haarautuu osa-alueisiin, joihin kuuluvat reaalimaailman tavoitteet ja tehtävät sekä rajoitteet kehitettävällä järjestelmälle. Lisäksi siihen kuuluu näiden osa-alueitten yhteys toisiinsa, mikä määrittelee tarkasti kehitettävän järjestelmän käyttäytymisen ja kehitysprosessin evoluution ajan myötä sekä vaikutukset ympäristöön, johon järjestelmää määritellään. Vaatimusten määrittely voidaan tyypittää funktionaalisiin, ei-funktionaalisiin ja toimintaympäristö vaatimuksiin. Funktionaaliset vaatimukset kuvaavat miten kehitettävän järjestelmän palvelut tuottavat, kun ne saavat jonkinlaisen herätteen toimintaan. Niitä voidaan kuvata muodoltaan yleispätevästi ja korkean tason kuvauksina. Vaihtoehtoisesti ne voivat olla hyvin tarkkoja kuvauksia siitä mitä järjestelmän sisään tulevista ja sieltä lähtevistä toiminnoissa tapahtuu sekä ehdot näille tapahtumille (Laplante, 2013). Paetsch, Eberlein ja Maurer (2003) totesivat ettei ei-funktionaalisia vaatimuksia käsitellä ketterissä menetelmissä tarpeeksi. Tästä esimerkkinä he mainitsivat mm. vaatimukset ylläpidettävyydelle tai siirrettävyydelle, turvallisuudelle ja suorituskyvylle. Samalla he totesivat, että niiden käsittely pitäisi määritellä selvemmin. Laplante (2013) totesi, että ei-funktionaaliset vaatimukset

kuvaavat ja määrittelevät ympäristöä, jossa kehitettävä järjestelmä toimii. Näihin ympäristötekijöihin kuuluvat tuotteen vaatimukset, organisaation vaatimukset ja ulkopuoliset vaatimukset. Tuotteen vaatimukset -kategoriaan kuuluvat luotettavuus, tietoturva, käytettävyys ja suorituskyky, jonka alaisuuteen kuuluvat tehokkuus ja tilavaatimukset. Organisaatiovaatimuksiin sisältyvät ympäristö, operaatio ja kehitysvaatimukset. Ulkopuoliset vaatimukset tulevat säädöksistä, etiikasta ja lainsäädännöstä, joka määrittelee kirjanpidon ja turvallisuuteen liittyviä vaatimuksia. (Laplante, 2013.)

Vaatimusmäärittelyprosessiin Sommervillen (2005) mukaan kuuluvat seuraavat vaiheet: Löytäminen, siinä yritetään saada selville asiakkaan toiveet ja tarpeet. Samalla pyritään identifioimaan vaatimusten sidosryhmät. Myös ei-funktionaalisten vaatimusten etsiminen ja löytäminen kuuluvat prosessiin. Analysointi, jossa edellisessä vaiheessa löydetyt ns. hiomattomat vaatimukset muokataan. Samalla käydään läpi vaatimusmäärittelyn päällekkäisyyksiä, vaillinaisuuksia, epäjohdonmukaisuuksia ja järjettömyyksiä, eli sovitetaan niistä ehjempi kokonaisuus. Kolmantena vaiheena on vahvistus, jolloin ohjelmiston vaatimukset esitellään sidosryhmille ja tarkistetaan, vastaavatko ne heidän asetettuja toiveita ja tarpeita. Seuraavana on neuvottelu, jolloin sidosryhmien eriävät tarpeet tulee neuvotella ja katsoa voitaisiinko sopia yhtenäiset ja kaikkia osapuolia tyydyttävät vaatimukset kehitettävälle järjestelmälle. (Sommerville, 2005.) Laplante (2013) korosti vaikeutta sidosryhmien priorisoinnissa, ja siinä kuinka saadaan jokaisen sidosryhmän osasen vaatimukset mukaan kehitykseen. Viidentenä vaiheena Sommerville (2005) vaatimusmäärittely listalla oli dokumentointi. Tällöin vaatimukset pitää kirjata niin, että sidosryhmän jokainen jäsen sekä kehitystiimin järjestelmän kehittäjät ymmärtävät ne. Viimeisenä kohtana oli vaatimusten hallinta, jotta järjestelmään mahdollisesti tulevat muutokset ja täydennykset ovat jäljitettävissä.

Laplanten (2013) mukaan vaatimusmäärittely ketterissä menetelmissä on vapaamuotoisempaa kuin perinteisissä kehitysmenetelmissä, joissa vaatimusmäärittelydokumentointi on virallinen ja muodollinen vaihe ohjelmistokehitysprosessia. Abrahamsson, Salo, Ronkainen ja Warsta (2002) totesivat, että melkein kaikissa ketterissä menetelmissä vaatimusmäärittely kuuluu osaksi menetelmää. Agile manifesto (2017) sisältää 12-kohtaisen listan perusperiaatteista ketterissä menetelmissä. Ensimmäisenä on tärkein periaate eli asiakastyytyväisyys, joka saavutetaan nopeilla ja jatkuvilla julkaisulla asiakkaalle hyödyllisestä ohjelmistosta. Toisena periaatteena vaatimusmäärittelyn muutokset ovat tervetulleita, jopa myöhäisessä vaiheessa kehitystä. Ketterät menetelmät tukevat muutosta, jotta asiakkaalle olisi siitä kilpailullista hyötyä. Kolmantena periaatteena on tuottaa toimiva ohjelmisto säännöllisesti aikavälillä kahdesta viikosta kahteen kuukauteen, suosien lyhyempää aikaväliä. Neljäntenä korostetaan asiakkaan ja kehittäjän yhteistyötä päivittäisenä toimena läpi koko projektin. Viidentenä periaatteena on projektin rakenne ja ympäristö, jossa on motivoituneita yksilöitä ja joita tuetaan sekä luotetaan heihin toteutuksen tekemisessä. Kuudentena periaatteena on kasvotusten kommunikoinnin tehokkuus ja hyöty informaation jakamiseen kehitystiimissä. Seitsemäntenä periaatteena on toimivan ohjelmiston toiminta mittarina kehitysprosessin etenemisestä. Kahdeksantena periaatteena ketterien menetelmien prosessi suosii kestävää kehitystä, jossa sidosryhmien jäsenten työtahti pitäisi pysyä tasaisena. Yhdeksäntenä periaatteena jatkuva huomio tekniseen erinomaisuuteen ja hyvään suunnitteluun, niin että se edesauttaa ketteryyttä. Kymmenentenä periaatteena on yksinkertaisuus, korostetaan tekemättömän työn keskeisyyttä. Yhdentenätoista periaatteena on arkkitehtuurin, vaatimusmäärittelyn ja suunnittelun esiin tuleminen itsejärjestäytyvästä kehitystiimistä. Viimeisenä periaatteena on tiimin toistuva itsetutkiskelu, jotta siitä tulisi tehokkaampi. (Agile Manifesto, 2017;

Laplante, 2013.) Ketterien menetelmien perusperiaatteet antavat tunnustusta vaatimusmäärittelyn käsitteelle ja omaksumiselle läpi prosessin (Laplante, 2017).

Cao ja Ramesh (2008) tunnistivat seitsemän vaatimusmäärittelykäytäntöä ketterien menetelmien vaatimusmäärittelystä organisaatioissa. Kasvokkain kommunikoinnin tärkeyttä kirjoitettuun dokumentaatioon nähden korostettiin kaikissa tutkimuksessa mukana olleissa organisaatioissa. Paetsch ja kumppanit (2003) totesivat asiakkaan osallistumisen tärkeäksi ei vain vaatimusmäärittelyssä, mutta muutenkin koko projektin ajan. Cao ja Rameshin (2008) vaatimusmäärittely toteutetaan käyttötapauskaaviona tai lyhyesti kirjoitetuilla käyttäjäkertomuksilla kehitettävästä järjestelmästä. Kun kommunikointi tapahtuu verbaalisesti ja asiakkaan ymmärtämällä tavalla, vaatimukset kehittyvät ja muovautuvat tarkemmiksi ohjelmiston kehitysprojektin kuluessa. Tämä epämuodollinen kommunikointi poistaa tarpeen aikaa kuluttavasta dokumentoinnista ja hyväksyntä prosesseista. Paetsch ja kumppanien (2003) totesivat, että dokumentointi ketterissä menetelmissä on mukana, mutta ei pääosassa. Heidän mielestään täydellisen vaatimusmäärittelyn dokumentointi on toteutuskelvotonta ja kallista. Cao ja Ramesh (2008) mukaan kommunikaation tulee olla jatkuvaa, että asiakas on kehitystiimin käytettävissä jatkuvasti, jotta sidosryhmien yhteisymmärrys tulee ilmi. Lisäksi he totesivat, että luottamus kehitystiimin ja sidosryhmien välillä on tarpeen varsinkin alkuvaiheessa, kun määritellään vaatimuksia kehitettävälle järjestelmälle.

Iteratiivinen vaatimusmäärittely oli Cao ja Rameshin (2008) toinen löytämä menetelmä vaatimusten määrittelylle ketterissä menetelmissä. Sen mukaan ensiksi määritellyt korkean tason vaatimukset tarkentuvat. Jokaisen sprintin alussa asiakkaan kanssa käytävä keskustelu kehitystiimin kanssa edesauttaa tarkempien vaatimusten löytämisessä. Näin ollen voidaan muodostaa alustava suunnitelma ja jopa implementointisuunnitelma. Kun asiakas on läsnä kehitystyössä, kehittäjät voivat pyytää ja saada tarkennuksia vaatimuksiin nopeasti. Tämä selkeyttää ja tekee vaatimuksia ymmärrettävämmiksi, asiakas kokee saavansa juuri sen mitä haluaa. Abrahamsson ja muut (2002) totesivat myös ketterien menetelmien vaatimusmäärittelystä, että kun asiakas on läsnä niin he voivat tuoda uusia vaatimuksia kehitykseen. Huonoina puolina Cao ja Ramesh (2008) pitivät vaikeutena määritellä hintaa ja kestoa projektille sekä dokumentaation vähyyttä. Lisäksi ei-funktionaalisten vaatimusten laiminlyöntiä aikaisessa vaiheessa projektia pidettiin ongelmallisena, koska se voi aiheuttaa skaalautumattomuutta, ylläpidon vaikeutta, siirrettävyyden ja tietoturvaongelmia sekä suorituskyvyn laskua. Nämä kohdat paljastuvat yleensä vasta, kun järjestelmää aletaan ottamaan käyttöön isommassa mittakaavassa. Paetsch ja kumppanit (2003) löysivät myös ei-funktionaalisien vaatimusmäärittelyn ongelmallisena ketterissä menetelmissä, koska niitä ei käsitellä tarpeeksi, vaikka ovat hyvin tärkeitä ja niiden käsittelyyn tulisi panostaa enemmän.

Vaatimusten jatkuva priorisointi tapahtuu Caon ja Rameshin (2008) sekä Paetschin ja kumppanien (2003) mukaan liiketoiminnan vaatimukset edellä, jotta asiakas hyötyisi siitä eniten. Kun asiakas on läsnä jatkuvasti kehitystiimin apuna, priorisointia voidaan tehdä missä tahansa kehitysvaiheessa. Tällöin kehitystiimi voi toteuttaa sidosryhmien vaatimukset paremmin. Liiketoimintavaatimusten priorisointi voi aiheuttaa negatiivisen vaikutuksen muihin vaatimuksiin, kuten skaalautuvuuteen, ja se voi pahimmillaan vaikuttaa koko projektin onnistumiseen. Cao ja Ramesh (2008) myös huomioivat, että jatkuva priorisointi aiheuttaa järjestelmään epävakautta, jos ei olla hyvin huolellisia. Paetsch ja kumppanit (2003) totesivat priorisointia tapahtuvan vaatimuksille koko ohjelmistoprojektin ajan, koska uusia vaatimuksia tulee jatkuvasti.

Jatkuvalla suunnittelulla hallitaan muutoksia vaatimuksissa. Cao ja Ramesh (2008) totesivat, että uusien ja olemassa olevien vaatimusten muutos sekä tarvittaessa poisto

muokkaavat ketterissä menetelmissä ohjelmistosta sellaisen, että asiakkaan tarpeet täyttyvät. Highsmith ja Cockburn (2001) totesivat, että ketterissä menetelmissä vaatimusten muutoksen hallinnan tekee kehitystiimi ja asiakas yhdessä, kunhan muutoksen koko ei muuta tai riko isompaa kokonaisuutta. Cao ja Ramesh 2008 totesivat, että kun jatkuvasti evaluoidaan vaatimuksia, tarve isojen muutosten tekemiseen vähenee ja näin ollen kehityskustannukset laskevat. Toisaalta se voi myös aiheuttaa kustannuksia, jos alkujaan on valittu väärä arkkitehtuuri projektin toteutukseen. Myös koodin uudelleen järjestelyllä eli refaktoroinnilla pyritään lisäämään ymmärrystä ja helpottamaan muokkaamista ilman, että järjestelmän käytös muuttuu. Kuitenkin ongelmaksi refaktoroinnissa Cao ja Ramesh (2008) totesivat kehittäjätiimin osaamisen sekä aikataululliset ongelmat. Voi myös käydä niin, että järjestelmän refaktoroinnilla ei saada tarpeeksi hyötyä, ja olemassa oleva joudutaan kirjoittamaan kokonaan uusiksi ja tämä nostaa järjestelmän kehityskustannuksia. Boehm (2002) mainitsi myös samanlaisesta ilmiöstä refaktoroinnista, eli että osaavan tekijän pienessä järjestelmässä tekemät muutokset ovat käytännössä ilmaisia. Kun järjestelmän koko ja vaatimuksien määrä kohoaa tietyn pisteen yli, niin järjestelmän arkkitehtuuri rikkoutuu. Tästä aiheutuu kustannuksia, kun kokonaisuus pyritään parsimaan takaisin toimivaksi systeemiksi.

Cao ja Rames (2008) löysivät tutkiessaan organisaatiota, että niissä käytettiin prototypointia vaatimusmäärittelyssä. Tällöin esille tulleet vaatimukset voidaan helpommin testata asiakkaalla ja saada selville tarkennuksia kehitettävälle järjestelmälle. Samankaltaisen huomion tekivät myös Highsmith ja Cockburn (2001) ketteristä menetelmistä. Cao ja Rames (2008) täsmensivät, että vaarana tässä lähestymistavassa ovat asiakkaan vääristyneet odotusarvot ja hätäily prototyypin käyttöönottamisella. Prototyypin käyttöönotto voi aiheuttaa ongelmia järjestelmän ylläpidettävyyteen, tietoturvaan ja skaalautuvuuteen. Prototyypin esittely voi aiheuttaa asiakkaalle tuntemuksen valmiimmasta tuotteesta kuin se todellisuudessa on. Näin ollen he eivät ole halukkaita jatkamaan kehitystyötä, vaikka järjestelmän implementointi sitä vaatisikin. (Cao ja Ramesh, 2008.)

Testivetoisessa kehityksessä järjestelmän ohjelmoija kirjoittaa ensiksi testin ennen kuin kirjoittaa toiminnallisen koodin (Cao ja Ramesh, 2008). Tällöin testi toimii osana vaatimusmäärittelyä, jossa järjestelmän testi määrittelee kirjoitettavan koodin käyttäytymistä. Paetsch ja kumppanit (2003) huomioivat saman ominaisuuden ketterissä menetelmissä olevasta Extreme Programming (XP), jossa testit kirjoitetaan ensiksi, sitten vasta varsinainen koodi implementoidaan. Organisaatioissa näin ollen käytetään testejä osana vaatimusmäärittelyä, jonka avulla voidaan koodi ja vaatimusmäärittely linkittää keskenään. Kun järjestelmän testit on kehitetty, on mahdollista kokeilla erilaisia tapoja toteuttaa tahdotut ominaisuudet. Muutoksien aiheuttamat virheet auttavat koodin implementointia. Cao ja Rames (2008) listasivat testivetoisen kehityksen ongelmiksi kehittäjien tottumattomuuden kirjoittaa testit ennen koodia ja vaatimusten ymmärtämistä, joiden ymmärtäminen vaatii kommunikointia asiakkaan kanssa. Testivetoinen lähestyminen vaatii kurinalaisuutta ja järjestelmällisyyttä.

Katselmointi ja hyväksyntätestaus olivat viimeisenä kohtana, kun Cao ja Ramesh (2008) listasivat tutkimuksessaan vaatimusmäärittelykäytännöistä organisaatioissa. Tällä tavoin useat organisaatiot varmistivat toteutuksen oikeellisuuden. Vaikka katselmointi toimii monesti tapana raportoida järjestelmän kehityksen etenemisestä projektin sidosryhmille, niin sillä myös pyritään lisäämään asiakkaan luottamusta kehittäjiin. Lisäksi katselmoinnin avulla pyritään löytämään mahdolliset ongelmat. Vaikka hyväksyntätestausta pidettiin useassa organisaatiossa hyvänä tapana määritellä vaatimusten implementointia kehitettävään järjestelmään, ongelmaksi nousi hyväksyntätestauksen suunnittelu asiakkaan päässä. (Cao ja Ramesh, 2008.)

Ketterien menetelmien vaatimusmäärittely voidaan kirjoittaa käyttäjäkertomuksina, jotka ovat muodoltaan: <Käyttäjänä tyyppinä> haluan <tehdä jotain>, jotta voin <saavuttaa jonkin tavoitteen>. Sidosryhmät yleensä kirjoittavat käyttäjäkertomukset. (Laplante, 2013; User story, 2017.) Toisaalta Leffingwell (2010) toteaa ketterien menetelmien käyttäjäkertomuksista verrattuna perinteiseen vaatimusmäärittelyyn, etteivät ne ole suoraan verrannollisia. Hänen mukaansa käyttäjäkertomukset eivät ole tarkkoja määritelmiä vaan enemmän neuvoteltavissa olevia kuvauksia siitä, mitä järjestelmän tulisi tehdä. Leffingwellin (2010) ja Laplanten (2013) mukaan ne voidaan kirjoittaa yksinkertaisesti indeksikorteille, jos kehitystiimi toimii samassa tilassa. Tällöin kehitystiimille syntyy yhteinen visio siitä, mitä he alkavat sitoutuneemmin toteuttamaan yhdessä. Kuvassa 2 näkyy indeksikortin sisältö esimerkillisesti. Tällöin käyttäjäkertomuksen tieto voidaan siihen kirjata seuraavasti: Indeksikortin numero, otsikko ja lyhyt kuvaus käyttäjäkertomuksesta. Hyväksyntätestin avulla käyttäjäkertomus voidaan validoida. Jokaisen käyttäjäkertomuksen tulee olla testattavissa, jotta se täyttää vaatimuksen määritelmän (Leffingwell, 2010; Laplante, 2013).

Kuva 2. Indeksikortti.

Prioriteettiarvo perustuu projektissa määriteltyihin spesifikaatioihin ja voi olla muodoltaan numeerinen tai sanallinen. Kertomuspisteet kuvaavat implementointiaikaa, se kuvastaa arviota, kuinka kauan sen toteuttamiseen oletetaan kuluvan aikaa. Kuvauskohtaan kirjataan yhdestä kolmeen lauseen mittainen kuvaus, mitä käyttäjä haluaa tehdä, jotta saavuttaa jonkin asian. (Laplante, 2013; User story, 2017.)

Jos edellä esitetty käyttäjäkertomus osoittautuu liian korkeantason kuvaukseksi, voidaan käyttäjältä kysyä tarkennuksia siitä, mitä hän on tarkoittanut (Laplante,2013). Kun käyttäjäkertomuksen toteutusta mietitään, voidaan se mallintaa vuokaaviolla tai aktiviteettikaaviolla (User story, 2017). Leffingwellin (2010) mukaan käyttäjäkertomuksissa on mukana lupaus keskustelusta, jossa kehitystiimi, asiakas, tuotteen omistaja ja muut sidosryhmät yhdessä tarkentavat, mitä on tarkoitus järjestelmään kehittää, koodata ja testata. Hyväksyntätestit vahvistavat, että käyttäjäkertomukset ovat toteutettu sillä tarkkuudella, jolla asiakkaan tarpeet täyttyvät (Leffinwell, 2010). Hyväksyntätesti (testimetodi) kuvassa 2 voidaan kirjoittaa muotoon **Annettu** (Given) <*Tietty järjestelmän tila*> **Milloin** (When) <*Jokin tapahtuma tapahtuu*> **Sitten** (Then) <*Järjestelmän tila on muuttunut ja vaihtunut*> ja se voidaan ohjelmallisesti suorittaa esimerkiksi Robot Framework:ssa. (Attd, 2020; Robot framework, 2017.)

Vaatimusmäärittely teknologialle ja rajoitteet, missä ympäristössä sovellus toimii, ovat tyypiltään ei-funktionaalisia vaatimuksia (Laplante, 2013). Nekin voidaan kirjoittaa indeksikortille. Leffingwell (2010) totesi ei-funktionaalisten vaatimuksien olevan yleensä tyypiltään käytettävyyteen, luotettavuuteen, suorituskykyyn ja tuettavuuteen liittyviä. Hänen mukaansa käytettävyyden määrittely voi olla haastavaa, mutta esimerkiksi käytettävyyttä voidaan verrata kilpailevaan tuotteeseen. Luotettavuudesta Leffingwell (2010) totesi, että se on yksi tärkeimmistä vaatimuksista, millä asiakastyytyväisyys voidaan todentaa. Samaa luotettavuuden todentamista voidaan tarkastella saatavuudella, mittaamalla aikaa häiriöitten välillä tai miten kauan menee aikaa siihen, että virheet korjataan tarkkuudella, virheettömyydellä ja tietoturvalla. Leffingwellin (2010) mukaan suorituskyky on hyvin laaja käsite, mihin kuuluu mm vasteaika, suoritusteho, kapasiteetti, rappiotilat ja resurssien jakaminen. Lisäksi hän mainitsee ylläpidettävyyden, missä määritellään, miten muutokset voidaan toteuttaa kehitettävään järjestelmään. Cao ja Ramesh (2008) totesivat ei-funktionaalisista vaatimuksista, että ne tahtovat unohtua asiakkaalta, koska heidän fokuksensa on ydintoiminnallisuuksissa. Yhden poikkeuksen he löysivät, ja se oli käytettävyys ja tarkemmin helppokäyttöisyys. He huomauttivat, että ei-funktionaaliset vaatimukset, kuten turvallisuus ja suorituskyvyn unohtaminen alkuvaiheessa, saattaavat aiheuttaa myöhemmissä vaiheissa ongelmia.

2.3 Järjestelmän arkkitehtuuri

Tässä luvussa käydään läpi järjestelmän arkkitehtuuriin liittyvä aikaisempi tutkimus. Ensiksi tarkastellaan Android-sovellusten arkkitehtuuria, minkä jälkeen käydään läpi palvelinsovelluksen arkkitehtuuria ja viimeisenä teknologian arkkitehtuuria.

2.3.1 Android-sovelluksen arkkitehtuuri

Android-sovelluksen arkkitehtuuri voidaan toteuttaa monella tavalla. Tässä tarkastellaan niitä vaihtoehtoja, mitä konstruktiossa tullaan käyttämään. Ensiksi tarkestellaan WebView-sovelluksen arkkitehtuuri Android-alustalla. Toisena JSON (JavaScript Object Notation) -sovelluksen arkkitehtuuri ja kolmantena on PWA (Progressive Web Application) -sovelluksen arkkitehtuuri.

Hyvin yksinkertainen Android-sovellus ei vaadi kuin muutaman pakollisen tiedoston, jotta sillä voidaan näyttää internetistä sisältöä Android-päätelaitteella. WebView-luokan avulla Android-ohjelmassa voidaan käyttää ja näyttää internetsivun sisältöä ohjelman aktiviteetin sisällä (Webview, 2017). Sivun sisällön hakemiseen tarvitaan lupa ja se täytyy lisätä AndroidManifest.xml-tiedostoon.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Luvan esittely pitää olla <manifest> ja </manifest> elementtien sisällä, tarkempi sisältö on liitteessä A. AndroidManifest.xml. Lisäksi Feltin ja kumppaneitten (2011) ja Permissions groups (2017) mukaan Android-sovelluksen muut mahdolliset lupaa vaativat luvat täytyy lisätä kyseiseen tiedostoon.

Lisäksi tarvitaan MainActivity.java-luokka, jossa luodaan aktiviteettiin uusi WebView.

```
WebView myWebView = new WebView(this)
```

Seuraavaksi myWebView avulla ladattava sivun tiedot määritellään.

```
myWebView.loadUrl("ip-address or domain");
```

Tässä ip-osoite tai domain-nimen perusteella haetaan aktiviteetissä näytettävä sivuston sisältö. Lisäksi Android-ohjelman navigointi WebView-sovelluksessa mahdollistetaan javascriptin avulla lisäämällä seuraava:

```
webSettings.setJavaScriptEnabled(true);
```

Kun arvoksi on asetettu true eli tosi, javascript-tuki on sovelluksen käytössä. Ohjelman toteutus on tehtävä niin, että käyttäjän pysyessä ennalta määritellyn ip-osoite tai domain-alueen sisällä, käyttäjä käyttää WebView-aktiviteettiä. Siinä vaiheessa, kun käyttäjä haluaa siirtyä muualle ennalta määrätystä alueesta sovelluksessa, hänelle avautuu valikko, mistä valitaan käytettävä selainohjelma sisällön näyttämiseen. Tällöin käyttäjä jatkaa toisella ohjelmalla informaation selaamista (WebView, 2017). Lisäksi Android-ohjelman toteutuksessa tarvitaan activity_main.xml tiedosto, jossa määritellään ohjelman ulkoasu. Liitteessä C activity_main.xml on esitetty esimerkki konstruktion toteutuksesta.

Luo, Hao, Du, Wang ja Yin (2011) tutkivat mahdollisia ja potentiaalisia tietoturvauhkia Android-sovelluksissa, kun kehitetään Android-sovelluksia, missä käytetään WebView komponenttia. He toteavat, että käytettäessä WebView Android-sovelluksessa siinä vaarannetaan Android-sovelluksen hiekkalaatikko, missä ohjelmat ovat omissa ympäristöissään ja näin ollen annetaan mahdollinen hyökkäysväylä Android-järjestelmään. Tähän kuitenkin on reagoitu Android-järjestelmän kehittäjien toimesta (WebView, 2017). Nykyinen WebView-toteutus on muuttunut Luon ja kumppanien (2011) löydöksien jälkeen ja WebView-komponenttia päivitetään jatkuvasti, myös vanhempiin Android-versioihin. Toteutuksen implementoinnissa joudutaan tarkastelemaan sitä, mille Android-versiolle Android-sovellusta ollaan tekemässä (WebView, 2017).

Toinen mahdollinen tapa toteuttaa Android-sovellus on hakea palvelimelta tieto ja näyttää tämä haettu tieto Android-sovelluksen käyttöliittymässä halutulla tavalla. Esimerkiksi tieto voidaan siirtää JSON-datana, jossa palvelimella oleva tieto haetaan API:n kautta (JSON, 2019; API, 2019).

Tällaisen Android-sovelluksen toteutus vaatii kaksi Java-luokkaa, toinen näistä on ns. pääluokka tai aktiviteetti niin kuin Android-sovelluksissa yleensä käytetään. Kutsutaan sitä, vaikka nimellä MainActivity.java. Toinen tiedosto olkoon HttpHandler.java, joka toimii tiedonhaun mahdollistavana ja haettavan tiedon käsittelijänä haluttuun muotoon. Lisäksi Android-sovellus tarvitsee AndroidManifest.xml tiedoston, jossa määritellään tarvittavat sovelluksen käyttämiseen ja näkymään liittyvät asetukset. Ohjelman käyttöliittymän määrittely tehdään activity_main.xml ja list_item.xml tiedostoissa.

Pääluokan luonnin alussa luodaan uusi objekti, joka laajentaa AsyncTask objektia (AsyncTask, 2019). Sen avulla voidaan helpolla tavalla ajaa säikeitä taustalla ja näyttää tulokset käyttöliittymässä. AsyncTask on tarkoitettu lyhyille käskyille, jollainen on myös JSON datan hakeminen palvelimelta. Siinä on yleensä neljä erivaihetta, onPreExecute, doInBackground, onProgressUpdate ja onPostExecute (AsyncTask, 2019). Ensimmäisessä vaiheessa onPreExecute luodaan ProgressDialog, jonka avulla voidaan informoida käyttöliittymän avulla käyttäjää odottamaan hetken, jotta voidaan seuraavassa doInBackground vaiheessa oleva data hakea HttpHandler apuluokan avulla. HttpHandler luokassa tehdään datan hakeminen palvelimelta ja konvertointi JSON objektista merkkijonoksi. Lopputulos palautetaan takaisin doInBackgroundiin. Datan perusteella

muutetaan data joko listaksi käyttäjän käyttöliittymään tai annetaan mahdolliset virhetiedot, jos sellaisia on datan hakemisen yhteydessä tullut viimeisessä onPostExecutessa vaiheessa.

Tässäkin toteutusvaihtoehdossa tarvitaan Internet-lupa, joka täytyy lisätä samaan tapaan AndroidManifest.xml tiedostoon kuin Webview-toteutuksessa. Lisäksi käyttöliittymän activity_main.xml tiedostoon lisätään käyttöliittymän tarvittavat asetukset. Tämän lisäksi list_item.xml määritellään ohjelman käyttöliittymä elementtien ominaisuudet, joihin mm. kuuluvat fontin tyyppi, koko ja väri sekä sijainti rivillä.

Kolmas mahdollinen toteutus on PWA-sovellus. Käytännössä se on ihan normaali web-sovellus tai web-sivu, mutta se on mahdollista asentaa puhelimen aloitusnäytölle samaan tapaan kuin Androidin natiivit-sovellukset. Ominaisuuksiin kuuluu, että PWA-sovellus toimii hyvin samaan tapaan kuin natiivit-sovellukset, sillä erotuksella että sen asentamiseen käyttäjän ei tarvitse mennä sovelluskauppaan, vaan riittää kun menee web-sivulle, jossa käyttäjälle annetaan ilmoitusikkuna mahdollisuudesta asentaa sovellus päätelaitteeseensa.

Lepage (2017) mukaan PWA-sovellus on progressiivinen eli toimii jokaisella selaimella, koska sen ydin on edistyksellinen. Hänen mukaansa sovelluksen käyttöliittymä pitää olla reagoiva eli toimia niin työpöydällä, matkapuhelimella kuin tabletilla. Lisäksi sovelluksen tulisi toimia myös ilman nettiyhteyttä tai huonosti toimivassa verkossa. Sovelluksen tulisi olla saman tuntuinen kuin niin sanottu natiivin sovelluksen. PWA-sovelluksen tulisi olla myös ajan tasalla eli sovelluksen toiminnallisuus ja sisältö tulisi erotella. Turvallisuus on myös yksi tekijä, sisältö välitetään salattuna HTTPS yhteyden yli. Löydettävyys on myös PWA-ohjelman yksi kriteereistä, hakukoneiden tulisi löytää ohjelman manifest.json ja service worker asennustiedostot. PWA-sovelluksen tulee olla uudelleen käytettävissä tai käynnistettävissä esimerkiksi palvelimelta tulevien push-ilmoitusten vastaanottamisen jälkeen. PWA-sovelluksen tulee olla asennettavissa laitteelle samaan tapaan kuin natiivien sovellusten. Lisäksi ohjelman osoite tulee olla jaettavissa ilman että sovellusta tarvitsee asentaa (LePage, 2017).

Käytännössä PWA-sovelluksessa tulee olla vähintään service-worker.js joka on PWA-sovelluksen perusta, Java-Script tiedosto (Gaunt, 2016). Kun selaimella tullaan web-sivulle, ensimmäisellä kerralla rekisteröidään service worker tiedosto (Biørn-Hansen, Majchrzak & Grønli, 2017). Service workerin implementointi käydään läpi tarkemmin palvelinohjelman arkkitehtuurissa. Gauntin mukaan toiminnaltaan service worker toimii taustalla ilman nettisivun tai käyttäjän vuorovaikutusta. Sen avulla voidaan toteuttaa taustalla tapahtuvat push-viestit ja datan synkronoinnit. Myös yhteydetön toiminnallisuus toteutetaan service workerin avulla (Gaunt, 2016).

PWA-sovellus tarvitsee myös manifest.json tiedoston, jossa Gauntin ja Kinlan (2019) mukaan määritellään ohjelman nimi, ikonit ja aloitusosoite. Näin voidaan saada selain näyttämään **lisää aloitusnäytölle**, josta käyttäjä voi aktivoida asennuksen niin halutessaan. Jos käyttäjä päättää lisätä PWA-sovelluksen omalle päätelaitteelleen, aukeaa uusi ikkuna. Siinä näytetään ohjelman ikoni, joka asennetaan käyttäjän aloitusnäyttöön. Samaiseen PWA-sovelluksen käynnistysikoniin näkyviin jää myös pieni Firefoxin kuvake, jos asennus on suoritettu Firefoxista käsin. Chromella asennettaessa PWA-sovelluksessa ei näy ylimääräisiä kuvakkeita sovelluksen käynnistysikonissa. Näin ollen kuvake on normaalin ja natiivin Android-sovelluksen kuvakkeen kaltainen ja sitä ei voi suorilta erottaa PWA-sovellukseksi.

Lisäksi manifestissä määritellään Gauntin ja Kinlanin (2019) mukaan PWA-sovelluksen taustaväri HEX-värikoodilla, jota käytetään myös sovelluksen käynnistyksen yhteydessä. Siinä näytetään sovelluksen ikoni ja sovelluksen nimi ikonin alapuolella. Lisäksi määritellään näyttöasetukset, joiden avulla PWA-sovelluksen käyttöliittymän tyyppi voidaan PWA-sovelluskehittäjän toimesta ennalta asettaa tietynlaiseksi. Esimerkiksi PWA-sovelluksen näyttötyypiksi voidaan määritellä standalone eli se mukailee mahdollisimman paljon natiivisovelluksen näköä ja tuntumaa. Manifestissä määritellään myös PWA-sovelluksen orientointi ja teeman värit HEX-värikoodilla. Teeman värikoodi muuttaa Android-päätelaitteissa yläpalkin määritellyn teeman HEX-värikoodin mukaiseksi myös PWA-sovelluksen ollessa käynnissä. PWA-sovelluksen yläpalkissa oleva väri voidaan muuttaa vihreäksi tai miksi tahansa väriksi, kunhan manifest.json tiedostossa on määrittely tehty. Tällä lisätään PWA-sovelluksen ominaisuuksia lähemmäksi natiivin Android-sovelluksen tuntumaa. Manifest.json implementointi käydään tarkemmin läpi palvelinohjelman arkkitehtuuriluvussa.

Gauntin (2016) mukaan PWA-sovelluksen asennusehtona on yhteyden HTTPS-salaus, tällä vahvistetaan käyttäjän luottamusta sovellukseen. Käyttäjä tietää sovellusta käyttäessään datan kulkevan salattuna sovelluksen ja palvelimen välillä. Samalla service worker voi toimia turvallisemmin, kun väliin ei niin helposti päästä muuntamaan tai suodattamaan tietoa. Service workerin rekisteröinti ja asennus tapahtuu, kun käyttäjä saapuu PWA-sovelluksen URL-osoitteeseen, jos vain selain tukee service workerin asennusta. Selain hoitaa omalta osaltaan service workerin rekisteröinnin ja sen implementointi käydään läpi tarkemmin palvelimen arkkitehtuuriluvussa.

Jos käyttäjä haluaa, pystyy hän asentamaan PWA-sovelluksen Android päätelaitteeseensa. Jos taas selain tai käyttäjä ei halua asentaa sovellusta, niin PWA-sovelluksen pitäisi pystyä toimimaan normaalin internetsivun tavoin. Kun service workerin asennus käynnistyy, ladataan yleensä välimuistiin tarvittavat tiedostot yhteydetöntä käyttöä varten. Asennus onnistuu vain, jos kaikki tiedostot saadaan ladattua (Gaunt, 2016). Yleensä välimuistiin ladataan ns. sovelluksen toiminnallisuuteen tarvittavat HTML, CSS ja JavaScript tiedostot. Tarkoitus on luoda käyttäjälle PWA-sovelluksen käynnistyksessä illuusio täysin toimintavalmiista sovelluksesta, vaikka varsinaista dataa oltaisiin vasta hakemassa palvelimelta tai käyttäjälle näytetään edellisellä kerralla haetusta välimuisti tiedoista koostuva näkymä. Kun PWA-sovelluksen service worker on hakenut uusimmat version palvelimelta, tulisi käyttäjää informoida tuoreemmasta datasta, jotta hän voi päivittää näkymän uudemmalla tiedolla. Toisaalta service workerin avulla voidaan ilmoittaa myös hänen internetyhteytensä toimimattomuudesta ja tarjota yhteydetön vaihtoehto nähtäväksi PWA-sovelluksessa.

2.3.2 Palvelinsovelluksen arkkitehtuuri

Palvelinsovelluksen arkkitehtuurissa käydään läpi mikä palvelinsovelluskehys on ja millä kielellä se tässä työssä toteutetaan. Esitellään mahdolliset vaihtoehdot palvelinsovellus kehykseksi. Lisäksi käydään läpi tarvittavat apusovellukset, jotta kokonaisuus on toimiva ja päivitettävissä erilaisille teknologioille.

Android-sovellus tarvitsee tiedon hakemiselle palvelinsovelluksen, joka tapahtuu yleensä palvelinsovellus kehyksen avustuksella ja sen tarkoitus on tukea verkkosovellusten ja API-rajapintojen toteutusta (Web_framework, 2020). Konstruktion vaatimusmäärittelyssä kehittäjä ja ylläpitäjä määrittelevät kehityskieleksi Pythonin. Se on

ohjelmointikieli, joka mahdollistaa työskentelyn nopeasti ja tehokkaan järjestelmien integroinnin (Python, 2020). Mahdollisia Python-pohjaisia palvelinsovelluskehyksiä on todella paljon. Valinta esimerkiksi Djangon, Bottlen, web2py ja Flask[3] välillä on ihan mieltymys siihen, miten palvelinsovellusta halutaan lähteä kehittämään. Koska vaihtoehtoja oli useampia, niin valitaan esimerkiksi web-kehitysalustaksi Flask (Flask, 2017), joka on BSD-lisenssin eli vapaa ohjelmistolisenssin alainen (BSD-Lisenssi 2017). Grinbergin (2018) mukaan Flask on ns. micro web-kehitysalusta, jossa on kaksi pääriippuvuusosaa. Ensimmäinen riippuvuus liittyy reititykseen, virheiden etsimiseen, ja web-palvelimen kommunikointiin web-sovelluksen kanssa ja Flask käyttää siihen Werkzeugia[4]. Toinen riippuvuus tulee Jinja 2[5] kaavakoneesta, jota käytetään web-sivujen renderöintiin (Grinberg, 2018). Muuten Flask antaa web-sovelluksen kehittäjälle vapaat kädet siitä miten hän haluaa web-sovelluksensa kehittää. Näin web-sovellus kehittäjää ei pakoteta käyttämään tiettyjä valmiita kirjastoja, vaan projektin tarpeitten mukaan pystyy valitsemaan sopivimmat laajennukset tai kehittämään omat (Grinberg, 2018).

```
.
├── cli_for_template_app.py
├── config.py
├── error.log
├── instance
│   └── config.py
├── Makefile
├── README.md
├── requirements.txt
├── run.py
└── template_app
    ├── api
    │   ├── __init__.py
    │   └── views.py
    ├── certs
    │   ├── private_key.txt
    │   ├── public_key.txt
    │   ├── vapid_private.pem
    │   └── vapid_public.pem
    ├── home
    │   ├── __init__.py
    │   └── views.py
    ├── __init__.py
    ├── redis.py
    ├── static
    │   ├── app.js
    │   ├── dark-mode-switch.min.js
    │   ├── favicon.ico
    │   ├── images
    │   │   └── success.jpg
    │   ├── main.js
    │   ├── manifest.json
    │   ├── offline.css
    │   ├── offline.html
    │   ├── script.js
    │   ├── service-worker.js
    │   ├── snack.js
    │   └── style.css
    └── templates
        ├── home
        │   ├── about.html
        │   ├── index.html
        │   ├── settings.html
        │   └── success.html
        └── layout.html

9 directories, 35 files
```

Kuva 3. Flask-sovelluksen arkkitehtuuri.

Kuvassa 3 oleva arkkitehtuuri on käytännön esimerkki Flask web-sovelluksen modulaarisuudesta ja laajennettavuudesta, joka on mahdollista toteuttaa sinikopioiden avulla (Blueprints, 2019). Kun sinikopio luodaan luomalla objekti luokasta Blueprint, se tarvitsee kaksi argumenttia, sinikopion nimen ja moduulin tai paketin sijainti hakemiston (Grinberg, 2019). Esimerkkinä tästä on, kun luodaan API:n sinikopiota, tehdään koodissa api- hakemistossa olevaan __init__.py tiedostoon seuraavat kohdat.

```
from flask import Blueprint

api = Blueprint('api', __name__)

from . import views
```

Itse ohjelmaan sinikopio rekisteröidään juurihakemistossa olevaan __init__.py tiedostoon lisäämällä loppuosaan tiedostoa ennen return app kohtaa.

```
from .api import api as api_blueprint
app.register_blueprint(api_blueprint)
```

Itse toiminnallisuus luodaan api-hakemistossa olevaan view.py tiedostoon. Siellä jokainen reitti sovelluksen näkymään muotoillaan seuraavan tavalla. Huomioitavaa on api reitti muotoilija(decorator), joka on muotoa `@api.route`. Tämä reitti muotoilija pitää muistaa nimetä oikein sinikopion mukaan. Muotoilijan avulla näkymä funktio rekisteröidään URL-osoitteelle (Flask, 2017).

```
@api.route('/api/v1/login/', methods=['GET', 'POST'])
def login_api():
    """Function for login via API."""
       # implementointi koodi tähän
    return jsonify(result)
```

Sinikopioiden kokonaisuuden käyttäminen Flask web-sovellukseen mahdollistaa myös erillisen Frontend-viitekehyksen käyttämisen, jos tarvetta tulevaisuudessa ilmenee sellaiselle (Blueprints, 2019). Frontend tarkoittaa tässä yhteydessä lähinnä käyttöliittymää sovellukselle. API:a voidaan käyttää myös Android- ohjelman konstruktion datalähteenä, joten se sopii myös erilliselle sovellukselle API-rajapinnaksi.

Static-hakemistossa on PWA-sovelluksen toiminnallisuuteen tarvittavat tiedostot ja sovelluksen web-näkymän tyylitiedostot. Jos halutaan käyttää muuta kuin Flaskin omaa Jinja2 ominaisuutta, Frontend-sovelluksen tiedostot voidaan lisätä static-kansioon. Templates-hakemistosta löytyy Flask sovelluksen Jinja2 kaavat, joiden avulla käyttöliittymään saadaan oikea toiminnallisuus. Periaatteessa Flask-sovelluksen käyttöliittymä on määritelty layout.html tiedostossa ja jonka sisään sitten esimerkiksi home-hakemistossa olevat index.html sisältö renderöidään, kun käyttäjä saapuu PWA-sovelluksen kotinäkymään (Flask, 2017).

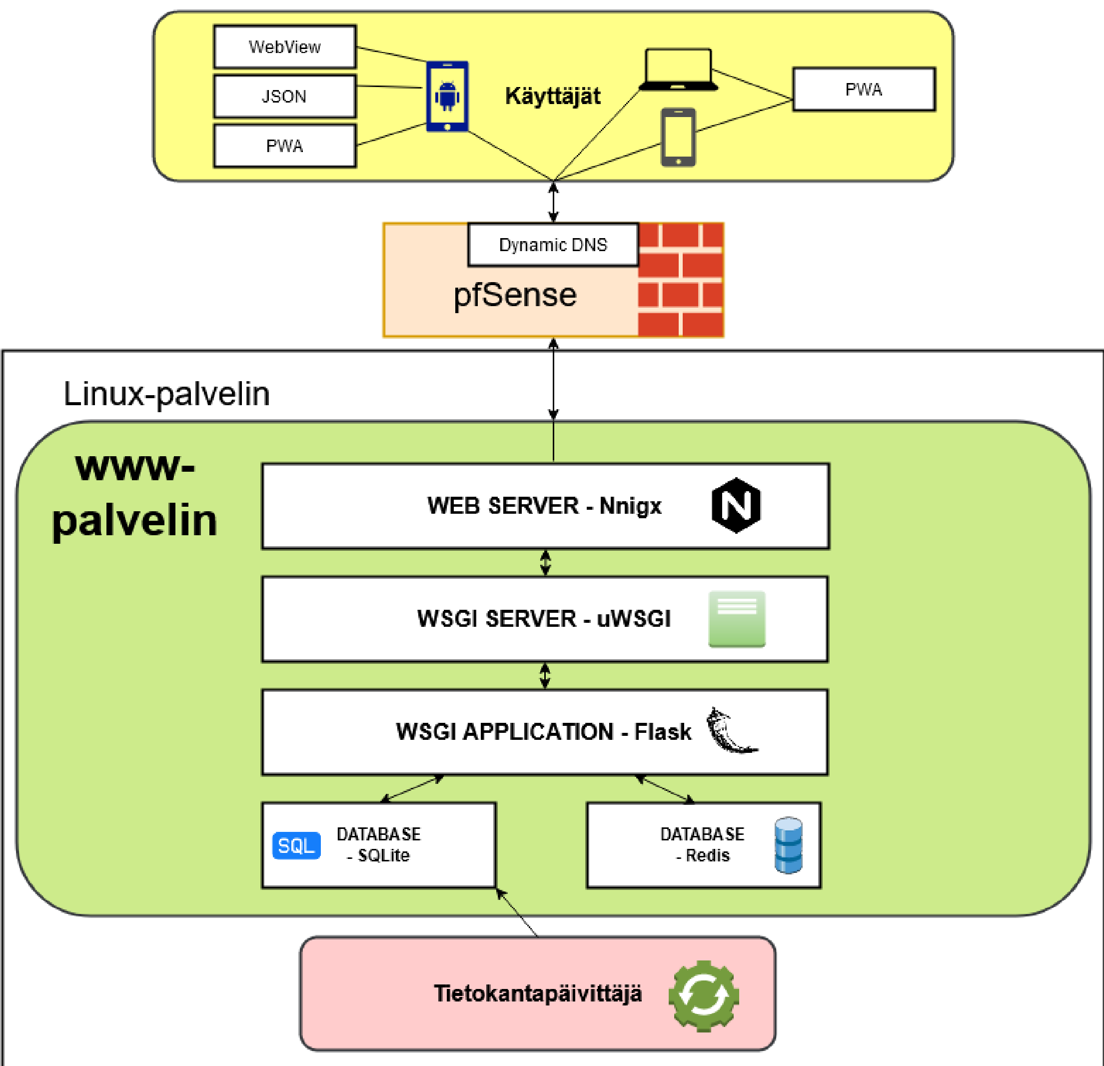
Flask-sovelluksen Python ja konfigurointitiedostot sijaitset joko juuri- tai template_app-hakemistossa. Config.py tiedostolla määritellään Flask sovelluksen käytön ja kehityksen asetuksia. Flask-sovelluksen tärkeimmät ominaisuudet ja käynnistys tapahtuu template_app-hakemistossa olevan __init__.py tiedoston avulla. Yleensä Flask-sovelluksen virtuaaliympäristö pitää joko asentaa ja aktivoida ennen kuin sopivat Python-kirjastot voidaan asentaa (Virtualenv 2019). Itse sovelluksen käynnistäminen voidaan toteuttaa monella tapaa, mutta tässä esimerkissä on käytetty Makefileä. Sen avulla kehittäjä voi käynnistää haluamansa konfiguraation helposti esimerkiksi

```
$ make run.development_api
```

käynnistäisi Flask-sovelluksen API-rajapinnan kehitysversiona. Ohjelman virhe- ja logi-tiedot tulevat error.log tiedostoon. Flask-ohjelman rakenne voidaan toteuttaa monella tapaa ja edellä esitelty toiminnallisuus on vain yksi mahdollinen ratkaisu arkkitehtuuriksi.

2.3.3 Teknologian arkkitehtuuri

Teknologian arkkitehtuuriin kuuluu useampia osa-alueita, kuten kuvasta 4 nähdään, Flask palvelinsovellus on vain osa kokonaisuutta. Lisäksi tarvitaan varsinainen www-palvelin ja alusta, jonka päällä www-palvelinkokonaisuus toimii. Kuvan 4 esimerkissä palvelimeksi on määritelty Linux, tarkemmin spesifioimatta toimintaympäristöä. WWW-palvelin vaihtoehtoja on useampia. Kolme suurinta isoimmasta pienimpään markkinaosuuksilla ovat Apache[6], Nginx[7] ja Microsoft-IIS[8].



Kuva 4. Teknologia-arkkitehtuuri.

Tässä tapauksessa valinta sattui Nginx, koska teknologia-arkkitehtuuri antoi osviitan sen sopivuudesta kehitettävään konstruktioon. Nginx tarvitsee SSL-sertifikaation ja sellainen on saatavilla mm. Let's Encryptiltä[9] (2019) ja sen avuksi tarvitaan Certbot[10] (2019), joka uusii sertifikaatin ajastettuna. Web-serverin ja WSGI[11] (Web Server Getaway Interface) -ohjelman väliin tarvitaan WSGI-palvelin mahdollistamaan HTTP-pyyntöjen välitys Flask-ohjelmalle. Kuvassa 4 esimerkkinä WSGI-palvelimesta käytetään uWSGI[12], mutta

vaihtoehtoisesti se voisi olla myös Gunicorn[13] (uWSGI, 2019). Lisäksi tarvitaan pari tietokantaa, joista SQLite[14] toimii datan säilömiseen, jota päivitetään tietokantapäivittäjällä automaattisesti cron ajastettuna tehtävänä. Tietokantapäivittäjä käyttää hyväkseen Sqlalchemy-kirjastoa[15] Sqlite-tietokannan kanssa kommunikoimiseen (Sqlalchemy, 2019). Tämän lisäksi Flask-ohjelma käyttää Redis-tietokantapalvelinta[16] avain – arvojen tallentamiseen, jos PWA-sovelluksen käyttäjä haluaa tilata Push-viestit päätelaitteelleen. Toinen vaihtoehto Push-viestien implementoimiseen on ulkoistaa se erilliselle palvelulle esimerkiksi OneSignal[17], jonka avulla voidaan lähettää Push-viestit päätelaitteelle (OneSignal, 2019). Kuvassa 4 oleva tietokantapäivittäjä on automatisoitu erillinen sovellus, joka vastaa tiedon hakemisesta ja päivittämisestä oikeaan muotoon tietokantaan. Teknologian arkkitehtuuri kuvassa näkyy myös PfSense-palomuuri[18], jossa on määritelty dynaaminen nimipalvelu. Se voi olla esimerkiksi Ducknds[19] (Duck DNS, 2019) jotta Flask-sovellukseen voidaan ottaa yhteyttä domain-nimen perusteella käyttäjät ryhmästä. Domain-nimen perusteella mahdolliset käyttäjät erilaisilla sovelluksillaan saavat haluamansa tiedon heidän erilaisiin päätelaitteisiinsa. Vaikka tässä työssä keskitytään Androidiin, kuvaan on lisätty myös mahdolliset muiden laitealustojen käyttäjät.

3. Tutkimusmenetelmät

Tässä luvussa esitellään konstruktiivinen tutkimusmenetelmä aikaisempaan tutkittuun tietoon perustuen. Kirjallisuuden perusteella määritellään, kuinka konstruktiivista tutkimusmenetelmää käytetään tietojärjestelmien kehittämisessä. Lopuksi esitellään konstruktiivisen tutkimuksen prosessikaavio ja siihen liittyvät osa-alueet.

Konstruktiivinen tutkimusmenetelmä tietojenkäsittelytieteissä pääpiirteittäin Nunamaker, Chen ja Purdinin (1990) tutkimuksen perusteella sisältää seuraavat kohdat: Ensimmäiseksi koostetaan käsitteellinen kehys, jossa on tarkoitus löytää merkityksellinen tutkimuskysymys, tutkia kehitettävän järjestelmän toiminta ja vaatimusmääritellä sekä, löytää ymmärrys siitä, mitä tulevalta järjestelmän toiminnoilta tarvitaan sekä merkitykselliset rajoitukset lähestymistavasta, millä ne toiminnot etsitään. Toisena listalla on arkkitehtuurin suunnittelu, jonka pitäisi olla yksinkertainen, mutta laajennettavissa oleva sekä modulaarinen. Kolmanneksi vaiheeksi he esittivät järjestelmän analysoinnin ja suunnittelun, missä tietokanta mallinnetaan ja järjestelmän toimintamalli kehitetään. Tässä vaiheessa voidaan myös tehdä useampia vaihtoehtoisia ratkaisuja, joista sitten valitaan yksi seuraavaan vaiheeseen. Neljänneksi rakennetaan prototyyppitietojärjestelmä ja samalla otetaan oppia tulevan järjestelmän kehityksen avuksi, jotta voidaan mahdollisista esiin tulevista ongelmista ja järjestelmän kompleksisuudesta löytää vaihtoehtoiset sekä toimivammat tavat toteuttaa lopullinen järjestelmä. Viides vaihe on kehitetyn järjestelmän havainnointi ja arviointi. Tämän pohjalta yritetään kehittää malli tai teoria, joka voitaisiin yleistää tai ainakin tehdään yhteenveto ja loppupäätelmät kehityksen tuloksista (Nunamaker ja muut, 1990.)

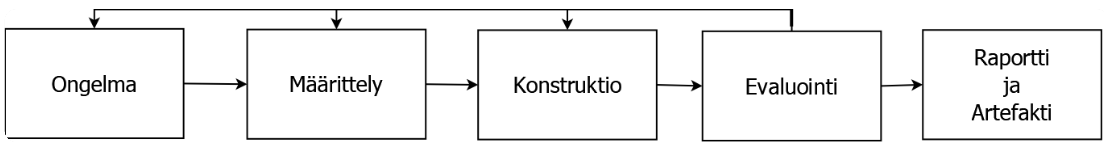
Kasanen, Lukka ja Siitonen (1993) määrittelivät konstruktiivisen tutkimuksen vaiheiksi seuraavat kuusi kohtaa. Ensiksi etsitään mahdollinen tutkimusongelma, joka on käytännöllinen sekä asiaankuuluva. Toiseksi hankitaan kokonaisvaltainen ymmärrys tutkimusaiheesta. Kolmanneksi kehitetään ja rakennetaan ratkaisu tutkimusongelmaan. Neljännessä vaiheessa esitellään ratkaisun toiminta ja toiminnallisuus. Viidenneksi esitellään teoreettinen yhteys ja mahdolliset löydökset ratkaisun pohjalta. Viimeiseksi tutkitaan ratkaisun soveltuvuutta.

Hevner, March, Park ja Ram esittelivät vuonna 2004 seitsemänkohtaiset ohjesäännöt. Ensimmäinen ohjesääntö on, että suunnitellaan artefakti eli tutkimuksen pitää tuottaa jokin mahdollinen toteutus, malli, metodi tai ilmentymä. Toinen ohjesääntö on ongelman merkityksellisyys eli tuottaa teknologiapohjainen ratkaisu johonkin tärkeään ja merkitykselliseen ongelmaan. Kolmas ohjesääntö on suunnitelman evaluointi eli suunniteltu artefakti tulee olla järjestelmällisesti arvioitu. Neljäs ohjesääntö on tutkimuksen kontribuutio eli tutkimuksen tulee tuottaa selvä ja todennettava artikkeli. Viides ohjesääntö on tutkimuksen täsmällisyys eli menetelmän rakentaminen ja arviointi luottavat täsmällisesti metodeihin. Kuudenteen ohjesääntöön kuuluu, että tutkimus suunnitellaan kuin hakuprosessi. Jotta saavutetaan tehokas artefakti, tarvitsee käyttää kaikki mahdolliset keinot sen löytämiseen. Seitsemäs ohjesääntö on tutkimuksen sanoma eli tutkimus pitää esitellä jokaiselle siihen kuuluvalle kohderyhmälle. (Hevner ja muut, 2004.)

Peffers, Tuunanen, Rothenberger ja Chatterjee (2007) esittelivät oman menetelmäoppinsa siitä, miten konstruktiivinen tutkimus tietojärjestelmien kehittämisessä tulisi toteuttaa. Heidän tutkimusprosessinsa oli kuusivaiheinen. Ensimmäisen on ongelma ja sen määrittely sekä motivaatio. Toisessa vaiheessa määritellään ratkaisun tavoite. Kolmannessa vaiheessa suunnitellaan ja kehitetään artefakti. Neljännessä vaiheessa

havainnollistetaan, kuinka suunniteltu artefakti ratkaisee aikaisemmin määritellyn ongelman. Viidennessä vaiheessa evaluoidaan, kuinka tehokkaasti toteutus toimii. Kuudennessa vaiheessa esitellään ja julkaistaan tutkimuksen tulokset.

Aikaisemmin esitellyiden tutkimusten perusteella konstruktiivisessa tutkimuksessa ensimmäinen ja tärkein tutkimuksen liikkeelle paneva voima on ongelma, joka täytyy ratkaista. Tuleva artefakti pitää seuraavaksi jotenkin määritellä, jotta voidaan suunnitella ja kehittää ratkaisu ongelmaan. Seuraavaksi luotu ratkaisu esitellään ja sitä arvioidaan sekä mahdollisesti iteroidaan. Lopuksi kokonaisuus täytyy tuoda toisten tutkijoiden ja kohderyhmien saataville, jotta artefaktia voidaan kehittää ja menetelmää testata toisessakin toimintaympäristössä. Konstruktion prosessikaavio on iteratiivinen. (Kuva 5).



Kuva 5. Konstruktion prosessikaavio.

Konstruktiivisessa tutkimuksessa lähtökohtana on ongelma, joka pitää jollain keinolla ratkaista. Nunamaker ja kumppanit (1990) totesivat, että ensiksi pitää rakentaa käsitteellinen kehys. Heidän mukaansa siihen kuuluu mm. merkityksellinen tutkimuskysymys, selvittää järjestelmän toiminnallisuudet ja vaatimukset. Lisäksi täytyy löytää ymmärrys miten tuleva järjestelmä tulisi rakentaa ja tutkia merkitykselliset rajoitteet uusille ideoille ja lähestymistavoille. Kasanen ja muut (1993) totesivat, että on löydettävä käytännönläheinen ongelma, jossa on myös mahdollista tutkimuksen tekeminen. Hevner ja kumppanit (2004) määrittelivät ongelman relevanssia, mistä esimerkkinä he esittivät, että ongelma voidaan määritellä nykyisen ja kehityksen alaisen järjestelmän eroina. Heidän mukaansa päämääränä on parantaa olemassa olevaa ja näin luoda tehokkaampi keino toteuttaa se. Peffersin ja kumppanien (2007) näkemys konstruktiivisen tutkimuksen ensimmäisenä vaiheena on ongelman määrittely ja se, millainen merkitys ongelman ratkaisulla voidaan saavuttaa. Näin ollen voidaan perustella ratkaisun toteutus.

Konstruktiivisessa tutkimuksessa Nunamaker ja kumppanit (1990) määrittelivät vaatimusmäärittelyn seuraavasti, vaatimukset pitäisi tarkentaa niin, että ne ovat mitattavissa ja näin ollen niiden toteutus voidaan arvioida myöhemmässä evaluointivaiheessa. Hevner ja kumppanit (2004) totesivat liiketoiminnan ympäristön muodostavan vaatimukset, jonka pohjalta voidaan kehitettävä artefakti evaluoida. Heidän mielestään suunniteltu artefakti on valmis ja tuloksia tuottava, kun se täyttää järjestelmän vaatimukset ja rajoitteet, mitkä ongelmaratkaisun on tarkoitus ratkaista. Peffers ja kumppanit (2007) totesivat konstruktiivisen tutkimuksen tukevan vaatimusmäärittelyä ja tuottavan suuntaviivat kehitettävälle artefaktille. Heidän mielestään sitä voidaan käyttää mm. seuraavissa tilanteissa: organisaatiota tarkasteltaessa, tiedonkeruussa, mallintamisessa ja vaatimusmäärittelyspefikaatiota laatiessa. Siitä miten tai mitä vaatimusmäärittely täytyy toteuttaa, he eivät kertoneet muuta kuin sen kuuluvuutta prosessiin.

Nunamaker ja kumppanit (1990) totesivat kehitettävän järjestelmän konstruktiosta konstruktiivisessa tutkimuksessa seuraavaa: ensimmäiseksi kehitetään tulevan järjestelmän arkkitehtuuri. Sen suunnittelussa tulisi ottaa huomioon mm. modulaarisuus

ja laajennettavuus. Lisäksi tulisi määritellä kehitettävän järjestelmän komponentit ja niiden väliset yhteydet. Toisena konstruktion vaiheena he esittivät analysoinnin ja suunnittelun tulevalle järjestelmälle. Tähän vaiheeseen kuuluvat heidän mukaansa tietokantojen mallit ja järjestelmän funktiot. Lisäksi tähän vaiheeseen kuuluvat mahdolliset vaihtoehtoiset ratkaisut ja niiden pohjalta valitaan yksi toteutettavaksi. Konstruktion kolmantena vaiheena he esittivät prototyypin toteuttamisen, jonka avulla voidaan oppia konseptin, kehyksen ja suunnitelman toteutus. Prototyypillä voidaan löytää mahdolliset ongelma-alueet toteutuksessa ja järjestelmän kompleksisuudessa. Kasanen ja kumppanit (1993) totesivat konstruktiovaiheesta seuraavaa: Tulee keksiä, toisin sanoen rakentaa ratkaisu ongelman poistamiseen. Tämä vaihe on heidän mukaansa heuristinen ja yksi keskeisin elementti onnistuneelle konstruktiiviselle tutkimukselle, jos löydetään uusi mahdollinen toteutus, jolla ongelma voidaan ratkaista. Hevner ja kumppanit (2004) totesivat konstruktiosta, että se on luonteeltaan iteratiivinen hakuprosessi, jossa luodaan artefakti ja mikä voi olla tyypiltään malli, metodi tai ilmentymä. Heidän mukaansa konstruktion tarkoitus on osoittaa implementointi järjestelmälle toimintaympäristössään, joka ratkaisee tärkeän aikaisemmin ratkaisemattoman ongelman. Peffers ja kumppanit (2007) totesivat konstruktion vaiheiksi suunnittelun ja kehityksen. Tällöin suunnitelma muutetaan kehityksen avulla ratkaisuksi ongelmakehyksessä esitettyyn ongelmaan. Tämän lisäksi heidän näkemyksensä mukaan konstruktioon kuuluu myös demonstroida toteutetun artefaktin kyky ratkaista ongelma. Tämän todentaminen voidaan todentaa mm. simuloimalla, tapaustutkimuksella tai kokeellisella aktiviteetilla. Lisäksi demonstrointi voidaan toteuttaa käytännön esimerkillä, miten artefakti voi ratkaista aikaisemmin esitetyn ongelman tehokkaasti.

Nunamaker ja kumppanit (1990) määrittelivät evaluoinnin seuraavasti: Kun järjestelmä on rakennettu ja toteutettu, sen tehokkuus ja käytettävyys voidaan testata niitä määreitä vastaan, mitä on vaatimusmäärittelyssä esitelty. Heidän mukaansa pitää myös tarkastella kehitetyn järjestelmän vaikutuksia yksilön, ryhmän ja organisaation tasolla. Niiden tulokset pitää tulkita vaatimusmäärittelyä vastaan, jonka pohjalta voidaan järjestelmää jatkokehittää edelleen. Kasanen ja kumppanit (1993) totesivat evaluoinnista, että kehitettyä järjestelmää tulee tutkia ratkaisun soveltuvuuden mukaan. Käytännönläheinen käytettävyys on yksi merkittävistä tekijöistä, mikä näyttää todenmukaisesti kehitetyn konstruktion. Tämän pohjalta voidaan tarkastella yleistyksiä ja samankaltaisuuksia, joista voidaan koostaa yhtenäisiä ja näkyviä ohjeistuksia vastaaville konstruktiolle. Hevner ja kumppanit (2004) määrittelivät artefaktin evaluoinnin tapahtuvan laskennallisesti tai matemaattisesti, kun tarkastellaan toteutuksen laatua ja tehokkuutta sekä vaikutusta. Kuitenkin he totesivat, että myös empiirisiä tekniikoita voidaan käyttää evaluoinnissa hyväksi. Kehitetyn artefaktin evaluointi voidaan todentaa toiminnallisuuden, johdonmukaisuuden, virheettömyyden, suorituskyvyn, toimintavarmuuden, käytettävyyden, organisaatioon sopivuuden ja jonkin olennaisen ominaispiirteen perusteella. Heidän mukaansa suunniteltu artefakti on valmis ja ratkaisun tuottava, kun vaatimusmääritellyt artefaktin suunnitteluvaatimukset täyttyvät, sekä ongelman ratkaisulle asetetut rajoitteet toteutuvat.

Hevner ja kumppanit (2004) listaavat 5 erillistä evaluointimenetelmää. Ensimmäisenä kohtana he esittävät havainnoinnin tai tarkkailun esimerkiksi tapaustutkimuksen tai kenttätutkimuksen avulla. Toisena menetelmänä analyyttisen evaluoinnin esimerkiksi staattisen, dynaamisen tai arkkitehtuaalisen analyysin avulla. Samaan kategoriaan he sisällyttivät myös optimointi analyysin. Kolmantena kohtana Hevner ja kumppanit (2004) esittivät kokeellisen evaluoinnin, johon lukeutuvat esimerkiksi käytettävyys testaus ja simulointi. Neljäntenä kohtana he esittivät testausta, kuten esimerkiksi funktionaalista tai rakenteellista testausta. Viidentenä evaluointimenetelmänä he esittivät kuvainnollisuuden esimerkiksi skenaarioitten avulla. Peffers ja kumppanit (2007) totesivat evaluoinnista

seuraavasti: Artefaktia tulee tarkkailla ja mitata kuinka hyvin sen avulla toteutettu järjestelmä tukee ongelmanratkaisua. Ongelmasta ja sen tyypistä riippuen, sitä voidaan esimerkiksi vertailla vaatimusmäärittelyssä esille tulleisiin vaatimuksiin. Kehitettyä järjestelmää voidaan evaluoida monella tapaa. Heidän mukaansa evaluointia voidaan toteuttaa esimerkiksi suorittamalla tyytyväisyyskysely kehitetystä järjestelmästä, asiakaspalautteina tai simulointeina. He myös esittivät, että järjestelmää voidaan evaluoida mitattavilla määreillä kuten vasteajoilla tai saatavuudella. Peffersin ja kumppaneitten (2007) mukaan artefaktia voidaan evaluoida myös käsitteellisesti, mikä voi pitää sisällään loogista todistusta tai täsmällistä kokemusperäistä näyttöä. Heidän mukaansa kehittäjän pitää evaluoinnin perusteella päättää, jatkaako artefaktin kehittämistä parantaakseen sen tehokkuutta vai siirtyykö seuraavaan vaiheeseen. Kehittäjä kertoo kehitetyn järjestelmän toteutuksesta ja oppimastaan muille.

Nunamaker ja kumppanit (1990) eivät tutkimuksessa määrittele tarkasti, miten raportoida artefaktista. Heidän lähestymisensä on, että kehitettyä artefaktia tutkitaan ja evaluoidaan, minkä pohjalta voidaan rakentaa uusia artefaktia. Lisäksi yhdistetään tämän iteratiivisen prosessin opit kokonaisuudeksi. Myös Hevner ja kumppanit (2004) totesivat iteratiivisyyden olevan osa konstruktiivisessa tutkimuksessa. Kasanen ja kumppanit (1993) ohjeistivat osoittamaan teoreettisen tutkimuksen ja ratkaisun yhteydet. Tämän lisäksi tulisi tutkia ratkaisun mittakaavaa. Hevner ja kumppanit (2004) totesivat, että tutkimus pitää esitellä sellaisella tarkkuudella, jotta se voidaan sen pohjalta implementoida toisaalle ja osoittaa myös artefaktin toimivuus. Lisäksi raportin tarkkuus pitäisi mahdollistaa konstruktiivisen tutkimuksen toteutuksen toistamisen tietojärjestelmien toteutuksessa, jolla tutkimuksen tietopohjaa voidaan laventaa tai tarkentaa entisestään. Peffers ja kumppanit (2007) esittävät viimeiseksi vaiheeksi kommunikoinnin. Sillä he tarkoittavat sitä, miten tutkimuksen tulos tulisi raportoida. Heidän ehdotuksensa on raportoida yleisellä tutkimustyylillä, missä on ongelman määrittely, aikaisemman tiedon esittely, tutkimuskysymys, tiedon keruu, analysointi, tulokset ja keskustelu sekä johtopäätökset.

Konstruktiivisen tutkimusmenetelmän kokonaisuus ja iteratiivisuus näkyy kuvasta 5, se miten tutkimusmenetelmän toteutus eri osa-alueittain toteutetaan, on tutkijan itsensä määriteltävä. Kunhan tutkimuksesta löytyy kuvan 5 osa-alueet, tämä antaa perustan minkä perusteella tutkija voi lähestyä ongelmaa, tutkittavaa artefaktin kehitystä ja toteutusta sekä evaluointia. Näiden pohjalta tutkija toteuttaa tutkimuksen ja raportoi sen.

4. Ongelmakehys

Tässä luvussa esitellään järjestelmän kehittämisen liikkeelle sysäävä ongelma, joka halutaan ratkaista. Konstruktiivisen tutkimuksen kirjallisuuden perusteella hyödynnetään Nunamakerin ja kumppaneitten (1990) esittelemää käsitteellistä kehystä. Tässä ongelmakehyksessä koostetaan kehitettävän järjestelmän toimintaa. Ensiksi esitellään tutkimuskysymykset. Seuraavaksi kuvaillaan ongelma ja olemassa oleva järjestelmä. Lopuksi esitellään tulevaa järjestelmää, jonka avulla ongelma pyritään ratkaisemaan.

4.1 Tutkimuskysymyksien esittely

Nunamakerin ja kumppaneitten (1990) mukaan konstruktiivisen tutkimuksen lähtökohtana on esitellä merkitykselliset tutkimuskysymykset alettaessa kehittämään tietojärjestelmää. Konstruktion prosessikaaviossa (kuva 5) ensimmäisenä kohtana on ongelma, joka pyritään ratkaisemaan. Tässä työssä tutkittiin seuraavia tutkimuskysymyksiä.

TK 1: *Sopiiko konstruktiivinen tutkimusmenetelmä Android-ohjelma suunnitteluun ja määrittelyyn sekä toteutukseen?*

TK 2: *Miten saadaan olemassa oleva internetsivun tai RSS-syötteen sisältö tietyin reunaehdoin mobiilisovellukselle käytettäväksi ja kehitetty järjestelmä toimisi automaattisesti?*

Ensimmäisenä tutkimuskysymyksen tavoite oli löytää vastaus konstruktiivisen tutkimusmenetelmän sopivuudesta Android ohjelman toteutukseen. Tällä haluttiin saada vastaus tutkimusmenetelmän sopivuudesta tutkittavaan kohteeseen. Toisena tutkimus kysymyksenä oli kohde, joka haluttiin ratkaista hyödyntäen menetelmää.

4.2 Ongelman esittely

Käyttäjä haluaa saada tarvitsemansa tiedon Android-laitteella helposti yhdellä kosketuksella tai automatisoidusti ilmoituksena käyttöjärjestelmän ilmoituspalkkiin. Nykyisen järjestelmän käyttö vaatii ensiksi verkkoselaimen aukaisun, pääsivun osoitteen kirjoittamisen ja sivustolle siirtymisen. Tämän jälkeen käyttäjä joutuu navigoimaan valikoista haettavan tiedon yläsivulle, jossa sivun listalta valitaan haluttu tieto ja käyttäjä saa haluamaansa tiedon, kunhan vain sivu latautuu. Vaihtoehtoisesti käyttäjä aukaisee verkkoselaimen ja etsii kirjanmerkeistä haluamallensa listasivulle ja näin pääsee tietoon hiukan nopeammin käsiksi. Kokonaisuudessa tähän kuluu käyttäjällä aikaa ja hänen on monen välivaiheen kautta etsittävä tarvitsemansa tieto.

Tulevassa järjestelmässä käyttäjän tarvitsema tieto tulee olla nopeammin ja helpommin saatavilla. Näin ollen tulevan järjestelmän pitää pystyä kokonaisuudessaan toimia jouhevasti ja antaa käyttäjälle hänen hakemansa tieto suoraan ohjelman käynnistämisen jälkeen tai automaattisesti Android-laitteen ilmoituspalkkiin ilmoituksena. Järjestelmän kokonaisuuden suunnittelusta pyritään tekemään tietoturvallinen ja tiedon saatavuus sekä oikeellisuus varmistetaan.

4.3 Olemassa olevan järjestelmän esittely

Nykyinen järjestelmä on www-sivusto, jossa käyttäjälle on tarjolla pääsivu ja useita aihealueita ja niiden alasivuja. Järjestelmä on toimiva, mutta kokonaisuutta ei ole suunniteltu mobiilille päätelaitteelle. Nykyinen järjestelmä skaalautuu kohtuullisen hyvin mobiilille näytölle, mutta kuvakkeet ja linkit ovat pienehköjä ja sormella koskettaessa virhepainallusten mahdollisuus kasvaa.

Järjestelmä tarjoaa mahdollisuuden päästä käsiksi RSS-syötteeseen, josta tarvittava tieto löytyy. RSS-syöte on tiedostomuodoltaan XML-tiedosto. Sen sisältö on puumainen hierarkia, josta löytyy tarvittavat tiedot. Näitä ovat listojen otsikot ja niihin liittyvä http-osoite. Lisäksi RSS-syötteestä löytyy listan laatija, luonti- ja päivitysajat.

4.4 Tulevan järjestelmän esittely

Tuleva järjestelmä hakee olemassa olevasta järjestelmästä käyttäjän tarvitseman tiedon ja näyttää sen Android-laitteessa, joko Android-sovelluksessa tai ilmoituspalkkiin valittuna aikana tulevana ilmoituksena. Kokonaisuuden tulee toimia mahdollisemman automaattisesti ilman, että käyttäjän tarvitsee etsiä tietoa enempää kuin käynnistää sovelluksen, jonka jälkeen hänelle avautuu suoraan hänen tarvitsemansa tieto. Tulevan järjestelmän tulee olla laajennettavissa tai tuettava laajentamista helposti myös toisille mobiileille alustoille, kuten iOS ja Windows OS. Järjestelmän pitää olla tietoturvaltaan ja yhteensopivuudeltaan sellainen, että käyttäjälle ja hänen yksityisyydellensä ei aiheudu uhkia tai niitä on mahdollisimman vähän. Järjestelmässä ei tule olemaan mainoksia tai mitään muutakaan, jonka vuoksi käyttäjän ja ohjelman asetuksia jouduttaisiin toteuttamaan siten, että käyttäjästä tarvitsee tietää mahdollisimman paljon. Käyttöliittymä suunnitellaan niin, että käyttö kosketusnäytön kautta on sujuvaa ja informaatio luettavaa.

Tulevan järjestelmän toteutusvaihtoehtoja on useita erilaisia, ensimmäinen ja yksikertaisin niistä on, että käyttäjä lisää haluamansa internetsivun Android-laitteensa WWW-selaimen kirjanmerkiksi. Ratkaisu on kyllä toimiva, mutta se ei suoraan tarjoa käyttäjälle hänen tarvitsemaansa informaatiota. Toinen vaihtoehto on kehittää natiivi Android-sovellus, joka prosessoi RSS-syötteestä päivittäin vaihtuvan informaation. Ratkaisu toimisi hyvin Android-alustalla, mutta laajennettavuus muille käyttöjärjestelmille on mahdoton. Kolmas vaihtoehto on hyvin samankaltainen kuin yllä, mutta RSS-syötteen prosessointi tehdään erillisellä palvelimella. Sitten tämä tieto haetaan palvelimelta Android-sovellukseen ja näytetään käyttäjälle. Tässä ratkaisussa laajennettavuus muille alustoille on mahdollinen, mutta vaatii natiivin ohjelman kehittämisen jokaiselle alustalle erikseen, mikä lisäisi ylläpidon vaativuutta. Neljäs ratkaisu on käsitellä RSS-syöte erillisellä palvelimella ja louhia tarvittava tieto siitä. Tämän jälkeen tiedosta muodostetaan www-sivu, jossa päivittäin vaihtuva informaatio on heti saatavilla. Tämän tiedon näyttäminen Android-laitteessa tapahtuisi natiivin ohjelman webviewin kautta. Tällöin käyttäjä sovelluskuvaketta painamalla pääsee käsiksi informaatioon nopeasti tarvittaessa. Laajennettavuus toisille järjestelmille vaatisi jokaiselle joko oman ohjelmansa tai olisi heti käytettävissä www-selaimen kautta. Lisäksi tieto olisi saatavilla tietokoneillakin. On myös mahdollista toteuttaa sovellus PWA-sovelluksena, joka tukee kaikkia edelle esiteltyjä käyttö skenaariota.

5. Konstruktion vaatimusmäärittely

Tässä luvussa esitellään konstruktion vaatimusmäärittely. Tässä tapauksessa käytettiin ketteristä menetelmistä tuttuja käyttäjäkertomuksia kuvaamaan kehitettävän artefaktin vaatimusmäärittelyä, koska iterointi on osana molemmissa sekä konstruktiivisessa tutkimuksessa että ketterissä menetelmissä. Vaatimusten tarkentuminen projektin edetessä, ymmärryksen kasvaminen ja toteutuksen tekeminen vaikuttivat lopputulokseen.

Ensiksi määriteltiin vaatimukset konstruktiivisesti kehitettävälle Android-sovellukselle. Vaatimukset esiteltiin parin käyttäjäkertomus esimerkin avulla ja ne kirjoitettiin indeksikorteille. Samoin tehtiin palvelinsovelluksen vaatimuksille. Lopuksi määriteltiin teknologian vaatimuksia, joista luotiin pari esimerkkiä ei-funktionaalisina käyttäjäkertomuksina.

5.1 Android-sovellus

Android-sovelluksen vaatimusmäärittely käyttäen käyttäjäkertomuksia eteni seuraavasti. Tulevan järjestelmän käyttäjä kirjoitti: ”Haluan tietää, mitä ruokaa on tarjolla koululla.” Sen sisältö lisätään indeksikorttiin kuvassa 6.

[6.] Käyttäjä hakee ruokalistan.

Hyväksyntä testi: ruokalista testimetodi

Prioriteetti: 1

Kertomus pisteet: 5

Opiskelija haluaa saada tietää, mitä ruokaa ruokalassa on tarjolla.

Kuva 6. Indeksikortti "käyttäjä".

Koska kuvassa 6 esitetty käyttäjäkertomus osoittautui liian korkeantason kuvaukseksi, voitiin käyttäjältä kysyä tarkennuksia, siitä mitä hän oli tarkoittanut. Tällöin käyttäjä kirjoitti: ”Android puhelimen käyttäjänä haluan saada tämän päivän ruokalistan puhelimen ruudulle nähtäväksi, jotta tiedän mitä ruokaa siellä on tänään tarjolla.” Tämä sama käyttäjäkertomus näkyy kuvasta 7. Samalla tavalla tehtiin muillekin korkeantason kuvauksille, jos ne olivat liian yleisiä kuvauksia implementoinnin kannalta.

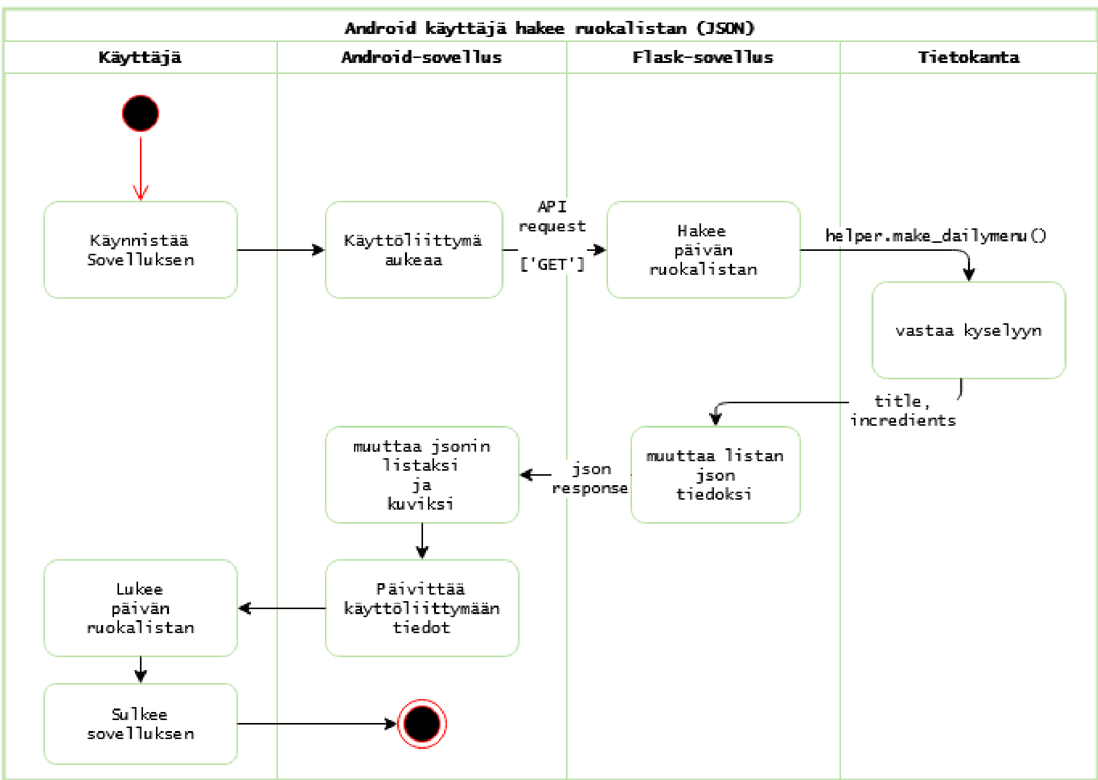
[7.]	Android käyttäjä hakee ruokalistan.	
Hyväksyntä testi: Android ruokalista testimetodi	Prioriteetti: 1	Kertomus pisteet: 5
Android puhelimen käyttäjä haluaa saada oman koulun ruokalan ruokalistan puhelimeen, jotta tietää mitä ruokaa siellä on tarjolla.		

Kuva 7. Indeksikortti ”Android käyttäjä”.

Kun haluttiin tarkentaa käyttäjäkertomuksia niin samalla tarkennetaan varmennus mihin määriteltiin vaatimuksien hyväksyntään liittyvät kohdat. Hyväksyntätesti kuvassa 7 voitiin kirjoittaa muotoon **Annettu** *<Android puhelimen käyttäjällä on ruokalista sovellus>* **Milloin** *<Hän päättä käynnistää sen>* **Sitten** *<Hän näkee tämän päivän ruokalistan>*. Samainen hyväksyntätesti voitiin toteuttaa Robot Frameworkin avulla android_foodlist.robot -testiksi.

```
*** Test Cases ***
Android user can get food list
    Given a user has an Android food list application
    When she decides open it
    Then she can see today’s food list
```

Kun käyttäjäkertomuksen toteutusta mietitään, voidaan se mallintaa vuokaaviolla tai aktiviteettikaaviolla. Kuvasta 8 nähdään esimerkki aktiviteettikaaviosta Android JSON toteutuksesta. Siinä esitettiin kuvan 7 Android käyttäjä hakee ruokalistan indeksikortti aktiviteettikaaviona.



Kuva 8. Aktiviteettikaavio ”Android-sovellus (JSON)”.

Kuten kuvasta 8 nähdään, käyttäjän käynnistäessä Android-sovellusta, kommunikoi se Flask-sovelluksen API-rajapinnan kanssa. Kun käyttöliittymä aukesi, niin Android-sovellus otti HTTP ’GET’-pyynnön Flask-sovelluksen API-rajapintaan. Tällöin Flask-sovelluksessa helper – apufunktion avulla haettiin tietokannasta päivän ruokalistan otsikko ja ruokatiedot. Jos tietoa ei löytynyt tietokannasta, palautettiin ”ei saatavilla” viesti. Tämän jälkeen Flask-sovellus muutti tiedon JSON-vastaukseksi Android-sovellukselle. Kun Android-sovellus sai vastauksen Flask-sovelluksesta, se muutti vastaan otetun JSON-tiedon oikeaan muotoon käyttöliittymään ja päivitti näkymän. Näin käyttäjä sai joko päivän ruokalistan tai virheestä kertovan näkymän Android-sovelluksen käyttöliittymään. Tiedon saatuaan käyttäjä sulki Android-sovelluksen.

5.2 Palvelinsovellus

Vaatimusmäärittely palvelinsovellukselle ja sen mahdolliselle toteutukselle voidaan esittää myös käyttäjäkertomuksena. Tulevan järjestelmän ylläpitäjä kirjoitti: ”Järjestelmän ylläpitäjänä haluan saada vakavan virheen sattuessa virheraportin järjestelmästä, jotta voin tarkistaa ja reagoida siihen.” Kuvaus lisätään indeksikorttiin ja kuvasta 9 se voidaan nähdä kokonaisuutena.

[9.]	Ylläpitäjä virheraportti	
Hyväksyntä testi: admin_error_report testimetodi	Prioriteetti: 1	Kertomus pisteet: 2
Järjestelmän ylläpitäjä haluaa saada vakavan virheen sattuessa virheraportin järjestelmästä, jotta voi tarkistaa ja reagoida siihen.		

Kuva 9. Indeksikortti "Ylläpitäjä"

Edellä kuvatun järjestelmän ylläpitäjän käyttäjäkertomuksen kokonaisuus ja toteutus ovat vielä tässä vaiheessa korkeatason kuvauksia. Ne eivät välttämättä suoraan ole Android-sovellukseen liittyviä, mutta ilman palvelinta järjestelmä ei ole toimiva. Prioriteetti ja tärkeys kehitettävälle Android-sovelluksen osalle on siten korkea.

Tämäkin kuvan 9 käyttäjäkertomus voidaan tarkentaa ja mallintaa tarkemmin indeksikortilla (kuva 10).

[10.]	Ylläpitäjä saa virheraportin, jos web-sovellus ei ole saatavilla.	
Hyväksyntä testi: web_app_report testimetodi	Prioriteetti: 1	Kertomus pisteet: 2
Järjestelmän ylläpitäjä haluaa saada ilmoituksen itselleen järjestelmästä tai järjestelmää valvovasta sovelluksesta, jos web-sovellus ei ole saatavilla 10 minuuttiin.		

Kuva 10. Indeksikortti "Ylläpitäjälle virheraportti".

Hyväksyntätestin avulla voitiin myöhemmin evaluoinnissa nähdä, toteutuiko siinä määritellyt kohdat.

5.3 Teknologiamäärittely

Android-sovellus ohjelmoidaan Javalla ja pyritään tukemaan mahdollisimman kattavaa joukkoa Android-laitteista. Sovelluksen palvelimen ohjelmointikielenä käytetään Pythonia ja web-kehyksenä sille toimii Flask. Nämäkin voidaan kirjoittaa käyttäjäkertomuksena. Esimerkiksi järjestelmän ylläpitäjänä haluan palvelimen tukevan

Python-ohjelmointikieltä, jotta tarvittaessa voin ohjelmoida lisäosia ja ominaisuuksia itse.

[11.]	Palvelimen ohjelmointikieli.	
Hyväksyntä testi: Python 3.x tuki	Prioriteetti: 1	Kertomus pisteet: 1
Järjestelmän ylläpitäjä haluaa itse ohjelmoida ominaisuuksia, jotta järjestelmän ominaisuudet laajenisivat.		

Kuva 11. Indeksikortti "Palvelimen ohjelmointikieli".

Kuvasta 11 voidaan nähdä ei-funktionaalisen vaatimusmääritellyn toteutus indeksikortilla. Tällaiset ei-funktionaaliset vaatimukset ovat hyvin tarpeellisia, kun konstruktiota aletaan toteuttamaan. Sillä voidaan rajata tai rajoittaa mahdollisten vaihtoehtojen lukumäärää, kun valitaan sopivaa web-kehystä palvelimen alustaksi.

6. Konstruktio

Tässä luvussa esitellään kehitettävä järjestelmä ja siihen liittyvien ohjelmien konstruktiota. Konstruktiivisen menetelmän kirjallisuudessa ei yksioikoisesti määritellä, miten konstruktio tulisi toteuttaa, mutta prototypointi ja vaihtoehtoisesten ratkaisujen etsiminen sekä ratkaisu ongelmakehyksessä esiteltyyn ongelman poistamiseen on mainittu. Ensiksi esitellään Android-ohjelman konstruktio ja mahdolliset vaihtoehtoiset tavat saada informaatio käyttäjän päätelaitteelle yhteensä kolmella mahdollisella toteutuskonstruktiolla. Android-ohjelman konstruktio tapahtui iteratiivisesti, ensiksi esitellään ensimmäisenä kehitetty ratkaisu. Toisena esitellään vaihtoehtoinen ratkaisu, joka on jatkokehitetty ensimmäisen toteutuksen oppiman pohjalta. Kolmantena esitellään PWA-sovellus konstruktio, jota voidaan pitää tässä iteraation lopputulemana. Seuraavaksi esitellään Android-ohjelman palvelimen konstruktio, minkä avulla voitiin toteuttaa kaikki kolme vaihtoehtoista Android-sovelluksien toteutusta. Viimeiseksi esitellään teknologian konstruktion, jonka avulla järjestelmäkokonaisuus on toteutettu.

6.1 Android-ohjelman konstruktio

Android-ohjelman vaatimusmäärittelyssä kuvassa 7 esitelty käyttäjäkertomus toteutettiin yksinkertaisimmalla mahdollisella tavalla. Ratkaisuksi Android-ohjelman konstruktion toteutuksessa käytettiin WebView-luokkaa. Sen avulla Android-ohjelmassa voitiin käyttää ja näyttää internetsivun sisältöä ohjelman aktiviteetin sisällä. WebViewin käyttö vaatii internetluvan, joka piti lisätä Android Manifest tiedostoon.

```
<uses-permission android:name=”android.permission.INTERNET” />
```

Luvan esittely pitää olla <manifest> ja </manifest> elementtien sisällä, tarkempi sisältö on liitteessä A. AndroidManifest.xml. Android-ohjelman toteutuksessa ei tarvittu muita kuin internetlupa. Manifestissä määriteltiin myös ohjelman käyttämä ikoni ja teema. Ulkoasun määrittelyssä mahdollistettiin ohjelman käyttäminen päätelaitteessa niin horisontaalisesti kuin vertikaalisesti.

Android-ohjelmassa käytettiin yhtä MainActivity.java-luokkaa (Liite B). Sen rakenne on hyvin yksinkertainen. Luodaan webview-aktiviteetti, jossa ladattava sivu määritellään seuraavasti:

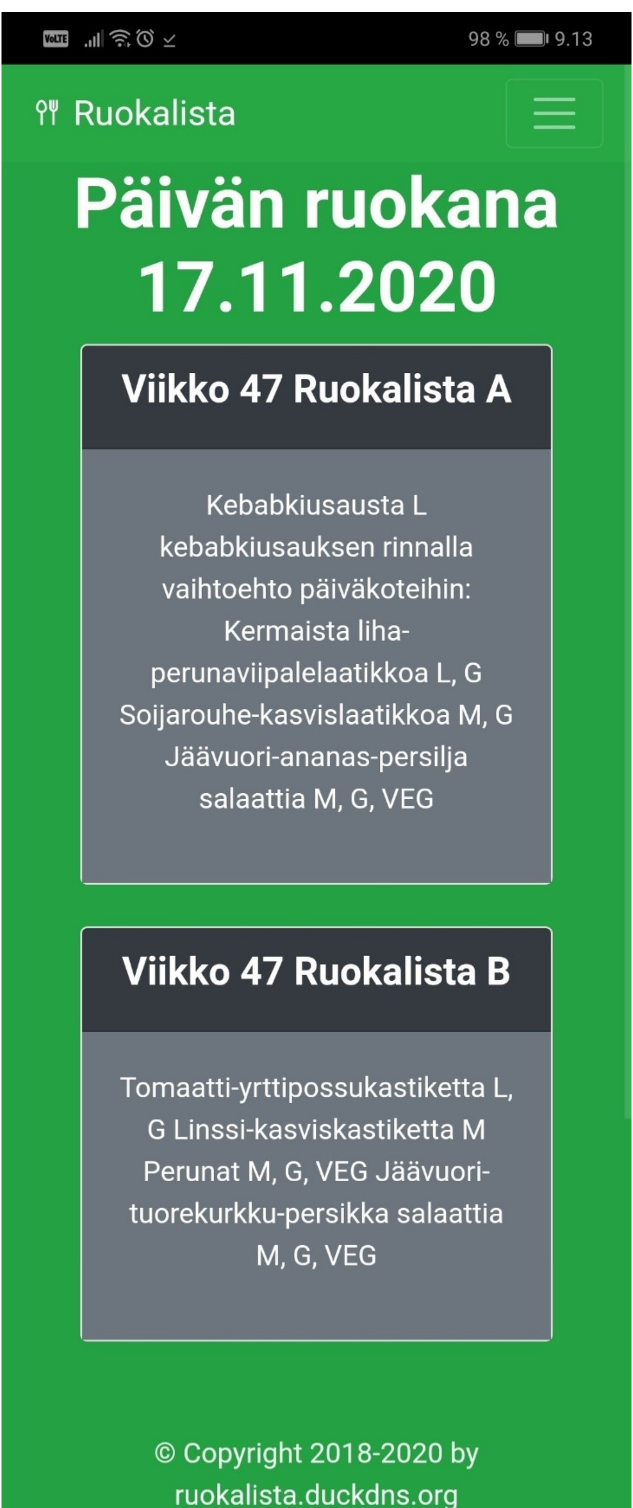
```
myWebView.loadUrl("ip or domain");
```

Tässä haetaan aktiviteetissä näytettävän sivuston sisältöä IP-osoitteen tai domain-nimen perusteella. Näin ollen Android-ohjelman käyttäjä saa tarvitsemansa tiedon mahdollisimman helposti. Lisäksi Android-ohjelman navigointi web-sovelluksessa mahdollistettiin JavaScriptin avulla lisäämällä seuraava:

```
webSettings.setJavaScriptEnabled(true);
```

Kun arvoksi asetettiin true, niin JavaScript-ituki on sovelluksen käytössä. Ohjelman toteutus tehtiin niin, että käyttäjän pysyessä ennalta määritellyn ip-osoite tai domain-alueen sisällä, käyttäjä käyttää WebView-aktiviteettiä. Siinä vaiheessa, kun käyttäjä haluaa siirtyä muualle ennalta määrätystä alueesta sovelluksessa, hänelle avautuu valikko, josta valitaan käytettävä selainohjelma sisällön näyttämiseen. Tällöin käyttäjä jatkaa toisella ohjelmalla informaation selaamista.

Lisäksi Android-ohjelman toteutuksessa tarvittiin activity_main.xml tiedosto, jossa määritellään ohjelman ulkoasu. Liitteessä C activity_main.xml on esitetty koko toteutus. Tällä kertaa ohjelmassa käytettiin RelativeLayout:ia ja määriteltiin kuinka lähelle reunaa Webview-aktiviteetti renderöi ladattavan sivun. Kuvasta 11 nähdään, kuinka sovelluksen ikkuna näkyy kokoruudun kokoisena.



Kuva 11. Kuvakaappaus Android-puhelimen Webview-sovelluksesta.

Android-ohjelman Webview-toteutus vastaa hyvin pitkälti samaa kuin, että selaimessa olisi tehty kirjanmerkki sivustolle ja sen avulla käyttäjä pääsisi käsiksi tarvitsemaansa sivustoon. Nykyisellä toteutuksella käyttäjä pääsee informaation käsiksi helpommin, kun

hänen täytyy vain käynnistää ohjelma oikeasta kuvakkeesta ja informaation sisältö tulee saataville suorilta ilman erillistä navigointia web-selaimessa.

6.2 Android-ohjelman vaihtoehtoinen konstruktio

Toinen mahdollinen toteutus Android-ohjelmalle toteutettiin käyttäen JSON dataa, jossa palvelimella oleva tieto haettiin Flask-sovelluksen API:n kautta. Tämä palvelimelta haettu data jäsenneltiin sopivaan muotoon esitettäväksi Android-sovelluksen käyttöliittymässä.

Toteutus oli yksinkertainen. Sovelluksessa oli kaksi JAVA-luokkaa, joista toinen oli pääluokka tai aktiviteetti niin kuin Android-sovelluksissa yleensä käytetään. Toinen http-yhteyden ja datan hakemiseen palvelimelta tekevä luokka. Lisäksi sovelluksella oli myös AndroidManifest.xml tiedosto, jossa määriteltiin tarvittavat sovelluksen käyttämiseen ja näkymään liittyvät asetukset. Ohjelman käyttöliittymän määrittely tehtiin activity_main.xml ja list_item.xml tiedostoissa.

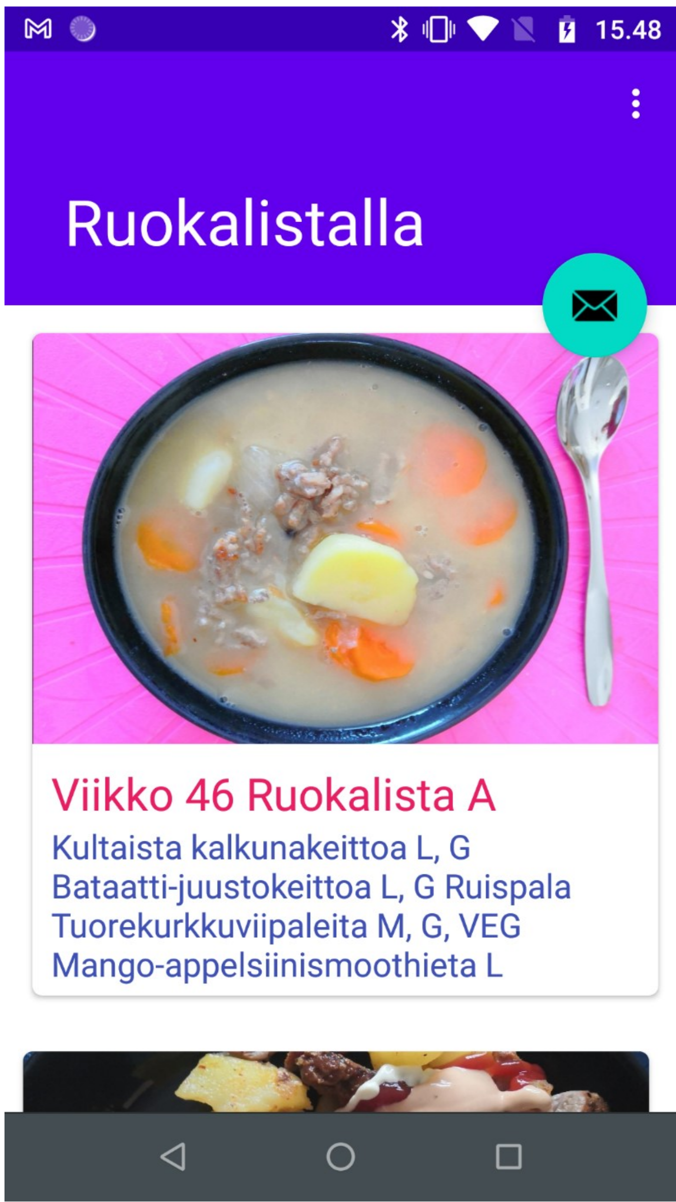
Pääluokan alussa määriteltiin MainActivity, jossa API-rajapinnan URL-osoite määriteltiin Flask-sovelluksen API-rajapintaan eli muodostamaan yhteys osoitteeseen.

```
"https://server_domain_name/api/today/"
```

Lisäksi määriteltiin ArrayList, johon myöhemmin haettava data lisätään. Tämän jälkeen suoritettiin

```
new GetFood().execute();
```

joka laajentaa AsyncTask objektia. Sen avulla voitiin helpolla tavalla ajaa säikeitä taustalla ja näyttää HttpHandler-apuluokan hakutulokset käyttöliittymässä. AsyncTask on tarkoitettu lyhyille käskyille, jollainen on myös JSON datan hakeminen palvelimelta. Siinä on yleensä neljä eri vaihetta, onPreExecute, doInBackground, onProgressUpdate ja onPostExecute. Ensimmäisessä vaiheessa onPreExecute luotiin ProgressDialog, jonka avulla voitiin informoida käyttöliittymän avulla käyttäjää odottamaan hetken, jotta voitiin seuraavassa doInBackground vaiheessa oleva data hakea HttpHandler-apuluokan avulla API-rajapinnasta Flask-palvelimelta. HttpHandler-luokassa tehdään datan hakeminen palvelimelta ja konvertointi JSON-objektista merkkijonoksi. Lopputulos palautettiin takaisin doInBackgroundiin. Datan perusteella muutettiin data joko listaksi käyttäjän käyttöliittymään tai annettiin mahdolliset virhetiedot, jos sellaisia on datan hakemisen yhteydessä tullut viimeisessä onPostExecutessa vaiheessa. Kuvasta 12 nähdään valmiin päiväkohtaisen ruokalistan sisältö siinä vaiheessa, kun haettu data on muutettu listaksi Android-päätelaitteen JSON-sovelluksen käyttöliittymään.



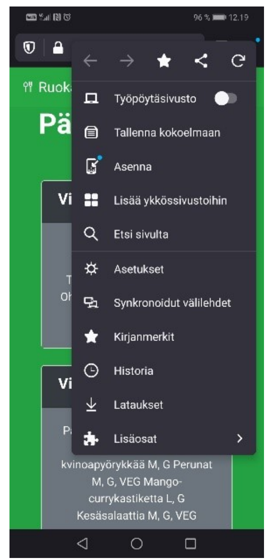
Kuva 12. Android-sovellus JSON datalla.

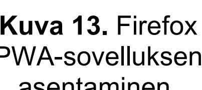
Vaihtoehtoisessa toteutuksessa tarvittava data siirrettiin JSON-objektina ja tarvittavan datan määrä vaihteli 500-600 tavun väliltä. Näin ollen mobiilissa verkossa siirrettävän datan pienuuden takia käyttäjä saa tarvitsemansa tiedon itsellensä mahdollisimman edullisesti, jos hänellä on datan siirtomäärän perusteella oleva laskutus. Samoin tiedonsiirtoon kuluva aika lyhenee ja tieto on nopeammin saatavilla päätelaitteella.

6.3 Progressive web application (PWA) konstruktio

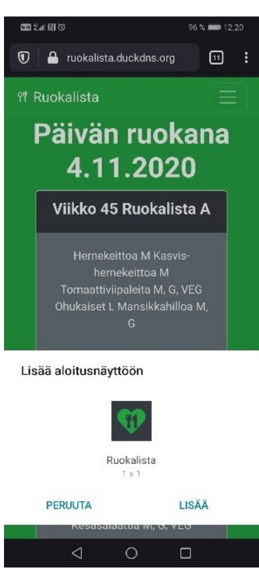
Kolmas mahdollinen konstruktio toteutettiin PWA-sovelluksena. Käytännössä se on ihan normaali web-sovellus tai web-sivu, mutta se on mahdollista asentaa puhelimen aloitusnäytölle samaan tapaan kuin Androidin natiivit-sovellukset. Ominaisuuksiin kuuluu, että PWA-sovellus toimii hyvin samaan tapaan kuin natiivit-sovellukset. Erona sillä on se, että asentamiseen käyttäjän ei tarvitse mennä sovelluskauppaan. Asentamiseen riittää meneminen web-sivulle, jossa käyttäjälle annetaan ilmoitusikkuna mahdollisuudesta asentaa sovellus päätelaitteeseensa.

Käytännössä PWA-sovellus toteutettiin, lisäämällä Flask-sovelluksen static -kansioon PWA-sovelluksen vaatimat tiedostot ja implementointiin ne käytettäväksi Flask-sovelluksessa. Yleensä PWA-sovelluksessa pitää olla vähintään seuraavat tiedostot: service-worker.js, manifest.json, offline.html ja offline.css sekä sovellus ikonit. Toteutus käydään tarkemmin läpi palvelimen konstruktiossa.



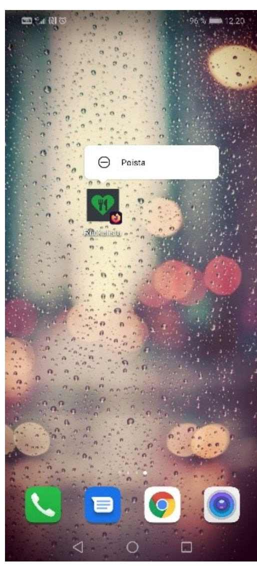


Kuva 13. Firefox PWA-sovelluksen asentaminen.



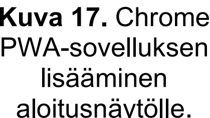
Kuva 14. Firefox PWA-sovelluksen lisääminen aloitusnäytölle

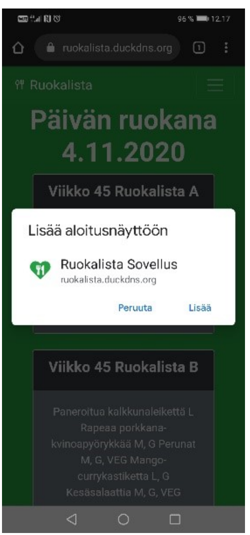
Kuva 15. Firefox PWA-sovelluksen pikakuvake.

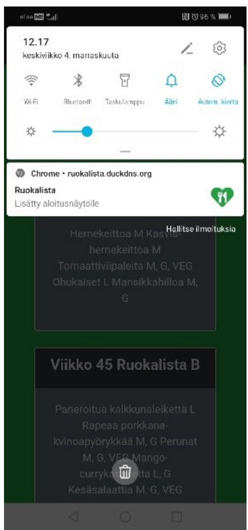


Kuva 16. Firefox PWA-sovelluksen poistaminen.

Kuvien 13-16 esimerkkiasennus PWA-sovellukselle on tehty Firefox-selaimen mobiiliversiolla 82.1.1. Kuvissa 17-20 nähdään PWA-sovelluksen asennus Chrome-selaimen mobiiliversiossa 86.0.4240.110. Kuvassa 13 PWA-ohjelman asennus aloitetaan valikosta ja sieltä valitaan asenna, kun vuorostaan kuvasta 17 nähdään Chrome-selaimen alalaitaan tuleva ilmoitus käyttäjälle. Kuvasta 14 nähdään, että Firefox-selaimen ilmoitus on hyvin samantapainen mitä kuvan 18 Chrome-selaimen ilmoitus PWA-sovelluksen lisäämisestä aloitusnäyttöön. Firefox-selain ei anna käyttäjälle samanlaista informaatiota ilmoitus-valikkoon asennuksesta, kuin kuvasta 19 näkyvä Chrome-selain. Firefox-selain vain näyttää seuraavaksi kuvan 15 PWA-sovelluksen asennus ikonin työpöydällä ja vastaava Chrome-selaimen toteutus näkyy kuvasta 20. Molemmat ovat hyvin samantapaiset, mutta Firefox-selaimen asennuksen kuvakkeessa näkyy, että PWA-sovellus on asennettu kyseistä selaimesta. Chrome-selaimesta asennettu PWA-sovelluksen kuvake ei eroa natiivista Android-sovelluksen kuvakkeesta. Kuvasta 16 nähdään, miten PWA-sovellus voidaan poistaa.







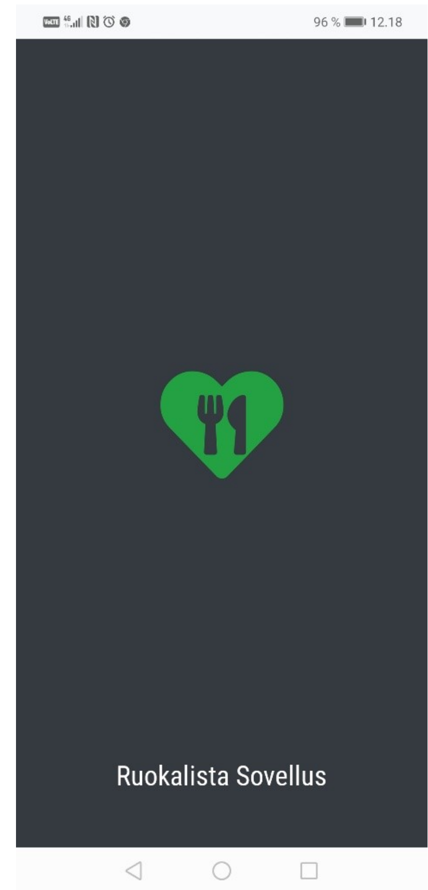
Kuva 17. Chrome PWA-sovelluksen lisääminen aloitusnäytölle.

Kuva 18. Chrome PWA-sovelluksen lisääminen aloitusnäytölle ilmoitus.

Kuva 19. Chrome PWA-sovellus lisätty aloitusnäytölle Android ilmoitus.

Kuva 20. Chrome PWA-sovelluksen pikakuvake.

Manifest-tiedostossa määriteltiin PWA-sovelluksen taustaväri HEX-värikoodilla, jota käytetään myös sovelluksen käynnistyksen yhteydessä. Tästä löytyy esimerkki kuvasta 21. Siinä näytetään sovelluksen ikoni ja sovelluksen ikonin alapuolella nimi. Lisäksi Manifest-tiedostossa määriteltiin näyttöasetukset, jonka avulla PWA-sovelluksen käyttöliittymän tyyppi voidaan PWA-sovelluskehittäjän toimesta ennalta asettaa tietynlaiseksi. Kuvassa 22 olevassa esimerkissä PWA-sovelluksen näyttötyypiksi on määritelty standalone eli se mukailee mahdollisimman paljon natiivisovelluksen näköä ja tuntumaa. Manifest-tiedostossa määriteltiin myös PWA-sovelluksen orientointi ja teeman värit HEX-värikoodilla. Manifest-tiedoston sisältö ja implementointi käydään tarkemmin läpi palvelinohjelman konstruktiossa.



Kuva 21. PWA-sovelluksen latautumisikkuna.

Kuva 22. PWA-sovellus käynnissä.

Kuten kuvista 21 ja 22 nähdään niin PWA-sovellusta ei välttämättä erota Androidin natiivisovelluksesta ulkonäön perusteella.

PWA-sovelluksen asennusehtona on yhteyden HTTPS-salaus. Service workerin rekisteröinti ja asennus tapahtuu, kun käyttäjä saapuu PWA-sovelluksen URL-osoitteeseen, jos vain selain tukee service workerin asennusta. Selain hoitaa omalta osaltaan service workerin rekisteröinnin ja sen implementointi käydään läpi tarkemmin palvelimen konstruktio- luvussa.

Jos käyttäjä haluaa, pystyy hän asentamaan PWA-sovelluksen Android-päätelaitteeseensa. Jos taas selain tai käyttäjä ei halua asentaa sovellusta, niin PWA-sovelluksen pitäisi pystyä toimimaan normaalin internetsivun tavoin. Kun service workerin asennus käynnistyy, ladataan yleensä välimuistiin tarvittavat tiedostot yhteydetöntä käyttöä varten. Asennus onnistuu vain, jos kaikki tiedostot saadaan ladattua. Yleensä välimuistiin ladataan ns. sovelluksen toiminnallisuuteen tarvittavat HTML, CSS ja JavaScript tiedostot. Tarkoitus on luoda käyttäjälle PWA-sovelluksen käynnistyksessä illuusio täysin toimintavalmiista sovelluksesta, vaikka varsinaista dataa oltaisiin vasta hakemassa palvelimelta tai käyttäjälle näytetään edellisellä kerralla haetusta välimuisti tiedoista koostuva näkymä. Kun PWA-sovelluksen service worker on hakenut uusimman version palvelimelta, tulisi käyttäjää informoida tuoreemmasta datasta, jotta hän voi päivittää näkymän uudemmalla tiedolla. Toisaalta service workerin avulla voidaan ilmoittaa myös hänen internetyhteytensä toimimattomuudesta ja tarjota yhteydetön vaihtoehto nähtäväksi PWA-sovelluksessa.

PWA-sovelluksen hyvänä puolena on, että sen pitäisi toimia myös muiden laitteiden, kuten Windows, Chrome OS ja Linux, työpöydällä. Asennus toimii hyvin samaan tapaan kuin Android-päätelaitteessa ja sovellus luo käynnistyskuvakkeen joko työpöydälle tai sovellusvalikkoon. Yksinkertaisimmillaan PWA-sovellus mahdollistaa käyttäjällensä natiivi-sovelluksen tuntuman ja ominaisuudet, ilman että hänen täytyy erikseen mennä sovelluskauppaan. Hänelle riittää vain PWA-sovelluksen asentaminen web-selaimesta.

6.4 Palvelinohjelman konstruktio

Palvelimen-ohjelman konstruktiokieleksi valittiin Python. Perusteeksi kielenvalinnalle oli tulevan järjestelmän ylläpitäjän vaatimusmäärittelyssä esittämä käyttäjäkertomus. Järjestelmän toteutuskielen perusteella vaihtoehtoisia web-alustoja tai web-kehyksiä oli tarjolla useita, mutta web-kehitysalustaksi valittiin Flask. Flask antaa web-sovelluksen kehittäjälle vapaat kädet siitä mitä tai miten hän haluaa web-sovelluksensa kehittää.

Palvelinohjelman konstruktio jakautui kahteen erilliseen ohjelmaan, joista toinen hoitaa tietokannan päivittämisen sekä tiedonhaun RSS-datasta. Samainen ohjelma hoitaa RSS-datan jäsentämisen tietokantaan. Sitä kutsuttakoon tässä yhteydessä tietokannan päivittäjäohjelmaksi. Toinen kehitetty ohjelma hoitaa tietokannassa olevan tiedon näyttämisen Flask web-sovelluksena, tässä yhteydessä puhutaan siitä web-sovelluksena. Syynä kahden eri ohjelman kehittämiselle tuli kehityksen kuluessa. Näin mahdollistettiin molempien ohjelmien itsenäinen kehitys, kunhan vain datamalli pysyi samana tietokannassa. Tietokantana käytettiin sqlite3, joka tämän tyyppisen datan säilömiseen oli helppo valinta, koska se oli yksinkertaista ottaa käyttöön ja resurssien kulutuksen suhteen kevyt. Redis-tietokantaa käytettiin Push-viestien tilaajien avainten ja niiden arvojen tallennukseen PWA-sovelluksessa.

6.4.1 Palvelinohjelman tietokannan päivittäjä

Tietokannan päivittäjä on komentorivisovellus, jota voidaan ajaa joko manuaalisesti tarpeen mukaan tai automatisoituna sovelluksen suorituksena lisäämällä se Linuxissa Cron-scriptiksi tai tarvittaessa myös Python schedule-scriptiksi. Ohjelman toimintaperiaate on, että se hakee halutun viikon ruokalistojen tiedot ja lisää ne tietokantaan Flask-sovelluksen saataville. RSS-syötteen päivittyessä satunnaisesti, ohjelman ajo ajastettiin kerran viikossa tapahtuvaksi suoritukseksi Cron-scriptinä tuotantoympäristössä. Kehitysympäristössä tietokannan päivityksen tapahtui kerran tai pari kertaa viikossa schedule-scriptinä, jos ensimmäisellä kerralla RSS-tietoa ei ollut saatavilla, scripti aktivoi toisen suorituksen samalle viikolle. Välillä myös sovelluksenylläpitäjä joutui tekemään ohjelman ajon manuaalisesti, jos datan jäsentämisessä oli automaatiossa tapahtunut jokin virhe tai dataa ei ollut saatavilla. Jos tarvittavaa RSS-syötettä ei ollut tarjolla, tietokannan päivittäjäohjelmalla voitiin luoda viikosta niin sanottu kalenterimalli, jossa tietoalkioina on päivämäärä sekä päivän nimi. Tällöin Flask-websovellus toiminta jatkuu normaalisti, tosin sen informaatio on vajaa.

Tietokannan päivittäjäohjelma suoritettiin virtualenv-ympäristössä, jotta saatiin projektin sopivat kirjastot asennettua ainoastaan vain kyseiselle ohjelmalle. Tietokannan päivittäjän tarvitsemat Python-kirjastot olivat requirements.txt tiedostossa. Näin sovelluksen kehittäjä pystyi määrittelemään sopivat versiot kirjastoista ja varmistamaan

paremman yhteensopivuuden kohdejärjestelmässä. Tietokannan päivittäjäohjelman requirements.txt sisälti seuraavat kirjastot aakkosjärjestyksessä.

- beautifulsoup4
- bs4
- Click
- feedparser
- python-Levenshtein
- schedule
- SQLAlchemy

Beautifulsoup4 ja bs4 ovat python-kirjastoja, joita käytettiin tiedon jäsentämiseen. Beautifulsoupia käytettiin sovelluksessa feedparserilla haetun tiedon jäsentämiseen. Tietokannan päivittäjäsovelluksessa käytettiin Click-kirjastoa, jonka avulla voitiin tehdä komentorivi-sovelluksesta helpommin lähestyttävä ja käyttöä helpottava, koska sen avulla mahdollistettiin argumenttien käyttäminen sovellusta manuaalisesti suorittaessa. Näin ollen käyttäjä voi suorittaa komentoja ilman, että tarvitsee tietää toiminnallisuudesta kaikkea. Click-kirjaston avulla tehtiin käyttäjälle ohjeet, mitä ja miten tietyt ominaisuudet voi suorittaa. Python-Levenshtein käytettiin tiedon tarkistamiseen ja sen avulla luotiin oikeat otsikot ja niiden sisällöt tietokantaan. Schedulen avulla voitiin toteuttaa sopivien aikataulutettujen ajojen suorittamista tietokantasovellukselle, jos sovellusta ei oltu ajastettu muulla keinoin. SQLAlchemy-kirjaston avulla data luettiin ja kirjoitettiin tietokanta päivittäjäohjelmassa sqlite3-tietokantaan.

Ohjelman suoritus käynnistettiin joko komentorivillä tai ajastettuna virtuaaliympäristössä komennolla.

```
(venv)$ python db_updater.py
```

Tällöin alkoi ohjelman suoritus. Ohjelma teki ensiksi varmuuskopion olemasta olevasta tietokannasta ja nimesi varmuuskopiotiedoston ohjelman käynnistymisajan mukaisesti. Tämän jälkeen ohjelma haki tietoa mahdollisten lisämääritteiden mukaan RSS-syöte. Näitä lisämääritteitä olivat muun muassa -viikkonumero ja -listatyyppi muuttujat. Jos näitä muuttujia ei ole määritelty käynnistysparametreissä, ohjelma haki käynnistyspäivämäärän perusteella RSS-syötteestä kuluvan viikon ja tulevien viikkojen listat. Ohjelma tarkasti, oliko hakutulos tietokannassa. Jos kyseisen viikon listaa ei ollut vielä tietokannassa, niin siinä vaiheessa ohjelma haki sen viikon ruokatiedot ja jäsensi tiedot oikeaan muotoon sekä kirjoitti ne tietokantaan. Jos RSS-syöteessä ei ollut saatavilla listoja halutuilla parametreilla, tietokantapäivittäjällä voitiin tuottaa viikosta päivämäärä ja päivän nimellä oleva yksinkertainen tietomalli Flask-sovellukselle.

Ensimmäinen versio tietokannanpäivittäjästä toimi edellä kuvatulla tavalla, mutta kun RSS-syötteen saatavuus hävisi yllättäen, sen tilalle kehitettiin toinen versio ohjelmasta. Toiminnalleen uudistettu tietokantapäivittäjä poikkesi alkuperäisestä, nyt tieto jouduttiin hakemaan verkkosivulta suoraan ja siitä koostettiin otsikko ja otsikko-osoite lista. Tämä lista muokattiin vielä sellaiseen muotoon, että siitä saatiin lisäksi vielä viikkonumero. Sen jälkeen tarkistettiin tietokannasta, oliko otsikko tietue tietokannassa. Jos otsikkoa ei löytynyt, kyseinen otsikko, osoite ja viikkonumero jatkoivat seuraavaan vaiheeseen. Jos otsikko löytyi tietokannasta, ohjelman suoritus keskeytettiin, ellei tarkastuksessa löytynyt tietokantaan lisättävää uutta tietoa. Uuden tiedon haku suoritettiin kyselynä osoitetiedon ja beautifulsoup-kirjastoa avulla. Haettu tieto jouduttiin vielä muotoilemaan oikeaan muotoonsa siten, että ainoastaan haluttu osa kokonaisuudesta olisi valmiina tietokantaan tallentamiseen. Viikkonumeron avulla muodostettiin sopivat hakusanat, millä haetusta

tiedosta muodostettiin arkipäiville sopiva lista. Lisäksi listaa jouduttiin muokkaamaan edelleen, koska siinä oli mm. aamupala ja päivällistiedot mukana. Ainoastaan lounastieto haluttiin tallentaa tietokantaan. Sitten kun tieto oli jäsennelty haluttuun muotoon, se tallennettiin tietokantaan. Tietokantaan tallennuksessa jokaiselle riville tallennettiin tunniste, otsikko, lisäyspäivämäärä ja lounastiedot sekä päivämäärä milloin lounas olisi tarjolla. Tämä uudistettu kokonaisuus tietokannanpäivittäjästä otettiin käyttöön Flask-websovelluspalvelimella ajastettuna Cron-scriptinä.

Erona näillä kahdella toteutuksella oli se, että iteraation jälkimmäistä toteutusta yksinkertaistettiin muun toiminnallisuuden osalta. Sen avulla ei enää tehty varmuuskopiota tai sen komentorivi suoritusta ei voinut yksilöidä. Uudempi toteutus tietokantapäivittäjästä toteutettiin yksinomaan ajatellen sen suorittamisen etupäässä cron-scriptinä siten, että suorituksesta tallennettaisiin kattava lokitiedosto. Jos ja kun automaattiseen suoritus jostain syystä keskeytyisi, syy-seuraussuhde olisi helpompi löytää lokitiedoston avulla.

6.4.2 Palvelinohjelman Flask-websovellus

Flask-websovelluksen konstruktio oli iteratiivinen, ensimmäisellä versiolla sen avulla toteutettiin websivu. Sen avulla Android-ohjelmassa pystyttiin näyttämään haluttu tieto Webview-sovelluksessa. Toisessa versiossa Flask-websovellukseen kehitettiin API, jonka avulla Android-ohjelman vaihtoehtoisen konstruktion JSON-data saatiin palvelimelta. Kolmannessa versiossa Flask-websovellukseen kehitettiin tuki PWA-sovellukselle. Sitä vielä jatkokehitettiin ja sen avulla voitiin lähettää push-viestejä niille käyttäjille, jotka olivat push-viestit tilanneet.

Flask-websovelluksen konstruktion hakemistorakenne kokonaisuudessaan näkyy kuvasta 23. Flask-sovellus jakautui juuri-, instance- ja ruokalistahakemistoihin. Juurihakemistossa olevat tiedostot käydään läpi aakkosjärjestyksessä sekä niiden toiminnallisuus tässä konstruktiossa. Automation.py tiedostossa käsiteltiin Flask-sovelluksen suoritusaikaisen raportoimisen ja asetusten päivittämisen automaatiota, mistä esimerkkinä service-worker.js tiedoston päivittäminen ja raportointi päivityksen tilasta sähköpostitse ylläpitäjälle. Config.py tiedostossa määriteltiin Flask-sovelluksen erilaiset sovelluksen suorituksen tilat, tosin tämä tiedosto on vain mallina ja pohjana instance-hakemistossa olevalle config.py tiedostolle. Flask-sovelluksen tietokanta on db.sqlite3 tiedosto, jota päivitettiin tietokantapäivittäjällä. Document_updater.py liittyi PWA-sovelluksessa olevan service-worker.js tiedoston päivittämiseen. Se toimi apuna automaatiolle ja tiedoston muokkaamiselle. Instance-hakemistossa oleva config.py oli varsinainen konfigurointi tiedosto Flask-sovellukselle, sen avulla voitiin määritellä erilaisten ympäristöjen ja tilojen parametrit. LISENCE ja README.md olivat versionhallintaan ja asennukseen liittyviä tiedostoja. Flask-sovelluksen tarvitsemat Python-kirjastot löytyvät requirements.txt tiedostosta. Kokonaisuuden käynnistys voitiin tehdä käyttäen run.py tiedostoa tai Makefile:n avulla.

```
.
├── automation.py
├── config.py
├── db.sqlite3
├── document_updater.py
├── error.log
├── instance
│   └── config.py
├── LICENSE
├── Makefile
├── README.md
├── requirements.txt
├── run.py
├── ruokalista
│   ├── api
│   │   ├── __init__.py
│   │   └── views.py
│   ├── helper.py
│   ├── home
│   │   ├── __init__.py
│   │   └── views.py
│   ├── __init__.py
│   ├── menulist
│   │   ├── __init__.py
│   │   └── views.py
│   ├── models.py
│   ├── static
│   │   ├── app.js
│   │   ├── favicon.ico
│   │   ├── images
│   │   ├── js
│   │   │   └── main.js
│   │   ├── manifest.json
│   │   ├── robots.txt
│   │   ├── service-worker.js
│   │   └── style.css
│   ├── templates
│   │   ├── errors
│   │   │   ├── 404.html
│   │   │   └── 500.html
│   │   ├── home
│   │   │   ├── about.html
│   │   │   └── index.html
│   │   ├── layout.html
│   │   └── menulist
│   │       ├── week_a.html
│   │       └── week_b.html
│   └── wsgi.py
├── ruokalista.ini
├── test_app.py
└── wsgi.py
```

Kuva 23. Ruokalista Flask web-sovelluksen rakenne

Ruokalista-hakemistosta löytyy __init__.py, joka on päätiedosto Flask-sovellukselle. Tässä tiedostossa tehtiin Flask-sovelluksen ja sinikopioiden määrittelyt sekä käyttöönotto. Samassa hakemistossa oli myös helper.py, joka toimi avustajana erilaisille näkymille ja tietokannasta tiedon hakemisessa sekä muuntamisessa oikeaan muotoon. Models.py tiedostosta löytyi tietokantamallit. Sinikopiot olivat api-, home- ja menulist-hakemistoissa. Static-hakemistossa on PWA-sovelluksen toiminnallisuuteen tarvittavat tiedostot ja Flask-sovelluksen web-näkymän tyylitiedostot. Templates- hakemistosta löytyy Flask-sovelluksen Jinja2-kaavat, joiden avulla käyttöliittymään saatiin oikea toiminnallisuus. Error.log-tiedostoon tuli ohjelman suorituksenaikaiset virhetiedot sekä tietyt informatiiviset tiedot riippuen siitä, millä parametreillä Flask-

websovellusympäristö on käynnistetty. Test_app.py tiedostossa oli Flask-sovelluksen yksikkö- ja integraatiotestit.

Flask web-sovelluksen kehitysympäristön ja tuotantoympäristön sekä testausympäristön muuttujat valitun ympäristön ajon aikaiseen toimintaan voitiin toteuttaa monella tavalla. Config.py tiedostossa olevien parametrien ja run.py sekä wsgi.py käynnistäjien avulla Flask web-sovelluskehittäjä pystyi käynnistämään ohjelman toivotulla tavalla. Esimerkiksi, kun sovellusta kehitettiin, hän valitsi development eli kehitysympäristön komentoriville syöttämällä sopivat komennot. Itse Flask web-sovelluksen käynnistämien tapahtuu komennolla

$ flask run

Tällöin kehitysympäristö käynnistyi ja web-sovellus on käytettävissä kuten kuvasta 24. nähdään. Kun Flask-sovelluksen kehitysympäristö oli kehitystilassa käynnistetty, niin myös virheet näytettiin web-käyttöliittymässä, jos virheitä ilmeni.

```
(venv) teemu@EliteBook-2570p:~/koodit/working_dir/zero_pi/ruo
 * Serving Flask app "run.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-775-544
```

Kuva 24. Flask web-sovelluksen kehitysympäristön käynnistäminen.

Flask-sovelluksen kehittäjä pystyi käynnistämään web-selaimen kuvan 24 osoittamaan web-osoitteeseen eli localhostin porttiin 5000. Tällöin hän näkee reaaliaikaisesti kehitystyönsä. Myös web-selaimen kautta tapahtuva virheiden etsintä oli mahdollista mahdollisten virheiden löytyessä ja niiden aiheuttajia etsiessä.

Kehitettävässä järjestelmässä Flask web-sovelluksen modulaarisuus ja laajennettavuus toteutettiin sinikopioiden avulla. Kun sinikopio luotiin luomalla objekti luokasta Blueprint, siihen tarvittiin kaksi argumenttia, sinikopion nimi ja moduulin tai paketin sijainti hakemistossa. Esimerkki sinikopio luotiin, kun API:n sinikopiota tehtiin koodissa, api- hakemistossa olevaan __init__.py tiedostoon lisättiin seuraavat kohdat.

```
from flask import Blueprint

api = Blueprint('api', __name__)

from . import views
```

Itse Flask-sovellukseen sinikopio rekisteröitiin ruokalista-juurihakemistossa olevaan __init__.py tiedostoon lisäämällä loppuosaan tiedostoa ennen return app kohtaa.

```
from .api import api as api_blueprint
app.register_blueprint(api_blueprint)
```

Itse toiminnallisuus sinikopiolle luotiin api-hakemistossa olevaan view.py tiedostoon. Siellä jokainen reitti sovelluksen näkymään muotoiltiin seuraavalla tavalla. Huomioitavaa oli api reitti-decorator, joka on muotoa `@api.route`. Tämä reitti-decorator pitää muistaa nimetä oikein sinikopion mukaan.

```
@api.route('/api/today/', methods=['GET', 'POST'])
def today_api():
    """Function json api for dailymenu."""
       # implementointi koodi tähän
    return jsonify(list_of_menus)
```

Sinikopioiden ominaisuuden lisääminen Flask web-sovellukseen mahdollisti myös erillisen frontend-viitekehyksen käyttämisen, jos tai kun sellaiselle tulee tarvetta tulevaisuudessa. API:a käytettiin myös Android- ohjelman vaihtoehtoisen konstruktion datalähteenä, joten se sopi myös erilliselle sovellukselle API:ksi.

Templates- hakemistossa olivat Flask-sovelluksen Jinja2-mallit. Layout.html oli perusmalli, jossa määriteltiin sovelluksen ylä- ja alatunnisteet. Sieltä määriteltiin web-sovelluksen sisältö, joka tuli näkyviin seuraavan HTML-luokka kontin sisälle.

```
<div class="container">

  {% block content %}
  {% endblock %}

</div>
```

Esimerkiksi kotisivu eli index-sivu renderöitiin block content ja end blockin väliin, niin kuin kaikkien muidenkin sivujen sisällöt. Kotisivun mallissa vuorostaan oli seuraavat kohdat, eli laajennetiin layout.html sisältöä index.html sisällöllä.

```
{% extends "layout.html" %}
{% block content %}

{% endblock %}
```

Näin index.html mallissa voitiin toteuttaa sinne tuleva toiminnallisuus ja sovelluksen kehys pysyi koko ajan samana jokaiselle sinikopioiden malleille. Home-hakemistossa olivat about.html ja index.html sivujen mallit. Tässä sovelluksessa eriteltiin virhesivut omaan errors- hakemistoon, mutta niistä ei tehty erillistä sinikopiota. Virheellisten sivujen ja API-kutsujen toiminnallisuus toteutettiin ruokalista-hakemistossa olevaan init tiedostoon. Siellä eriteltiin erikseen API ja websivujen virheet, jolloin Flask-sovellus vastasi oikealla tavalla sille tuleviin kutsuihin. Templates-hakemistossa oli myös menulist-hakemisto, jossa olivat viikkokohtaisten sivujen mallit. Niiden toteutus tehtiin sinikopiona ja niille lisättiin omahakemistonsa ruokalista-hakemistoon.

Lisäksi Flask web-sovelluksessa oli myös static-hakemisto. Siellä sijaitsivat sovelluksen ns. muuttumattomat tiedostot, joita olivat favicon.ico ja style.css tyylitiedosto. Samaisessa hakemistossa olivat myös PWA-sovelluksen vaatimat app.js, manifest.json ja service-worker.js tiedostot. Static- hakemistossa oli PWA-sovelluksen tarvitsemat kuvat images- kansiossa. PWA-sovelluksen käynnistysikonien kuvat ja sovelluksen aloitusikkunalle määriteltiin manifest.json tiedostossa. PWA-sovelluksen

toimintalogiikka implementoitiin service-worker.js tiedostoon. Flask web-sovellukseen PWA-sovelluksen toiminnallisuuden toteuttaminen vaati myös muutaman muutoksen templates-hakemistossa olevaan layout.html tiedostoon, jotka lisättiin tiedoston <head> osioon.

```
<meta name="theme-color" content="#009933">
<meta name="Description" content="Ruokalista sovellus kouluille ja päiväkodeille.">
<link rel="manifest" href="{{ url_for('static', filename='manifest.json') }}">
<link rel="icon" sizes="192x192" href="{{ url_for('static', filename='images/ruokalista-web192.png') }}">
<link rel="apple-touch-icon" href="{{ url_for('static', filename='images/ruokalista-web192.png') }}">
```

Ensimmäisenä oli PWA-sovelluksen teeman väri, joka oli HEX-arvona. Seuraavana oli PWA-sovelluksen deskriptio eli kuvaus sovelluksesta, jossa kerrottiin lyhyesti sovelluksen tarkoitus. Tämän jälkeen määriteltiin Flask-sovelluksen avulla staattisen manifest.json tiedoston sijainti, myös sovelluskuvakkeiden sijainti määriteltiin samalla tavalla. Saman layout.html tiedoston loppuun </footer> osion jälkeen ennen </body> lisättiin pari scriptiä, jotka olivat PWA-sovelluksen toiminnallisuuden kannalta tarpeelliset.

```
<script type="text/javascript" src="{{ url_for('static', filename='app.js') }}" ></script>
<script>
  $(document).ready(function(){ $('[data-toggle-tooltip="tooltip"]').tooltip(); });
  </script>
```

Näistä ensimmäisenä olevan app.js avulla toteutetiin PWA-sovelluksen service-worker.js tiedoston ja niiden tuen olemassaolon tarkistus käyttäjän päätelaitteella olevassa selaimessa. Samaisessa tiedostossa toteutettiin ilmoituksen hallinta käyttöliittymään ja mahdollisen uudelleen näyttämiseen. Alemman scriptin avulla toteutettiin sovelluksessa ponnahdusikkuna, joka informoi käyttäjää uudemmasta sisällöstä sovelluksen käyttöliittymän alaosassa.

PWA-sovelluksen service-worker.js tiedostolla hallittiin sovelluksen toiminnallisuuden kannalta merkittäviä asioita. Siellä sijaitsevat PWA-sovelluksen asennus ja asennukseen kuuluvien tiedostojen logiikka. Samaisella tiedostolla voitiin määritellä, miten kyseinen sovellus käyttäytyy, lataako se ensiksi tiedon internetin kautta vai näyttääkö se käyttäjälle välimuistissa olevan tiedon. Esimerkkinä, jos internetyhteyttä ei ole saatavilla tai verkko on katkonainen, millaisen näkymän PWA-sovelluksen käyttäjälle näytetään. Ruokalista PWA-sovelluksessa lähtökohtaisesti ensiksi näytettiin käyttäjän päätelaitteella välimuistissa olemassa oleva tieto, joka päivitettiin taustalla uudemmaksi, jos sellainen oli vain saatavilla. Uudesta PWA-sovelluksen versiosta annettiin käyttäjälle kuvan 25 mukainen informaatio sovellus ikkunan alareunassa. Käytännössä uusi versio on vain sovelluksen tietojen päivitys uudempiin versioihin.

Uusi versio ruokalista sovelluksesta saatavilla. Päivitä tästä

Kuva 25. PWA-sovelluksen informaatio ilmoitus uudesta sovellus versiosta.

Halutessaan käyttäjä pystyi päivittämään PWA-sovelluksensa ja näin saada uusimman version heti käyttöönsä tai hän pystyi jatkamaan edellisen version käyttämistä. Kun käyttäjä aktivoi päivityksen, niin uudempi service worker otti komennon. Jos yhteyttä ei ollut saatavilla, niin käyttäjää informoitiin, että hänellä ei ole internet-yhteyttä tällä hetkellä ja hänelle näytettiin välimuistissa oleva tieto. Evaluointi luvussa käydään läpi tarkemmin miten tavallisen Flask web-sovelluksen ja Flask PWA-sovelluksen toiminnallisuuden ja toteutuserot näkyvät käyttäjälle. PWA-sovellus vaatii myös muutaman muutoksen teknologian konstruktion toteutukseen, jotta kaikki PWA-sovelluksen toiminnallisuuden vaatimukset täyttyivät.

PWA-sovellukselle olennaiset push-viestit toteutettiin ensiksi erillisen palvelun avulla. Ensimmäisen toteutuksen push-viestien välityksen hoiti OneSignal. Sen integraatio projektiin oli hyvin yksinkertainen ja helppo toteuttaa. Flask-sovelluksen static-kansioon lisättiin OneSignalSDKWorker.js ja OneSignalSDKUpdaterWorker.js tiedostot. Nämä tiedostot ladattiin OneSignal palvelusta palvelimelle ja niiden sisältöä ei itse pidä mennä muuttamaan. Lisäksi olemassa olevaan manifest.json tiedostoon lisättiin yksi rivi, jossa määriteltiin OneSignaalin palvelussa määritelty sovelluksen tunniste. Edellä lisättyjen tiedostojen näkyvyys Flask-sovelluksessa piti sovelluksen pohjamalli layout.html tiedostoon lisätä scriptit.

```
<script src="https://cdn.onesignal.com/sdks/OneSignalSDK.js" async="">
</script>
<script>
 var OneSignal = window.OneSignal || [];
 OneSignal.push(function() {
   OneSignal.init({
     appId: "Tähän sovelluksen id",
     notifyButton: {
       enable: true,
     },
   });
 });
</script>
```

Tällöin saatiin kuva 26 näkyvä punainen push-viestien tilausnappi näkyville PWA-sovelluksessa. Napin sijainti oli sovellusikkunan oikeassa alalaidassa. Se pysyi koko ajan näkyvillä käyttöliittymässä. Kun push-viestin tilaus oli aktivoitu, käyttäjälle näytettiin samainen nappi kuvan 27 tapaan. Tällöin nappi oli kooltaan hiukan pienempi ja se oli jonkin verran läpinäkyvä. Samalla kun käyttäjä aktivoi push-viestien tilauksen hänen käyttöliittymäänsä näytetään kiitos viesti push-viestin aktivoimisesta. Kuvan 27 punaista tilausnappia painamalla käyttäjä pystyi poistamaan käytöstään push-viestien vastaanottamisen niin halutessaan.

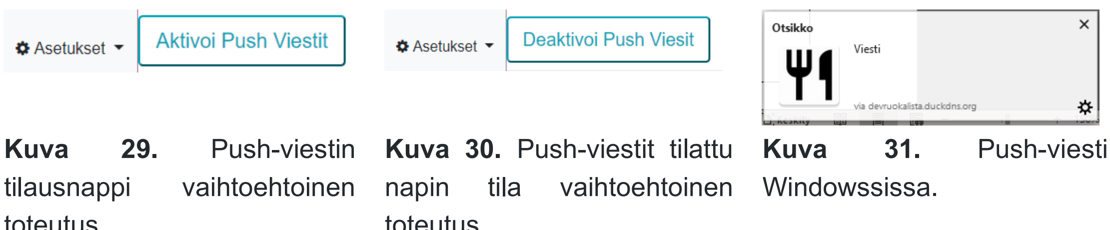
Kuva 26. Push-viestin tilausnappi.

Kuva 27. Push-viestit tilattu napin tila.

Kuva 28. Push-viesti Androidissa.

Kuvasta 28 nähdään esimerkki push-viestistä Android päätelaitteella. Se koostui otsikosta ja viestistä sekä PWA-sovelluksen kuvakkeesta. PWA-sovelluksen push-viestit olivat näkymältään ja tuntumaltaan hyvin samankaltainen kuin natiivien Android-sovelluksien push-viestit. PWA-sovelluksen push-viestien välityksen päätelaitteessa kulki service-worker.js ja OneSignal palvelun tiedostojen avulla. Push-viestit voitiin lähettää käyttäjälle OneSignalin palvelun web-sivuhallinnan kautta tai erillisen Python-sovelluksen avulla. Ruokalista projektin toteutuksessa viestien lähetys automatisoitiin, niin että niiden tilaaja sai push-viestin tiettyyn kellon aikaan.

Push-viestien vaihtoehtoinen toteutus kehitettiin, kun ensimmäisessä toteutuksessa ja sen toiminnassa ilmeni jollain selaimilla toiminnallisuuden rajoitteita. Etupäässä viestit eivät tulleet perille. Toisessa toteutuksessa push-viestien välitys hoidettiin samalla palvelimella, millä itse Flask-sovellusta ajettiin. Siinä käytettiin Redis avain-arvo-tietokantaa, push-viestien tilauksien tallentamiseksi. Vaihtoehtoisessa toteutuksessa push-viestien tilaus vaihdettiin Flask-sovelluksen asetuksiin. Kuvasta 29 nähdään tilausnappi ja kuvasta 30 tilauksen perumisnappi.



Kuva 29. Push-viestin tilausnappi vaihtoehtoinen toteutus.

Kuva 30. Push-viestit tilattu napin tila vaihtoehtoinen toteutus.

Kuva 31. Push-viesti Windowssissa.

Tilauksen push-viesti näkyy kuvasta 31, se toimi myös tietokoneella. Tässä esimerkki tapauksessa viesti tuli Windows 10 näkyviin ilmoituksena työpöydälle oikeaan reunaan. Kun vertaillaan keskenään kuvan 28 ja 31 push-viestejä niin voidaan todeta toiminnallisuuden olevan hyvin samankaltaisia. Molemmissa löytyy viestiotsikko ja itse viestiosuus sekä ikoni kuvake. Lisäksi molemmissa oli myös osoite, mistä viesti on lähetetty.

Vaihtoehtoinen push-viestien toteutuksen implementointi vaati Flask-sovellukseen flask-jwt_simple asennuksen, jonka avulla voitiin toteuttaa ylläpitäjän kirjautuminen. Sen avulla voitiin rajata push-viestien lähettäminen vain ylläpitäjälle. Toteutuksessa käytettiin Push- ja Notification-apia hyväksi. Käytännössä, kun käyttäjä tilasi push-viestit, hänelle generoitiin asiakastunniste, joka tallennettiin Redis-tietokantapalvelimelle ja käyttäjän selaimeen paikalliseen varastoon. Samalla käyttäjän käyttöliittymän asetukset päivitettiin kuvan 29 ja 30 mukaiseksi. Käyttäjäviestien välitys vaati myös VAPID -avaimien luonnin palvelimelle. Ylläpitäjän viestien lähetys tapahtui komentorivi pohjaisella käskyllä, joka oli muotoa.

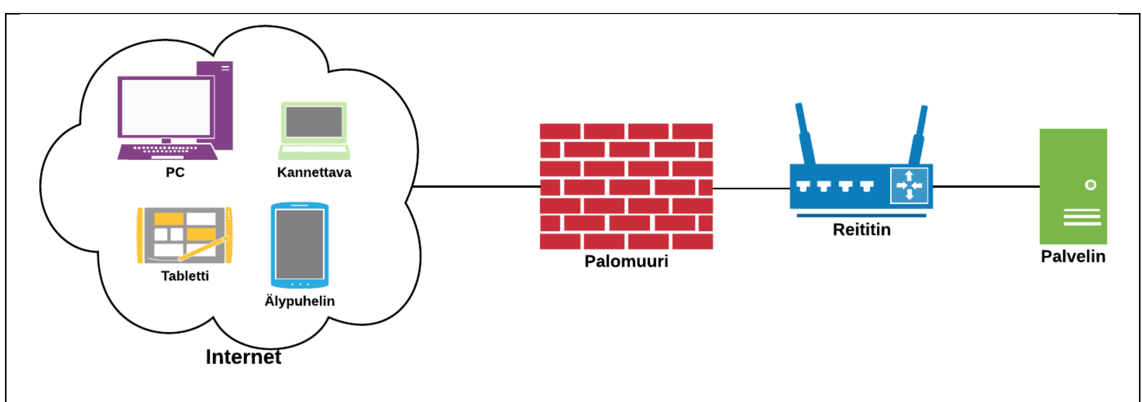
```
(venv)$ python pushmanager.py --title="Otsikko" --
url=https://palvelimenosoite.net --body="Viesti"
```

Tällöin käyttäjä sai kuvan 31 mukaisen viestin päätelaitteeseensa.

Yksi tärkeimmistä asioista palvelinohjelman konstruktiossa oli siihen liittyvät testit, jotta toiminnallisuuden ja koodin testaus voitiin automatisoida mahdollisimman pitkälle. Ennen kuin muutokset voitiin hyväksyä ja implementoida kehitysversiosta tuotantoversioon, tuli muutoksien vaikutukset testata. Sovelluksen Python- koodin testit olivat test_app.py tiedostossa. Näillä testeillä testattiin esimerkiksi Flask-sovelluksen sivujen toiminta. Lisäksi projektissa oli käytössä myös Robot Framework-käyttöliittymän toiminnallisuuden testauksessa. Testien avulla voitiin toteuttaa jatkuva integraatio ohjelman julkaisusta, joko palvelimelle tai erilliseen Docker[20]-virtualisointiympäristöön. Lisäksi Flask-sovelluksesta tehtiin staattinen vedos freezer-kirjaston avulla. Tämä staattinen vedos voitiin julkaista sellaisenaan Netlify[21] palvelussa.

6.5 Teknologian konstruktio

Teknologian konstruktio toteutettiin kuvan 32 mukaisesti. Tuotantopalvelimen alustana toimi Raspberry Pi 1 (model B+) yhden piirilevyn tietokone, jossa oli asennettuna Raspbian Buster Lite Linux järjestelmä. Sen pohjalle rakennettiin palvelinasennus Flask web-sovellukselle. Reitittimenä toimi Linksys WRT54G sarjan langaton reititin. Sen valmistajan ohjelmisto oli korvattu DD-WRT:n Linux pohjaisella avoimen lähdekoodin ohjelmistolla.



Kuva 32. Teknologian konstruktio

Tuotantopalvelimen palomuurin rautana toimi atom-pohjaisella prosessorilla toimiva tietokone, jossa on palomuurina Pfsense versio 2.4.5-RELEASE-p1. Internettiin päin kaistaa oli 100 Mbps verran. Dynaamisena nimipalvelimena toimi DuckDns. Palomuuri hoiti nimipalvelimen ja portin ohjaukset palvelimelle. Palomuurissa oli myös rajoitettu pfBlockerNG avulla eniten spämmiä lähettävät ip-osoitealueet, jottei palvelin tukehtuisi jatkuvaan porttien skannaamiseen. Käytännössä konstruktoidun palvelun tyypin vuoksi voitiin rajoittaa kaikki Suomen ulkopuolelta tulevat ip-alueet pois, koska palvelun tarve oli rajoittunut pienelle alueelle.

Kehityspalvelimen konstruktio oli hyvin samantapainen kuin tuotantopalvelimen konstruktio. Siellä palvelimena toimii Raspberry Pi 3 (Model B), johon oli asennettuna Raspbian Buster Lite Linux. Palomuurina toimi Celeron- pohjainen tietokone, jossa oli asennettuna Opnsense versio 20.7. Reitittimenä toimii Linksys WRT54G sarjan langaton reititin, johon oli asennettu DD-WRT. Internettiin päin kaistaa oli saman verran kuin tuotantopalvelimella. Samaan tapaan DDNS eli dynaamisena nimipalvelimena toimi DuckDns.

Tuotantopalvelimen konfigurointi ja ohjelmien asennuksen kulku meni pääpiirteittään seuraavasti. Kun Rasbian Buster lite- levykuva oli kirjoitettu muistikortille, kortin juurihakemistoon lisättiin ssh -niminen tiedosto. Näin saatiin otettua käyttöön SSH-palvelin, kun Raspberry pi käynnistettiin. SSH-palvelin oli turvallisuus syistä otettu pois käytöstä oletuksena, koska käyttäjä ja salasana oli valmiiksi konfiguroitu levykuvaan. Raspbianissa käyttäjä oli pi ja salasana raspberry, tällöin kenellä tahansa pääsy koneelle olisi todella helppoa pi-käyttäjänä. Kun käynnistys oli suoritettu ja Raspberry pi oli käynnissä, täytyi Raspbian- järjestelmä päivittää. Se tapahtui seuraavalla komennolla komentoriviltä.

```
$ sudo apt-get update && sudo apt-get upgrade -y
```

Tämän jälkeen luotiin uusi käyttäjä järjestelmään tietoturvallisuuden vuoksi. Tässä tapauksessa olkoon esimerkki käyttäjän nimi pääkäyttäjä.

```
$ sudo adduser pääkäyttäjä
```

Yllä olevalla komennolla luotiin uusi pääkäyttäjä järjestelmälle. Tämän jälkeen lisättiin uudelle pääkäyttäjälle sudo oikeudet, jotta sitä voitiin käyttää järjestelmän ylläpidossa.

```
$ sudo adduser pääkäyttäjä sudo
```

Seuraavaksi luotiin pääkäyttäjän ssh-avaimille kansio ja lisättiin oikeudet kansioon oikeiksi.

```
$ sudo mkdir /home/pääkäyttäjä/.ssh

$ sudo chown pääkäyttäjä:pääkäyttäjä /home/pääkäyttäjä/.ssh/
```

Seuraavaksi lisätiin omalta koneelta oma ssh-avain äskettäin luotuun ssh-kansioon komennolla.

```
$ ssh-copy-id pääkäyttäjä@<raspberry piin ip-osoite>
```

Kun avain oli kopioitu kohdejärjestelmään, voitiin muuttaa ssh-kirjautuminen avain pohjaiseksi ja poistaa salasana kirjautuminen kokonaan.

```
$ sudo nano /etc/ssh/sshd_config
```

Tällöin komennolla aukesi nano tekstieditori. Editoitiin kohta PasswordAuthentication yes arvo muotoon PasswordAuthentication no, jolloin poistettiin käytöstä salasana kirjautuminen. Lisäksi samalla muutettiin Port 22 arvoksi esimerkiksi 44344. Tallennettiin tiedostomuutokset ja käynnistettiin ssh.service uudestaan komennolla.

```
$ sudo systemctl restart ssh.service
```

Olemassa olevan ssh- yhteyden pitäisi katketa ja uudelleen ssh-yhteyttä otettaessa raspberry piihin yhteydenotto muuttui seuraavanlaiseen muotoon

```
$ ssh -p 44344 pääkäyttäjä@<raspberry_pi_ip_osoite>
```

Komentoon lisättiin p-vivulla ssh_config tiedostoon muutettu porttinumero. Kun yhteys oli muodostunut pääkäyttäjälle, voitiin seuraavaksi poistaa pi- käyttäjä järjestelmästä kokonaan.

```
$ sudo deluser pi
```

Näin saatiin konfiguroitu tuotantopalvelimen etähallinta pykälää turvallisemmaksi. Tämän lisäksi hyvä olisi asentaa Fail2ban, jonka avulle voitaisiin rajoittaa epämääräiset yhteydenotot tuotantopalvelimeen ja tarvittaessa myös estämään kokonaan. Tässä toteutuksessa Fail2ban:ia ei asennettu, koska tuotantopalvelimen konfiguraatio tapahtui paikallisverkon kautta. Palomuurista estettiin pääsy muihin portteihin kuin Flask-sovelluksen tarvitsemiin portteihin 80 ja 443.

Tuotantopalvelimen konfigurointi jatkui Flask web-sovelluksen toimintaan saamiseksi. Tässä projektissa käytettiin www-palvelimena Nginx versiota 1.14.2, joka asennettiin pakettienhallinnan kautta. Lisäksi järjestelmään piti asentaa python3- ja virtualenv-ympäristöt, jotta Flask web-sovellus voisi toimia. Itse websovellukselle luotiin hakemisto komennolla.

```
$ mkdir /var/www/ruokalista
```

Tämän jälkeen piti kopioida hakemistoon Flask web-sovellus. Kuvan 23 mukainen hakemistorakenne havainnollistaa perusrakenteen. Kun web-sovellus haluttiin saada tuotantokäyttöön, jouduttiin tekemään muutama pieni, mutta tärkeä muutos tiedostoihin. Ensimmäiseksi piti luoda virtualenv-ympäristö ja aktivoida se, jotta vaadittavat requirements.txt tiedostossa olleet Python-kirjastot voitiin asentaa rajattuun ympäristöön vain kyseiselle web-sovellukselle. Lisäksi piti asentaa myös uWSGI, jotta saatiin Nginx keskustelemaan Flask web-sovelluksen kanssa. Kehitysympäristössä uWSGI ei ole tarpeen, mutta kehityspalvelimelle se täytyi myös asentaa. Asennus tapahtui päätteessä seuraavalla komennolla.

```
(venv)$ pip install uwsgi
```

Sen jälkeen piti ruokalista- hakemiston juuressa olevaan wsgi.py tiedostoa muokata oikeaan muotoon.

```
from ruokalista import create_app

config_name = 'production'

app = create_app(config_name)

if __name__ == '__main__':
    app.run()
```

Tiedostossa muutettiin config_name arvoksi production eli tuotantokäyttöön. Tämän lisäksi piti luoda uusi tiedosto nimeltään ruokalista.ini ja tallentaa se hakemiston juureen. Ini-tiedostoa tarvittiin uWSGI konfigurointiin. Tähän ini-tiedostoon sitten lisättiin seuraava sisältö.

```
[uwsgi]
module = wsgi:app
master = true
processes = 4
socket = /tmp/ruokalista.sock
chmod-socket = 666
vacuum = true
die-on-term = true
```

Yllä oleva käynnisti uWSGI master- tilassa ja neljällä prosessilla. Socket luotiin väliaikaiseen tmp-hakemistoon ja sille annettin sopivat oikeudet. Vacuum on yhtä kuin tosi eli, jos prosessi pysäytettäisiin niin se poistaisi myös socketinkin. Viimeisenä muokattiin die-on-term arvoksi tosi, jotta uWSGI ja systemd välinen yhteys toimisi. Tällä saatiin niiden välillä haluttu toiminnallisuus.

Tämän jälkeen luotiin järjestelmäpalvelu komennolla

```
$ sudo nano /etc/systemd/system/ruokalista.service
```

Teksieditori loi uuden systemd konfigurointitiedoston projektille ja sinne lisättiin.

```
[Unit]

Description=uWSGI instance to serve ruokalista

After=network.target

[Service]

User=pääkäyttäjä

Group=www-data

WorkingDirectory=/var/www/ruokalista

Environment="PATH=/var/www/ruokalista/venv/bin"

ExecStart=/var/www/ruokalista/venv/bin/uwsgi --ini ruokalista.ini

[Install]

WantedBy=multi-user.target
```

Tiedostossa määriteltiin käyttäjä ja se mihin ryhmään käyttäjä kuuluu. Kohdehakemisto määriteltiin sinne, missä web-sovellus sijaitsee. Ympäristömuuttujaan määriteltiin virtuaaliympäristön hakemisto. Lisäksi sille määriteltiin uwsgi- polku ja ini- tiedosto. Lopuksi määriteltiin servicen käynnistystaso. Tallennettiin se ja uWSGI palvelu täytyi käynnistää sekä ottaa käyttöön komennolla.

```
$ sudo systemctl start ruokalista && sudo systemctl enable ruokalista
```

Tämän jälkeen piti vielä konfiguroida nginx www-palvelimen asetukset kohdilleen nano tekstieditorilla.

```
$ sudo nano /etc/nginx/sites-available/ruokalista
```

Tiedostoon lisättiin seuraavat kohdat.

```
server {

    listen 80;

    server_name ip_osoite/domain;

    location / {

        include uwsgi_params;
```

```
        uwsgi_pass unix:/tmp/ruokalista.sock;

    }

}
```

Tässä määriteltiin mitä porttia kuunnellaan sekä palvelimen ip-osoite tai domain nimi. Location kohdassa määriteltiin uwsgi socketin sijanti. Tämä piti tallentaa ja luotiin symbolinen linkki.

```
$ sudo ln -s /etc/nginx/sites-available/ruokalista /etc/nginx/sites-enabled
```

Seuraavalla komennolla tarkistettiin, että konfiguroinnissa ei tullut virheitä.

```
$ sudo nging -t
```

Jos ongelmia ei ilmentynyt, niin Nginx system service piti käynnistää uusiksi komennolla.

```
$ sudo systemctl restart nginx
```

Näiden lisäksi tarkistettiin Flask web-ohjelman hakemiston oikeudet ja että pääkäyttäjä on www-data ryhmässä. Flask PWA-sovelluksen toiminnallisuus vaati vielä lisää muutoksia teknologian konstruktioon. Toimiakseen PWA-sovelluksen serverin ja käyttäjän välinen tieto tulee olla salattua, yhteyden pitää olla HTTPS muodossa. HTTPS vaatima sertifikaatti toteutettiin Let’s encryptin avulla, se on ilmainen Certificate authority (CA). Toimiakseen Let’s encrypt käyttää ACME protokollaa. Tämän ominaisuuden implementointi toteutettiin seuraavasti. Ensimmäiseksi piti palvelimelle asentaa Certbot-sovellus. Luotiin ssh-yhteys palvelimelle ja lisättiin Certpot ppa repository.

```
$ sudo apt-get install -y software-properties-common

$ sudo add-apt-repository ppa:certbot/certbot

$ sudo apt-get update

$ sudo apt-get install -y python-certbot-nginx
```

Asennuksen jälkeen luotiin sertifikaatti komennolla.

```
$ sudo certbot --nginx -d ruokalista.duckdns.org -d www.ruokalista.duckdns.org
```

Vastattiin kysyttyihin kohtiin ja luotiin sertifikaatit. Tämän jälkeen muokatiin Nginx tiedostoa Certbottia ja Let’s encryptiä tukevaan muotoon komennolla.

```
$ sudo nano /etc/nginx/sites-available/ruokalista
```

Kuvassa 33. on näkyvissä, kuinka server_name on saanut arvokseen web-osoitteen, joka rekisteröitiin DuckDNS palvelusta. Samalla kaikki porttiin 80 tuleville HTTP pyynnöille tuli vastaus koodiksi 301 eli HTTP-portti on poistettu pysyvästi käytöstä. Ne ohjattiin palvelimen HTTPS-porttiin eli porttiin 443.

```
server {
    listen 80;
    listen [::]:80;

    server_name ruokalista.duckdns.org www.ruokalista.duckdns.org;
    return      301 https://$server_name$request_uri;

    location / {
        include uwsgi_params;
        uwsgi_pass unix:/tmp/ruokalista.sock;
    }
}
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;

    server_name ruokalista.duckdns.org www.ruokalista.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/ruokalista.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/ruokalista.duckdns.org/privkey.pem;
    include  /etc/letsencrypt/options-ssl-nginx.conf;
    location / {
        include uwsgi_params;
        uwsgi_pass unix:/tmp/ruokalista.sock;
    }
}
```

Kuva 33. Nginx web palvelimen asetukset Flask PWA-sovellukselle.

Kuvasta 33 nähdään, että HTTPS- asetuksissa kuunnellaan porttia 443, SSL ja HTTP/2 protokollaa, joiden avulla Flask PWA-sovelluksen toiminnallisuus vaatimukset täyttyivät web-palvelimen osalta. Näiden lisäksi täytyi tarkastaa, löytyikö palvelimelta seuraavaa hakemistoa ja sieltä tiedostoa.

```
$ sudo cat /etc/letsencrypt/options-ssl-nginx.conf
```

Jos tiedostoa ei ole, luotiin sellainen komennolla.

```
$ sudo touch /etc/letsencrypt/options-ssl-nginx.conf
```

Jonka jälkeen editoitiin tiedostoa komennolla.

```
$ sudo nano /etc/letsencrypt/options-ssl-nginx.conf
```

Kuvassa 34 nähdään mitä kaikkea sinne täytyi lisätä.

```
# This file contains important security parameters. If you modify this file
# manually, Certbot will be unable to automatically provide future security
# updates. Instead, Certbot will print and log an error message with a path to
# the up-to-date file that you will need to refer to when manually updating
# this file.

ssl_session_cache shared:le_nginx_SSL:1m;
ssl_session_timeout 1440m;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;

ssl_ciphers "ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES1
M-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-SHA256:
-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA
HE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-
56-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS";
```

Kuva 34. Options-ssl-nginx.cong tiedoston sisältö.

Kuvan 34 tiedostoon ei kannata tehdä itse ylimääräisiä muutoksia, jotta Let's encrypt päivitykset toimisivat ongelmitta tulevaisuudessa. Tämän jälkeen piti muistaa tehdä symbolinen linkki saatavilla olevista nginx-sivuista käyttöön otettuihin sivuihin.

```
$ sudo ln -s /etc/nginx/sites-available/ruokalista /etc/nginx/sites-enabled/
```

Luonnollisesti konfiguraation toimivuus testattiin komennolla.

```
$ sudo nginx -t
```

Jos edellinen komento ei aiheuttanut virheitä, niin uudet konfiguraatiot uudelleen ladattiin nginx-palvelu ja tarkastettiin nginx-palvelun tila.

```
$ sudo systemctl reload nginx.service

$ sudo systemctl status nginx.service
```

Sertifikaatin toiminnallisuuden tarkastus voitiin suorittaa sen jälkeen joko selaimella SSLLabs [22]testillä tai sama tarkistus suorittaa komentoriviltä komennolla serverillä.

```
$ sudo openssl x509 -noout -dates -in /etc/letsencrypt/live/ruokalista.duckdns.org/cert.pem
```

Tarkastuksella nähtiin, mihin asti sertifikaatti on voimassa. Let's encrypt sertifikaatti tulisi uusia ennen kuin 90 päivän aikaraja tulee vastaan. Uusimisen pystyy automatisoimaan esimerkiksi lisäämällä tietyllä aikasyklillä tapahtuvaksi cron ajastukseksi. Tosin sertifikaatin uusiminen voidaan myös suorittaa komentoriviltä manuaalisesti.

```
$ sudo -H certbot certonly --standalone -d ruokalista.duckdns.org -d www.ruokalista.duckdns.org
```

Tällöin tulee nginx-palvelu pysäyttää sertifikaatin uusimisen ajaksi. Uudelleen käynnistää sen jälkeen ja tarkastaa uusiutuiko sertifikaatti.

Teknologian konstruktion tämänhetkinen toteutus sallii järjestelmän päivittämisen tai korvaamisen toisella alustalla nopeastikin, kunhan vain kohdejärjestelmässä voidaan edellä esitetyt asiat toteuttaa. Esimerkiksi tuotantopalvelimen Raspberry Pi 1 jouduttiin korvaamaan uudemmalla Raspberry Pi 3:lla, kun edellinen lopetti toimintansa. Tällöin riitti, että muistikortti siirrettiin uudempaan ja virtajohto sekä verkkokaapeli yhdistettiin. Jos taas palvelimen arkkitehtuuri olisi muuttunut, niin asennus voi jonkun verran poiketa edellä esitetystä, koska osassa järjestelmistä on käytössä eri versiota tarvittavista ohjelmista. Niiden konfigurointi voi olla muuttunut versioiden välillä huomattavastikin.

7. Evaluointi

Tässä luvussa evaluoidaan vaatimusmäärittelyssä määritelty ja konstruktiossa toteutettu järjestelmä. Ensiksi arvioidaan analyyttisesti, kuinka hyvin toteutettu järjestelmä palvelee käyttäjäänsä ja kehittäjäänsä nykyisellä toteutuksella. Toiseksi testattiin järjestelmä yksikkötestausta, integraatiotestausta, järjestelmätestausta ja hyväksyntätestausta hyväksi käyttäen.

7.1 Analyyttinen

Analyyttinen evaluointi toteutettiin seuraavalla tavalla. Ensiksi tarkasteltiin Android-sovelluksien toiminnallisuuksien eroavaisuuksia. Tarkastelu tehtiin niin käyttäjän kuin kehittäjän perspektiivistä. Toiseksi tarkasteltiin sitä, miten näiden edellä mainittujen toteutusten tukeminen voitiin toteuttaa palvelimella.

7.1.1 Android-sovelluksien toteutuksen analysointi.

Android-sovelluksen toteutukset ovat toiminnallisuuden kannalta hyvin lähellä toisiaan. Jokainen toteutus vaihtoehto täyttää ongelmakehyksessä esitellyn tulevan järjestelmän määrittelyn ja jokainen toteutus myös ratkaisee ongelman. Jaotellaan ne kuitenkin toteutuksen ja toiminnallisuuden mukaan seuraavasti.

1. Webview-sovellus
2. JSON-sovellus
3. PWA-sovellus.

Näistä jokainen on käyttäjän näkökulmasta hyvin samankaltainen Android-päätelaitteessa käytettäessä. Jokaisella sovelluksella on oma käynnistys ikoni, josta ne käynnistetään. Niiden perusnäkymästä näkee suoraan kulloisenkin päivän ruokalistan. Erot tulevat asennustavasta ja tiedonsiirtomääristä eli siitä, miten paljon dataa ohjelma tyypin käyttäminen vaatii.

Taulukko 1. Käyttäjän näkökulma Android-sovelluksille.

	1. Webview-sovellus	2. JSON-sovellus	3. PWA-sovellus
Asennus	APK-paketti	APK-paketti	Ohjelman web-sivulta
Siirrettävän datan määrä	5251 tavua	297 tavua	6195 tavua

Käyttäjän perspektiivistä katsottuna erilliseen sovelluskauppaan meneminen ja sieltä ohjelman asentaminen vaatii hieman enemmän vaivaa kuin sovelluksen websivulta tapahtuva asentaminen. Siirrettävän datan määrällä JSON-sovellus oli datan käytön kannalta kaikkein paras, koska se vie vain 5,66% Webview-sovelluksen ja 4,79% PWA-sovelluksen vaatimasta tiedonsiirtomäärästä. Toisaalta jokainen sovelluksen toteutus vie hyvin vähän dataa, joten tiedonsiirto perusteisenkaan laskutuksen perusteella käyttäjälle

ei tule suurta laskua kuukausittain. PWA-sovelluksen käyttäjällä on ohjelmasta aina tuorein versio tai ainakin mahdollisuus päivittää siihen suoraan käyttöliittymästä.

Kehittäjän näkökulmasta katsottuna Webview- ja JSON-sovelluksen kehittäminen ja ylläpito sekä päivittäminen ovat vaivalloisempia. Erillisen sovelluksen kehittäminen vaatii lisäresursseja. Lisäksi hän joutuu sovelluksen erikseen lisäämään sovelluskauppaan ja sekin on osassa sovelluskaupoista maksullista, kun taas PWA-sovelluksen ylläpito ja päivitys tapahtuu samalla kuin palvelinsovellusta päivitetään. Toinen tärkeä seikka on PWA-sovelluksen toiminta myös toisilla alustoilla, kuten tietokoneilla ja muilla mobiileilla käyttöjärjestelmillä. Kuten taulukosta 2 huomataan, niin PWA sovellustuki on laajempi kuin Webview- ja JSON-sovelluksilla.

Taulukko 2. Ylläpitäjän näkökulma Android-sovelluksille.

	1. Webview-sovellus	2. JSON-sovellus	3. PWA-sovellus
Ylläpito ja kehitys	Erillinen Android sovellus ja web-sovellus.	Erillinen Android sovellus ja web-sovellus.	Web-sovellus
Tuetut käyttöjärjestelmät	Android ja Chrome OS	Android ja Chrome OS	Android, Chrome OS, Windows, Linux
Tietoturva	HTTPS	HTTPS	HTTPS

Tietoturvan näkökulmasta katsottuna sovelluksilla ei ole suurta eroa. Jokainen niistä hakee tiedon salatun HTTPS-yhteyden yli. Kehittäjän näkökulmasta katsottuna Android-sovellusten tietoturva-vaatimukset eivät ole niin korkeat kuin PWA-sovelluksella. Sen toiminta vaatii palvelimen toteutukselta enemmän kuin Webview- tai JSON-sovellus. Toisaalta PWA-sovelluksen vaatimat tietoturvan parannukset ovat hyödyksi myös muissa toteutuksissa. Tämän työn kirjoitushetkellä PWA-sovellus oli kehittäjän näkökulmasta helpoin ylläpitää ja jatkokehittää. Sen tuki käyttöjärjestelmissä niin mobiileissa että työpöydällä on myös laajin kuten taulukosta 2 nähdään.

7.1.2 Palvelin-sovelluksen toteutuksen analysointi.

Palvelin-sovelluksen toteutus tehtiin ensin ihan perinteisenä web-palvelimena, jossa Nginx toimi www-palvelimena. Sovelluspalvelimena Flask-ohjelmalle käytettiin uWSGI:tä, jonka avulla www-palvelin ja Pythonilla kirjoitettu Flask-ohjelma saatiin keskustelemaan keskenään. Tietokantana Flask-sovellukselle toimi Sqlite3-tietokanta, jonka päivitys ja RSS-tiedon haku sekä jäsennys tietokantaan suoritettiin ajastettuna cron tehtävänä. Tämä toteutuskokonaisuus oli käytössä niin tuotanto- että kehitys-palvelimella. Tuotanto-palvelimella ei vielä otettu PWA-sovelluksen toteutusta käyttöön kokonaisuudessaan push-viestien toiminnallisuudessa olevien ongelmien vuoksi, kuten esimerkiksi tietyillä selaimilla viestit eivät toimineet kuin hetken. Mahdollisen vaihtoehtoisen push-viestipalvelimen testausta jatkettiin kehityspalvelimella. Paras push-viestien vastaanotto oli Firefox-selaimella, käyttöjärjestelmästä riippumatta. Muilla selaimilla push-viestit toimivat yleensä hetken, mutta jossain välissä niiden vastaanotto keskeytyi push-viestien tilaajilla.

Käyttäjän näkökulmasta palvelimen toteutuksella ei ole suurta merkitystä, kunhan käyttäjän tarvitsema tieto oli vain saatavilla tarvittaessa. Ylläpitäjän näkökulmasta tuotantopalvelimen toteutus oli hyvin yksioikoinen. Sen toiminta varmuus on ollut hyvä ja suuremmilta palvelu katkoilta on vältytty. Toteutuksen siirtämistä toisenlaiseen palvelinarkkitehtuuriin ei ole vielä ollut tarvetta, koska nykyisellä käyttäjämäärillä palvelimen kuormitus on ollut sen verran pienehkö.

Konstruktion vaatimusmäärittelyssä esitettyjen palvelimen vaatimusten, kuten järjestelmän saatavuudesta, toteutus tehtiin erillisellä Python-sovelluksella. Tämä sovellus voitiin ajaa joko komentorivipohjaisesti palvelimella tai Python-sovelluksille sopivassa palvelussa. Konstruoidulla ohjelmalla voitiin tarkastaa ylläpitäjän asettaman aikavälin mukaisesti palvelimen otsikko-vastauksen. Vastaus tallennettiin tietokantaan ja jos vastaus oli, että palvelin oli saatavilla, niin tietokantaan tallennettiin totuusarvomuuttujalle tosi. Jos palvelimen vastaus oli jotain muuta kuin haluttu vastaus, niin palvelimelle tallennettiin totuusarvomuuttujalle epätosi. Kun peräkkäisiä epätosiarvoja oli kolme, niin ohjelma lähetti ensimmäisen sähköpostiviestin järjestelmän ylläpitäjälle. Samalla käynnistyi palvelinkohtainen ajastettu tehtävänä, joka lähetti uudelleen sähköpostia ylläpidolle heidän määrittelemän aikataulun mukaisesti.

7.2 Testaus

Konstruktion testaus toteutettiin Flask-ohjelman yksikkötesteillä ja niiden testikattavuudella. Integraation ja järjestelmä testausta suoritettiin Python-koodilla, Postman[23]-ohjelmalla ja komentoriviltä cURL-käskyjä tekemällä. Android-ohjelmien testaus suoritettiin usealla eri päätelaitteilla ja Android-versiolla. Itse Android-sovelluksien testausta ohjelmallisesti ei suoritettu, koska käytännössä ohjelmalla oli ns. kaksi tilaa, toisessa se näytti halutun tiedon käyttäjälle ja toisessa se antoi virhe ilmoituksen yhteyden toimimattomuudesta palvelimelle. PWA-sovellus testattiin Android-päätelaitteilla ja eri selaimilla sekä Chromen kehitystyökaluista löytyvällä Lighthouse[24]-sovelluksella, joka on tarkoitettu web-sivujen testaukseen. Sen avulla voitiin todentaa, toteutuksen tehokkuus, helppokäyttöisyys, parhaat käytännöt, hakukone optimointi ja PWA-sovelluksen vaatimukset. Lisäksi Flask-ohjelman tuotanto- ja kehityspalvelimen websivut testattiin sitespeed.io[25] hyväksi käyttäen, jotta mahdolliset yhtäläisyydet ja eroavaisuudet Webview- ja PWA-sovellusten toteutukselle löydettäisiin. Robot frameworkin avulla testattiin Flask-sovelluksen toimintaa järjestelmätestauksen osalta.

7.2.1 Yksikkötestaus

Yksikkötestauksella testattiin Flask-sovelluksen Python-koodia. Esimerkkinä testattavasta funktiosta Flask-ohjelmassa voidaan nähdä kuvasta 35. Kyseinen funktio tarkastaa totuusarvon sisään tulevasta syötteestä

```
def has_numbers(input):
    """Search for numbers."""
    return bool(re.search(r'\d', input))
```

Kuva 35. Yksikkötestauksen kohteena oleva funktio.

Kuvan 35 funktio yksikkötestattiin kuvan 36 testillä.

```
def test_home_views_has_numbers(self):
    """Functio test for has numbers."""
    self.assertTrue(has_numbers('1Foo'))
```

Kuva 36. Yksikkötesti funktiolle.

Kuten kuvasta 36 nähdään, testissä syötetään syöte, jossa on numero. Testi palauttaa ok, kun ehto on tosi, niin kuin tässä tapauksessa.

Tällä hetkellä yksikkötestejä on 37 kpl ja niiden avulla saatiin testikattavuudeksi 98%, kuten kuvasta 37 voidaan nähdä. Flask-sovelluksessa oli yhteensä kaksitoista Python-tiedostoa. Eri tiedostojen testauksen kattavuusprosentti vaihteli 91-100% välillä.

Coverage report: 98%

Module ↓	statements	missing	excluded	coverage
config.py	24	1	0	96%
ruokalista/__init__.py	95	1	0	99%
ruokalista/api/__init__.py	3	0	0	100%
ruokalista/api/views.py	182	8	0	96%
ruokalista/fetcher.py	36	0	0	100%
ruokalista/home/__init__.py	3	0	0	100%
ruokalista/home/views.py	119	5	0	96%
ruokalista/menulist/__init__.py	3	0	0	100%
ruokalista/menulist/views.py	43	0	0	100%
ruokalista/models.py	11	1	0	91%
ruokalista/redis.py	3	0	0	100%
test_app.py	246	2	0	99%
Total	**768**	**18**	**0**	**98%**

Kuva 37. Flask-sovelluksen yksikkötestien kattavuus.

Kuvasta 37 nähdään, että Flask-sovelluksen lausekemäärä on 768, joista 18:aa lausetta ei testattu. Yksikkötestit ajettiin kehitysympäristössä, jossa web-serverinä toimi Flask-sovelluksen kehitys web-serveri testikonfiguraatiolla. Testin suorittamisen avuksi kehitettiin Makefile-konfiguraatio, jonka avulle tarvittavat parametrit voitiin helposti ottaa käyttöön. Esimerkiksi yksikkötestaus suoritettiin seuraavalla komennolla.

```
(venv)$ make run.testapi
```

Tämän jälkeen luotiin kattavuusraportti projektille komennoilla.

```
(venv)$ make run.coverage
```

```
(venv)$ coverage html
```

Edellisten komentojen avulla saatiin yksikkötestin kattavuus HTML-kuten kuvasta 37 nähdään. Yksikkötestien avulla testattiin yksittäisten apufunktioden toiminnallisuuden oikeellisuutta Flask-ohjelman osana, sekä niiden avulla testattiin myös Flask-ohjelman

toteutusta. Regressiotestauksen avulla yksikkötestaukset suoritettiin sitä mukaan uudelleen, kun Flask-ohjelman toiminallisuus muuttui.

7.2.2 Integraatiotestaus

Integraatiotestauksessa käytettiin hyväksi samoja yksikkötestejä kuin luvussa 7.2.1 ja niiden lisäksi testaustasoa laajennettiin koskemaan esimerkiksi push-viestien tilausten API-rajapintaa vastaan, jolloin pystyttiin testaamaan Flask-ohjelman ja Redis avain-arvo-tietokannan integraatiota sekä web-selaimen integraatiota. Tällaisen isomman kokonaisuuden testaamisesta esimerkki on kuvasta 38, siinä esitetään Pythonilla kirjoitettu testi, jossa testattiin push-viestien tilauksen peruutus ilman olemassa olevaa asiakas avainta, jonka ei pitäisi olla selaimesta mahdollista suoraan Flask-ohjelman käyttöliittymästä. Silti sen API-rajapinnan integraatiotestaaminen ja sopivan vastauskoodin palauttaminen kyselyyn tuli testata, koska kyseisen API-rajapinnan kanssa voidaan keskustella myös muillakin ohjelmilla kuin web-selaimessa toimivalla Flask-ohjelmalla tai Android-sovelluksilla.

```python
def test_post_api_unsubscribe_without_key(self):
    """Test unsubscribe without a key."""
    logger.debug(" test_post_api_unsubscribe_without_key @ TestApis")

    headers = {'Content-Type': 'application/json',
               'Cache-Control': 'no-cache'}
    data = {"client_uuid": None}
    response = self.client.post(url_for('api.unsubscribe'),
                                headers=headers,
                                data=json.dumps(data))

    assert response.status_code == 400
    assert response.json == {'message': 'client_uuid is required'}
```

Kuva 38. Pythonilla integraatiotesti tilauksen peruutus ilman asiakas avainta.

Integraatiotestillä testattiin kehitysympäristössä, mitä palvelin palauttaa kyselyyn, kuten kuvasta 38 nähdään. Testissä testattiin kaksi asiaa kerralla. Ensiksi testattiin vastauskoodi palvelimelta, jonka tulisi olla 400 eli viallinen pyyntö. Toiseksi testattiin JSON-viesti vastaus, joka kertoi, että käyttäjän yksikäsitteinen tunniste tarvittaisiin tilauksen perumiseen. Samalla tavalla samainen toiminnallisuus testattiin Postman-ohjelmalla, joka näkyy kuvasta 39.

Kuva 39. Postman ohjelmalla tehty integraatiotesti kehitysympäristössä.

Kuten kuvasta 39 nähdään, testit ovat samanlaiset kuin kuvan 38 Python-koodilla tehdyissä testeissä. Postmanissa testit kirjoitettiin JavaScriptillä joko käsin tai valmiita pohjia hyväksikäyttäen. Testien vastaukset Postman-ohjelmalla olivat identtiset niin palvelimen vastauskoodi, että JSON-viestin osalta. Lisäksi testattiin samainen integraation toiminnallisuus komentokehotteessa cURL-komentona niin kuin kuvassa 40 nähdään.

```
teemu@EliteBook-2570p:~$ curl -H "Content-Type: application/json" -d '{"client_uuid":""}' -X POST http://127.0.0.1:5000/api/unsubscribe
{
  "message": "client_uuid is required"
}
```

Kuva 40. Komentokehotteessa suoritettu cURL integraatiotesti kehitysympäristössä.

Vastaus web-palvelimelta on samanlainen kuin edellisissäkin testeissä, silloin kun yksikäsitteinen tunnistekoodi puuttuu asiakkaalta (kuva 40). Testattaessa http-vastausta komentoriviltä cURL-komennolla, nähdään kuvasta 41 kehitysympäristössä tehty kysely.

```
teemu@EliteBook-2570p:~$ curl -s -o /dev/null/ -w "\n\n%{http_code}\n\n" -H 'Content-Type: application/json' -H 'Cache-Control: no-cache' -d '{"client
_uuid":""}' -X POST http://127.0.0.1:5000/api/unsubscribe


400
```

Kuva 41. Komentokehotteessa suoritettu cURL HTTP-koodi kysely ja vastaus kehitysympäristössä.

Kuvasta 41 nähdään myös, että vastaukseksi saatiin HTTP-koodi 400 palvelimelle tehtyyn kyselyyn, niin kuin pitikin.

Näiden edelle esitettyjen esimerkkien lisäksi integraatiota ja rajapintojen keskinäistä kommunikointia testattiin sitä mukaan, kun erilaiset vaihtoehtoiset toteutukset Android-sovelluksen konstruktiosta toteutettiin ja integroitiin järjestelmään.

7.2.3 Järjestelmätestaus

Kokonaisuuden testaus niin kehitys- ja tuotantoympäristössä toteutettiin Robot Frameworkia hyväksi käyttäen. Esimerkiksi, kun palvelimella olevaa järjestelmää päivitettiin ensiksi kehitysympäristössä, järjestelmätestattiin jo aikaisemmin toteutetut ominaisuudet ja samalla testattiin myös uudet ominaisuudet, jotta voitiin todentaa kokonaisen järjestelmän toiminnallisuus. Kun testit suoritettiin ja saatiin oikeanlaiset vastaukset, samat toiminnallisuudet voitiin päivittää myös tuotantoympäristöön. Samat testit ajettiin myös tuotantoympäristölle. Jos järjestelmätestit menivät läpi, niin integroidut ominaisuudet jäivät käyttöön.

Robot Framework-testit ovat muodoltaan samankaltaisia kuin kuvan 42 testi, missä testattiin **tietoa-sivun** toiminta. Ensiksi testissä yksilöidään testin asetukset esimerkiksi lyhyt kuvaus, mitä tässä testissä testataan. Seuraavaksi määritellään mitä resursseja testissä käytetään. Lopuksi määritellään mitä testin jälkeen tehdään. Testitapaus kuvataan seuraavaksi, ensiksi testi avaa palvelimen indeksisivun. Sen jälkeen selain siirtyy **tietoa-sivulle** ja lopputuloksena pitäisi olla avoin **tietoa-sivu**.

```
1   *** Settings ***
2   Documentation     A test suite with a single test for Tietoa page.
3   ...
4   ...               This test has a workflow that is created using keywords in
5   ...               the imported resource file.
6   Resource          resource.robot
7   Test Teardown     Close Browser
8
9   *** Test Cases ***
10  Valid About page
11      Open Browser To Index Page
12      Go To Tietoa Page
13      Tietoa Page Should Be Open
```

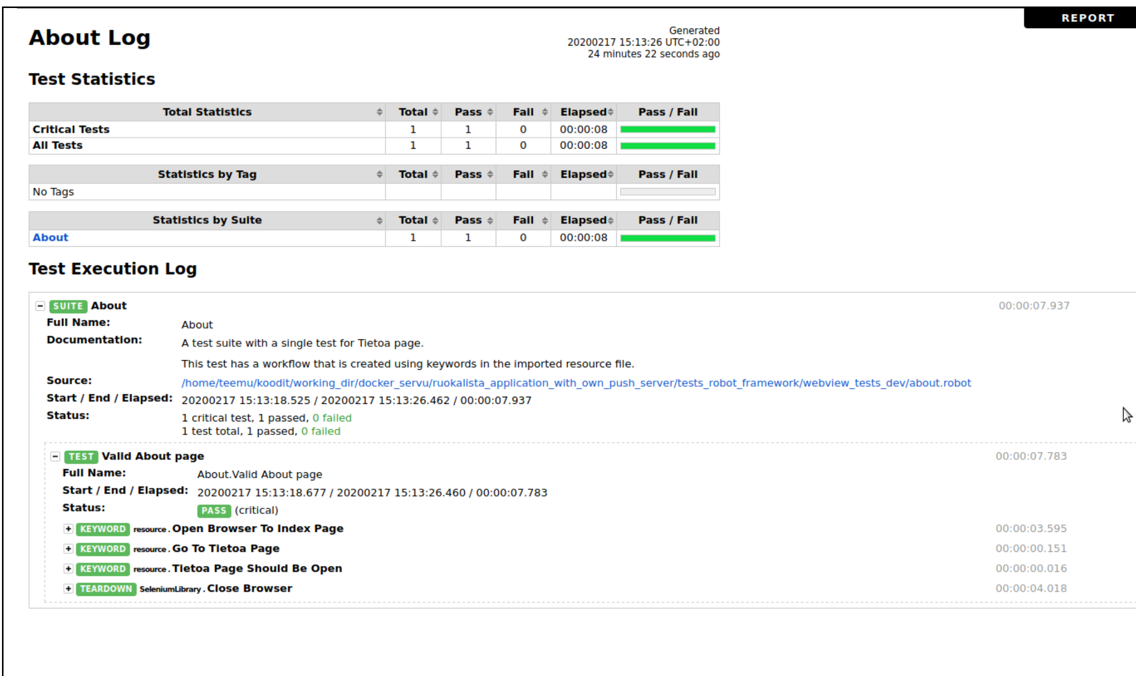
Kuva 42. Robot Framework testi tietoa-sivulle.

Kuvan 42 testi suoritetaan yleensä osana suurempaa kokonaisuutta, mutta käytännössä se voidaan suorittaa yksittäinkin kuten kuvasta 43 nähdään.

```
robot --outputdir result_0170220 webview_tests_dev/about.robot
==============================================================================
About :: A test suite with a single test for Tietoa page.
==============================================================================
Valid About page                                                      | PASS |
------------------------------------------------------------------------------
About :: A test suite with a single test for Tietoa page.             | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
```

Kuva 43. Robot Framework testin suoritus terminaalissa.

Kuten kuva 43 kertoo myös, että testin suoritus on hyväksytysti suoritettu. Suorituksen aloitus käynnisti selaimen ja suoritti testin kuvauksessa esitetyt kohdat. Kun itse testi oli suoritettu, generoituu testistä kolme erillistä lokitiedostoa. Kuvasta 44 nähdään esimerkki log.html tiedostosta selaimessa.



Kuva 44. Robot Framework log.html tiedoston raportti selaimessa.

Kuvasta 44 nähdään Robot Frameworkin tekemä **tietoa-sivun** testin tulos. Näin yksittäisen muutoksen toiminta voitiin järjestelmätestinä testata joko automaattisesti tai yksittäin manuaalisesti, esimerkiksi kun tehtiin muutoksia palvelimen toteutukseen.

7.2.4 Hyväksyntätestaus

Hyväksyntätestaus toteutettiin ohjelmiston vaatimusmäärittelyn ja yleisten ohjelmiston laatuominaisuuksia testaten, näihin kuuluu mm. seuraavat kohdat

- virheettömyys
- käytettävyys
- suorituskyky
- luotettavuus
- tietoturvallisuus
- siirrettävyys
- yhteensopivuus
- ylläpidettävyys

Konstruktion vaatimusmäärittelyssä Android-sovellukselle määriteltiin käyttäjän halu saada tietää arkipäivän ruokalista koululla (kuva 6). Tämän tiedon käyttäjä saa, kun hän käynnistää minkä tahansa version Android-sovelluksista. Jokaisessa vaihtoehtoisessa sovelluksessa hänelle näytetään kaikki kaksi mahdollista ruokalista vaihtoehtoa. Virheettömyyteen vaikuttavat tekijät ovat nähtävillä taulukossa 3. Tiedon virheettömyyteen vaikuttaa ulkoisia tekijöitä, jotka on käyttäjän ja ohjelman ylläpitäjän tiedostettava. Ruokalistojen tiedot koostetaan ruokatoimittajan omasta RSS-syötteestä ja muutetaan käyttäjän päätelaitteelle sopivaan muotoon. Suurin yksittäinen vaikuttaja virheettömyyteen tulee RSS-syötteestä. Sen olemassaolo ja muoto ovat muuttaneet muotoa kehityksen aikana useampaan otteeseen. Loppujen lopuksi koko RSS-syötteen olemassa olo loppui. Muutokseen on pyritty mukautumaan aina muutoksen tapahtuessa mahdollisimman nopeasti. Toisaalta käyttäjän velvollisuus on tarkastaa mihin ruokalistan ryhmään hänen koulunsa tai tarhansa kuuluu. Toisin sanoen, on osattava lukea oikein näkymää.

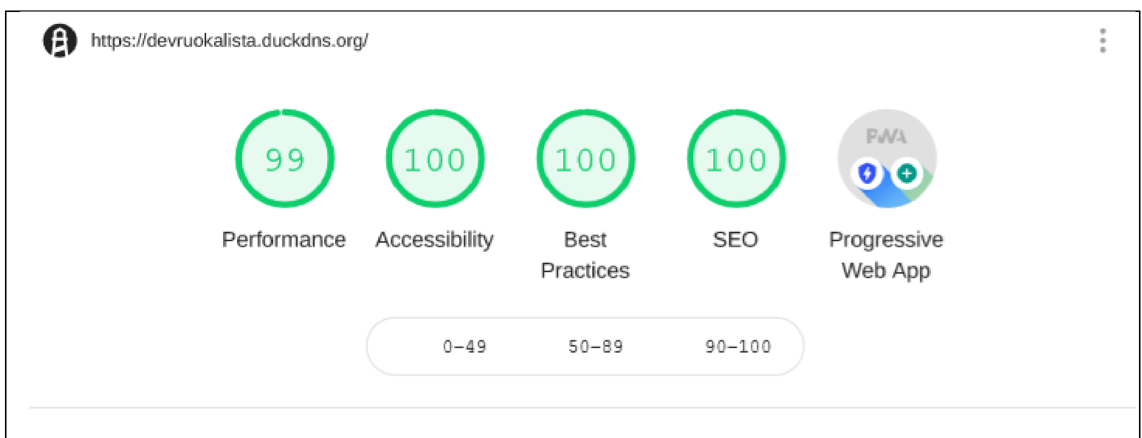
Taulukko 3. Android sovelluksen virheettömyys.

	1. Webview-sovellus	2. JSON-sovellus	3. PWA-sovellus
Virheet tömyys	Mikäli palvelimelta tuleva tieto on ajan tasalla. Käyttäjä tietää mikä on hänelle oikea lista.	Mikäli palvelimelta tuleva tieto on ajan tasalla. Käyttäjä tietää mikä on hänelle oikea lista.	Mikäli palvelimelta tuleva tieto on ajan tasalla. Käyttäjä tietää mikä on hänelle oikea lista.
Huomi oita	Ruokana voi olla jotain muutakin, kuin listalla pitäisi olla.	Ruokana voi olla jotain muutakin, kuin listalla pitäisi olla.	Ruokana voi olla jotain muutakin, kuin listalla pitäisi olla.

			Jos sovelluksen service worker on päivittänyt sovelluksen tiedot taustalla.

Kuten taulukosta 3 voidaan nähdä, virheettömyyteen vaikuttaa ulkoisia tekijöitä, kuten keittiö on voinut toteuttaa jotain muutakin tai koululla voi olla yleisestä listasta poikkeava ruoka. Lisäksi PWA-sovelluksen virheettömyyteen vaikuttaa service workerin toteutuksen rajoitukset, koska se näyttää päätelaitteella jo olemassa olevan tiedon ennen kuin hakee uudemman tiedon palvelimelta. Se päivittääkö käyttäjä näkymänsä siinä vaiheessa, kun uusi tieto on haettu, vaikuttaa tiedon virheettömyyteen.

Käytettävyys jokaisella toteutuksella on hyvin samankaltainen. Tärkein tieto pitäisi olla käyttöliittymässä näkyvillä ensimmäisenä. Sovelluksille ei erikseen tehty käytettävyys-testausta, mutta erilaisia apuvälineitä käyttämällä niistä pyrittiin tekemään mahdollisimman helppokäyttöisiä. Webview- ja PWA-sovellukset testattiin Chromen kehitystyökaluista löytyvää Lighthousea apuna käyttäen. Testauksen perusteella mm. käyttöliittymää parannettiin, jotta helppokäyttöisyys paranisi erilaisilla päätelaitteilla. Kuten kuvasta 45 nähdään, kehitetyn sovelluksen toteutuksen optimoinnissa onnistuttiin hyvin ainakin numeerisesti mittaamalla.

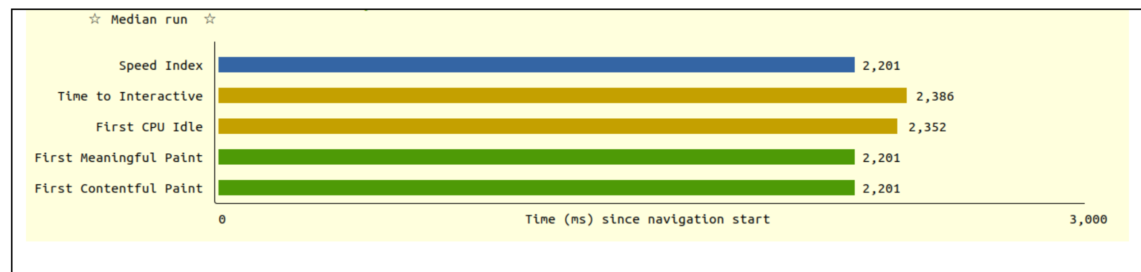


Kuva 45. Lighthousen yhteenveto PWA-sovelluksesta.

Suorituskykyä mitattaessa mittarina olivat mm seuraavat kohdat:

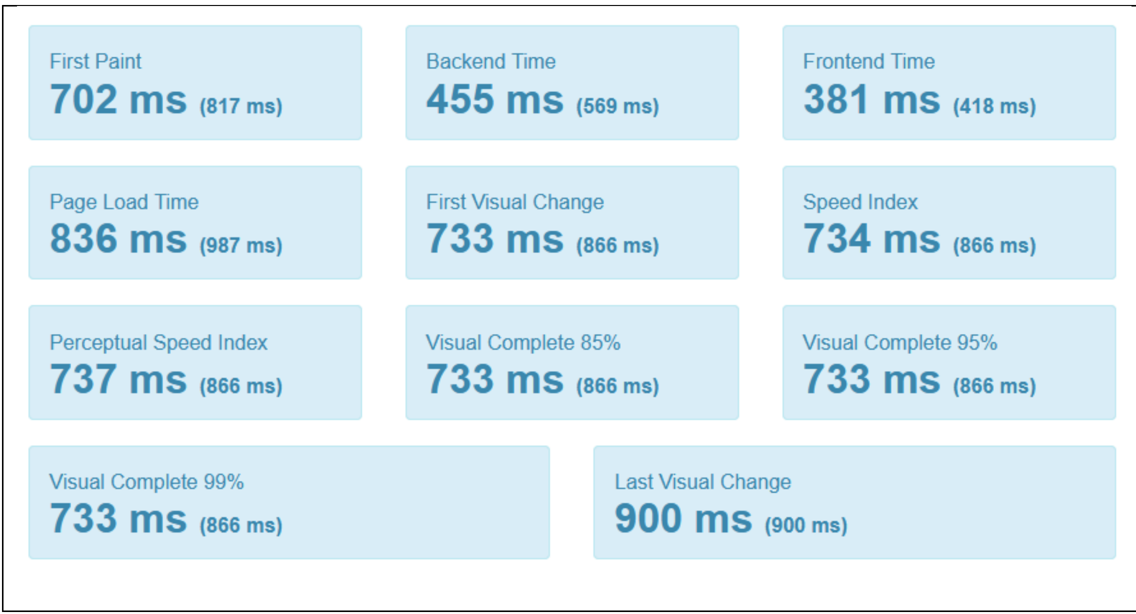
1. Kuinka nopeasti ensimmäinen merkittävä teksti tai kuva on käyttöliittymässä näkyvillä?
2. Kuinka nopeasti sivun sisältö näkyvästi täytetty?
3. Miten pian sivu on interaktiivinen käytettäväksi?
4. Miten nopeasti sivu on valmis ottamaan syötettä vastaan?
5. Maksimaalinen ensimmäisen syötteen viive?

Kuvan 45 suorituskykymittaukset näkyvät tarkemmin kuvasta 46.



Kuva 46. Pwmetrics viiden testi suorituksen keskiarvot.

Testin tulokset ovat viiden testikerran keskiarvoja. Näin virhemarginaali ja yhden testikerran vaikutus minimoitiin. Käytännössä testit suoritettiin myös suuremmalla testimäärällä, kun kartoitettiin sopivaa testimäärä. Niiden perusteella sopivaksi testi määräksi tuli viisi. Jokaisen edellä mainittujen suureiden optimoinnin avulla sovelluksesta saatiin käyttäjälle parempi käyttökokemus, kun esimerkiksi käyttäjän ei tarvitse odottaa tiedon saatavuutta, vaan se on saatavilla mahdollisimman nopeasti. Samalla kehittäjälle selviää mahdolliset parannuskohteet sovelluksen käyttöliittymässä. Tämän lisäksi suorituskykyä testattiin muillakin tavoilla. Esimerkki sitespeed.io-mittauksella mitattiin PWA-sovelluksen suorituskykyä ja mittaus tulokset ovat näkyvillä kuvassa 47. Sen perusteella sivun ensimmäinen näkymä on käyttäjän nähtävissä 702ms(millisekunti) kuluessa. Sivun latautuminen niin palvelimella 455ms ja selaimessa 381ms kestää yhteensä 836ms. Kokonaisuuden viimeinen näkyvä muutos on valmiina 900ms:n kuluttua käyttäjän käyttöliittymässä. Tuloksia voi pitää hyvinä, tosin pitää muistaa, että nämä tulokset ovat huomattavasti parempia, mitä Lighthousella tehdyissä arvoissa. Lighthousen mittaus tekee perusasetuksilla simulointia. Sen mittauksessa verkkoyhteyttä simuloidaan mobiiliverkkoa vastaavaksi. Lisäksi päätelaitteena mittauksessa on matkapuhelin ja testejä hidastetaan neljännekseen. Kun otetaan huomioon simuloinnin vaikutukset, molemmilla tavoilla mitatut tulokset ovat keskenään vertailukelpoisia.



Kuva 47. Sitespeed.io mittaukset PWA-sovelluksen suorituskyvystä.

Kun sitespeed.io mittauksia tarkastellaan, niiden perusteella Webview- ja PWA-sovelluksien vasteaika on hyvä, varsinkin kun otetaan huomioon serverin ja sen yhteyden rajoitteet. Avoimen lähdekoodin testaustyökalulla Locust:lla [26]testattiin nykyisen palvelimen rajoja, mutta niiden perusteella ei ainakaan tässä vaiheessa ole mitään tarvetta muuttaa palvelimen konstruktiota, koska käyttäjämäärät ovat vähäisiä ja heidän aiheuttaman kuorma serverille on pieni.

Luotettavuus on useamman osan yhteissumma. Siihen kuuluvat vakaus, mikä tarkoittaa, että ohjelma ei saisi kaatua tai aiheuttaa toisten sovellusten kaatumisia. Android-sovelluksien käytöstä ei ole tullut palautetta, että ohjelma kaatuisi tai aiheuttaisi toisten ohjelmien kaatumisia. PWA-sovelluksen jumiutumisia ei ole tullut vastaan yleisimmillä selaimilla käytettäessä. Asennettaessa Android-päätelaitteeseen PWA-sovelluksen toiminta on ollut tarkoituksen mukaista. Toinen luotettavuuden mittari on, miten sovellus reagoi aavistettaviin ja aavistamattomiin virheisiin. Käytännössä sovellukset ohjaavat tai kertovat käyttäjälle, miten toimia, jos hän on kohdannut jonkin virheen. Virheen sattuessa Webview- ja PWA-sovellukset antavat virheviestin, joka sitten ohjaa käyttäjän sivuston indeksi sivulle. Kuormituksen kesto on kolmas luetettavuuden mittari. Sinänsä ohjelma ei ole millään tavallaan kriittinen. Käyttäjä saa tiedon vain hitaammin, jos palvelin kuormittuu liikaa. Neljäntenä kohtana luotettavuudelle voidaan pitää toipuminen virhetilanteesta. Sovelluksen käyttäjä päivittää sivun/sisällön ja yleensä ongelma poistuu sillä. Tiedon yhtenäisyys on yksi luotettavuuden mittareista. Käyttäjän näkökulmasta, jokainen konstruktoitu sovellus käyttää samaa palvelinta tietolähteenä. Käyttäjä ei itse pääse muuttamaan tietoja, ainoastaan kuluttamaan niitä. Ylläpitäjän näkökulmasta tiedon yhtenäisyys riippuu ulkopuolisesta tietolähteen oikeellisuudesta. Turvallisuus on myös yksi osa luotettavuutta, koska sovellus ei saa aiheuttaa ihmisille tai omaisuudelle vaaraa. Tuhosta palautuminen on yksinkertainen, koska silloin sovellus on vain käynnistettävä uudelleen. Uskottavuus on myös yksi luettavuuden tekijöistä, toimiiko sovellukset yhtenäisesti, aavistettavasti ja luotettavasti. Jokainen sovelluksen versio toimii samantapaisesti, erot tulevat lähinnä käyttöliittymästä, mutta data on yhtenäinen jokaiselle versiolle.

Tietoturvallisuuteen kuuluu myös monta erinäistä osa-aluetta. Kuten jo luvussa 2.1.3 on esitetty, tietoturva aspekti on tärkeysjärjestyksessä hyvin korkealla tässä konstruktiossa. PWA-sovelluksen toiminnallisuuden vaatimukset pakottivat kehittämään järjestelmää siten, että palvelimen yhteydet toimivat HTTP/2-standardin ja HTTPS-yhteyden yli. Tämä vaikuttaa myös Android-sovellusten tietoturvaan positiivisesti. Näin ollen jokaisesta ohjelma vaihtoehdosta saatiin tietoturvallisempia tiedonsiirron osalta. Yksityisyys on yksi osa-alue tietoturvassa ja sen takia Android-sovelluksen toteutuksessa ei käytetty valmiita palveluita, joiden avulla voitaisiin käyttäjistä tietää mahdollisimman paljon. PWA-sovelluksen push-viestien tilaaminen vaatii, että jokaisen ohjelman käyttäjät voidaan identifioida. Tosin tässä konstruktiossa identifioinnin avulla heille vain voitiin viestittää yleisen tason viestejä. Kehittäjän näkökulmasta tietoturvan ylläpidon helppous oli myös yksi merkittävä tekijä. Nykyisen palvelimen arkkitehtuuri ja mahdolliset tietoturvakorjaukset voidaan toteuttaa päivittämällä ainoastaan palvelimen toteutusta.

Siirrettävyydestä ja yhteensopivuudesta voidaan todeta, että molemmat Webview- ja JSON-sovellukset toimivat vain Android-päätelaitteilla. Niiden siirrettävyys on rajoittunut muille alustoille, vaikka se on mahdollista esimerkiksi emulaattorin avulla. Yhteensopivuus eri Android-laitteiden kanssa on tuettu uusimpaan Android versioon saakka. PWA-sovellus toimii muissakin laitteissa, joihin kuuluu tietokoneiden käyttöjärjestelmät ja muutkin matkapuhelin käyttöjärjestelmät. Niissä on vielä tässä vaiheessa jonkin verran eroavaisuuksia toiminnallisuudessa, mutta ne ovat kuitenkin asennettavissa niille. Palvelimen konstruktiokokonaisuus on toteutettu siten, että sen API-rajapintaa käyttämällä voidaan toteuttaa erillinen sovellus muillekin alustoille tarvittaessa, kunhan vain JSON-datan vastaanotto ja näyttäminen on mahdollista. Palvelimen siirrettävyys toiselle Linux jakelulle on mahdollista ja tarvittaessa se pystytään myös ajamaan Docker-konttina.

Ylläpidettävyys on käytännössä hyvin yksinkertainen. Jos jompikumpi Android-sovelluksista lisätään Googlen Play kauppaan, niin käyttäjä saa ilmoituksen päivitetystä versiosta ja voi halutessaan ladata sen. PWA-sovelluksen uusin version latautuu aina, kun service-worker päivittyy palvelimella. Tällöin käyttäjällä on uusin versio käytettävissään aina, kun hän käynnistää sovelluksen uudestaan. Ylläpitäjän kannalta palvelimen konstruktointi voidaan toteuttaa monella eri alustalla ja näin ollen sen varsinaiset ylläpidolliset tehtävät on automatisoitu alustalla ajettaviin scripteihin, mitkä hoitavat päivittämiseen ja virheiden raportoimisen ylläpitäjälle.

Kokonaisuutena kehitetystä järjestelmästä ja sen osa-alueista voidaan todeta evaluoinnin perusteella, että kehitetty järjestelmä toteutti vaatimusmääritellyssä ja ongelmakehyksessä esitetyn ongelman. Evaluoinnin perusteella PWA-sovelluksen on paras konstruktio. Sen ylläpidettävyys on helpoin kehittäjän näkökulmasta ja käyttäjän näkökulmasta sen tuki on laajin eri alustoilla.

8. Pohdinta

Tässä luvussa käydään läpi, sopiiko konstruktiivinen tutkimusmenetelmä Android-ohjelman suunnitteluun, määrittelyyn ja toteutukseen sekä evaluointiin suhteessa aikaisempaan tutkimukseen.

Kuvan 5 konstruktion prosessikaaviossa, ongelma on lähtökohta, joka tulee ratkaista. Tässä työssä lähtökohtana oli ongelmakehys, missä ensiksi esiteltiin ongelma, joka piti ratkaista konstruktiivista menetelmää apuna käyttäen. Ongelmakehyksessä esiteltiin tutkimuskysymykset, ongelman esittely, olemassa olevan järjestelmän spesifiointi ja mahdollisen tulevan järjestelmän spesifiointi. Nunamaker ja kumppanit (1990) totesivat hyvin samantapaisesti, että täytyy luoda käsitteellinen kehys ensin. Saman suuntaisesti totesivat myös Kasanen ja muut (1993), heidän mukaansa konstruktio aloitetaan käytännönläheisestä ongelmasta, jota voidaan tutkia. Hevner ja kumppanit (2004) esittivät, että ongelma voidaan määritellä nykyisen ja tulevan järjestelmän eroina. Hyvin samaan tapaan Peffers ja kumppanit (2007) esittivät ensimmäiseen vaiheeseen kuuluvan ongelman määrittelyn ja sen ratkaisun avulla saavutettavana toteutuksena. Konstruktion tutkimukseen iteratiivisuus tuli tässä työssä esille, kun järjestelmää kehitettiin. Aikaisemman tutkimuksen kirjallisuuteen jouduttiin palamaan, kun järjestelmän kehityksessä täytyi löytää mahdollisia ratkaisutapoja kohdattuun ongelmaan.

Kuvassa 5 seuraavana vaiheena oli määrittely, se toteutettiin tässä työssä ketteristä menetelmistä löytyvien indeksikorttien avulla. Nunamaker ja kumppanit (1990) totesivat vaatimusmäärittelystä, että ne pitää tarkentaa ja niitä pitää pystyä mittamaan sekä evaluoimaan myöhemmin. Hevner ja kumppanit (2004) vuorostaan totesivat vaatimusmäärittelyn muodostuvan liiketoimintaympäristön avulla, jonka pohjalta kehitettävä artefakti myöhemmin evaluoidaan. Siinä vaiheessa, kun kehitettävän järjestelmän vaatimukset ja rajoitteet toteutuvat, kehitettävä artefakti on valmis. Peffers ja kumppanit (2007) totesivat, että konstruktiivinen tutkimusmenetelmä tukee vaatimusmääritelmää, mutta he eivät määritelleet kuinka se tulisi toteuttaa, vaan kuuluvan osana kehitykseen. Konstruktiivisen tutkimusmenetelmän kirjallisuus siis tunnistaa vaatimusmäärittelyn osaksi tietojärjestelmien kehitystä, mutta miten se toteutetaan tutkimuksessa, joutuu jokainen tutkija itse päättämään omassa tutkimuksessaan.

Konstruktion prosessikaaviossa seuraavana vaiheena oli konstruktio (kuva 5). Nunamaker ja kumppanit (1990) totesivat, että ensimmäiseksi tulee kehittää järjestelmän arkkitehtuuri sekä määritellä tulevan järjestelmän komponentit ja niiden yhteydet. Tässä työssä alkuvaiheen arkkitehtuurin määrittely oli karkea hahmotelma järjestelmän toteutuksesta, joka tarkentui työn edetessä moneen otteeseen. Hevner ja kumppanit (2004) totesivat konstruktion olevan iteratiivinen hakuprosessi. Kasanen ja kumppanit (1993) totesivat konstruktion olevan heuristinen, Molemmat niin iteratiivisyys että heuristisyys tulivat ilmi tässä työssä, kun järjestelmän toteutuksen vaihtoehtoja konstruktoitiin. Peffers ja kumppanit (2007) totesivat, että konstruktion pitäisi pystyä demonstroimaan toteutuksen kyvyn ratkaista ongelma. Tämäkin tuli ilmi jokaisella konstruktoidulla Android-sovelluksen toteutuksella. Nunamakerin ja kumppanien (1990) mukaan osana konstruktiota oli myös vaihtoehtoisten toteutuksien tekeminen, joiden perusteella sitten valitaan yksi toteutukseen. Tässä työssä myös toteutettiin vaihtoehtoisetkin toteutukset. Heidän tutkimuksessaan prototyypin avulla voidaan löytää mahdolliset ongelma-alueet toteutuksessa, joka piti paikkaansa tässäkin työssä. Se näkyi selvästi, kun ensimmäisenä toteutettu Webview-sovellus ei tukenutkaan suorilta push-viestejä ja niiden välitystä päätelaitteelle. Peffers ja kumppanit (2007) totesivat, että

Artefaktin demonstrointi voidaan toteuttaa esimerkin avulla ja tässä työssä huomattiin sen ajatuksen ja toteutuksen toimivan hyvin, koska näin voidaan osoittaa miksi ja miten artefaktin kyseiseen ratkaisuun lopulta päädyttiin.

Konstruktion prosessikaaviossa seuraava vaihe on evaluointi (kuva 5). Konstruktiivisen tutkimuksen evaluointi Nunamakerin ja kumppanien (1990) mukaan toteutetaan, kun konstruktio on rakennettu ja sen tehokkuus voidaan testata niillä määreillä, mitä vaatimusmäärittelyssä on määritelty. Tämän perusteella järjestelmää voidaan sitten jatkokehittää tarpeen mukaan. Kun taas Kasanen ja kumppanit (1993) totesivat evaluoinnista, että järjestelmää pitää tutkia ratkaisun soveltuvuuden mukaan. Heidän mielestään käytännönläheinen käytettävyys on yksi tärkeimmistä mittareista, joka näyttää todenmukaisesti kehitetyn konstruktion toteutuksen. Tässä työssä kehitetty järjestelmä evaluoitiin analyyttisesti, jossa muun muassa jokaisen Android-toteutuksen mittareina käytettiin tehokkuutta ja ylläpidettävyyden helppoutta. Ratkaisun soveltuvuutta tarkasteltiin esimerkiksi tuettujen käyttöjärjestelmien mukaan ja todettiin esimerkiksi PWA-sovelluksen toimivan useimmilla järjestelmistä hyvin samaan tapaan mitä Android-alustallakin. Se oli myös tietoturvan kannalta hyvä ratkaisu, koska näin ollen järjestelmään jouduttiin toteuttamaan ominaisuuksia, jotka paransivat myös muiden toteutuksien tietoturvaa. Hevner ja kumppanit (2004) totesivat evaluoinnista, että esimerkiksi voidaan käyttää funktionaalista ja rakenteellista testausta. Tässä työssä toteutettiin järjestelmäntestaus, niin yksikkö-, integraatio-, järjestelmä- ja hyväksyntätestauksen tasoilla. Tällä voitiin osoittaa, että kehitetty järjestelmä eli artefakti on toimiva ja osoitettiin toiminnallisuuden oikeellisuus. Peffers ja kumppanit (2007) totesivat, että artefaktia tulee tarkkailla ja mitata miten hyvin se ratkaisee ongelmakehyksessä esitetyn ongelman. Esimerkkinä he mainitsivat mm vasteajat. Tässä työssä vasteaikoja järjestelmän osalta tarkasteltiin hyväksyntätestauksessa. Lisäksi Peffers ja kumppanit (2007) mainitsivat, että evaluoinnin perusteella pitää päättää, jatketaanko artefaktin kehittämistä vai siirrytäänkö seuraavaan vaiheeseen, missä kehitetyn järjestelmän toteutus ja oppima esitellään muille. Tässä työssä todettiin, että kehitetty järjestelmä toteutti vaatimusmääritellyssä ja ongelmakehyksessä esitetyn ongelman, joten toteutus ja oppima on dokumentoitu tähän työhön.

Nunamaker ja kumppanit (1990) eivät tutkimuksessaan määritelleet tarkasti, miten raportoida artefaktista. Heidän näkemyksensä on, että kehitettyä artefaktia tutkitaan ja evaluoidaan, jotta voidaan kehittää uusia artefakteja. Hevner ja kumppanit (2004) totesivat iteratiivisuuden olevan osana konstruktiivista tutkimusmenetelmää. Kasanen ja kumppanit (1993) ohjeistivat osoittamaan teoreettisen tutkimuksen ja ratkaisun yhteyden. Tässä työssä huomattiin iteratiivisuuden olevan todella tärkeä osa konstruktiivista tutkimusmenetelmää, koska ensimmäinen eikä toinenkaan toteutus välttämättä ratkaise ongelmaa kokonaisvaltaisesti. Huomataan, että ongelma, joka on jo saatu ratkaistua, kuten push-viestit ja niiden välitys päätelaitteelle, ei enää toimi tai toimii vain osittain. Joten kehitettyä artefaktia joudutaan evaluoimaan uudelleen ja uudelleen, koska ympärillä olevat järjestelmät muuttuvat ja niiden muutoksiin joudutaan reagoimaan. Toinen konkreettinen esimerkki iteratiivisuuden tärkeydestä tuli työn teon aikana ilmi, kun järjestelmän lähteenä oleva RSS-syöte muuttuu, muuttaa muotoaan ja lopulta katoaa. Tietokantapäivittäjän toteutusta on jouduttu muuttamaan ja korjaamaan useasti työnteon aikana, vaikka ohjelman perusrunko on pysynyt hyvin samana arkkitehtuurisesti. Lisäksi työn edetessä huomattiin, miten teknologian merkitys järjestelmän toteutuksessa vaikuttaa esimerkiksi ylläpidettävyyteen. Tästä esimerkkinä tässä työssä jouduttiin korvaamaan alkuperäinen tuotanto-palvelin uudemmalla Raspberry pi-koneella, entisen rikkouduttua. Tässä konstruktiossa toteutettiin teknologia siten, että entinen palvelin vaihdettiin vain uuteen ja järjestelmä toiminta jatkui laiterikon jälkeen ilman ylimääräistä säätämistä. Peffers ja kumppanit (2007) esittävät konstruktion viimeiseksi vaiheeksi

kommunikoinnin. Tässä työssä käytettiin konstruktion prosessikaaviota (kuva 5) ohjenuorana ja sen avulla käytiin läpi erilaiset konstruktiiviseen tutkimuksen osa-alueet. Lopputuloksena kehitetty järjestelmä raportoitiin yleisellä tutkimustyylillä, niin kuin Peffers ja kumppanit (2007) ehdottivat.

Seuraavassa esitetään tiivistetysti, miten esitettyihin tutkimuskysymyksiin vastattiin.

TK 1: *Sopiiko konstruktiivinen tutkimusmenetelmä Android-ohjelma suunnitteluun ja määrittelyyn sekä toteutukseen?*

Konstruktiivisen tutkimusmenetelmän avulla voidaan Android-ohjelma suunnitella, määritellä ja toteuttaa. Kuitenkin vastuu eri vaiheiden toteutuksesta ja valittavista metodeista on itse tutkijalla, mutta kokonaisuuden ratkaisussa menetelmä on hyvä kehys.

TK 2: *Miten saadaan olemassa oleva internetsivun tai RSS-syötteen sisältö tietyin reunaehdoin mobiilisovellukselle käytettäväksi ja kehitetty järjestelmä toimisi automaattisesti?*

Toiseen tutkimuskysymykseen vastaus on käytännössä koko tämä maisterityö, jonka avulla pystyy toteuttamaan vastaavan toteutuksen. Kokonaisuuden toteutus painottuu enimmäkseen palvelimen ja teknologian konstruktioon. Sen toteutus ja toiminnallisuus on reunaehto, mikä mahdollistaa itse mobiilisovelluksen toiminnallisuuden.

9. Yhteenveto

Tässä gradussa tutkittiin Android-ohjelman suunnittelua, määrittelyä ja toteutusta. Ensiksi käytiin läpi Android-arkkitehtuuri ja kuinka ohjelmien sisäinen kommunikointi sekä viestinvälitys toimivat. Seuraavaksi tarkasteltiin Android-arkkitehtuurin tietoturvamekanismit ja siihen liittyvää SELinuxia. Tämän jälkeen esiteltiin lukijalle Androidin tietoturvauhat ja siihen liittyvä tieteellinen tutkimus. Vaatimusmäärittelyn pääpiirteet ja tieteellinen tutkimus perinteisestä vesiputousmallista ketteriin menetelmiin tarkasteltiin. Konstruktiivisen tutkimuksen iteratiivisen luonteen takia järjestelmän arkkitehtuuri löytyy seuraavana. Konstruktiivisen tutkimuksen kirjallisuuteen tutustuttiin ja määriteltiin yhteiset piirteet eri tutkimusten välillä konstruktiivisen prosessikaavion avulla. Ongelmakehyksessä esiteltiin konstruktiivisen tutkimuksen ensimmäisen vaiheen eli ongelma alueen sisältö. Vaatimusmäärittelyn kirjallisuuden avulla esiteltiin sekä perinteisen vesiputous kehitysmallin, että ketterien menetelmien avulla konstruktiivista tutkimusta tukeva määrittelykeino. Tässä tapauksessa päädyttiin malliin, jossa käytettiin vaatimusmäärittelyn esittämiseen käyttäjäkertomuksia. Niiden avulla luotiin ja kirjoitettiin indeksikortit. Näiden pohjalta esiteltiin esimerkin Android- ja palvelinohjelman sekä teknologian vaatimusmäärittelyä. Konstruktion toteutus esiteltiin vaihtoehtoisina Webview-, JSON-, ja PWA-sovelluksina. Niiden toteutuksen mahdollistava palvelinohjelman konstruktio toteutettiin ja esiteltiin. Teknologian konstruktion avulla esiteltiin kokonaisuuden implementointi. Kehitetty järjestelmä evaluoitiin ensiksi analyyttisesti ja testausta käyttäen, niin yksikkö, integraatio, järjestelmä kuin hyväksyntätestauksen avulla. Lopuksi pohdittiin tutkimusmenetelmän sopivuutta ja todettiin, että konstruktiivisen tutkimusmenetelmä avulla voidaan suunnitella, määritellä ja toteuttaa Android-sovelluksen kehitys.

Tämän maisterityön rajoitteet olivat konstruktiivisen tutkimuksen avulla määritellä, suunnitella ja toteuttaa Android-sovellus. Työssä esiteltiin (kuva 5) konstruktion prosessikaava, jonka pohjalta ja avulla voidaan lähteä tutkimaan vastaavia ongelmakehykessä esiteltyjä järjestelmien toteuttamista mobiileille päätelaitteille.

Mielenkiintoisia ja mahdollisia jatkotutkimusaiheita ovat konstruktion asiakas-ohjelman implementointi myös toisille alustoille ja niiden evaluointi. Niiden perusteella tutkimustuloksen muodostaminen ja mahdollinen järjestelmän kehityksen yleistäminen samanlaiseen tai vastaavaan käyttöön. Toinen mahdollinen jatkotutkimusaihe on PWA-sovelluksen palvelimen toteuttamista, joko toisella Python pohjaisella web-frameworkilla tai toisella ohjelmointikielellä ja alustalla. Niiden evaluointia keskenään esimerkiksi vertaillen suorituskykyä.

Lähteet

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Atdd. (2020). Acceptance test-driven development. Luettu 9.12.2020 https://en.wikipedia.org/wiki/Acceptance_test%E2%80%93driven_development

Agile Manifesto. (2017). Principles behind the Agile Manifesto. Luettu 7.5.2017 http://agilemanifesto.org/principles.html

Android One. (2020) Android one secure, up-to-date and easy to use. Luettu 22.03.2020 https://www.android.com/one/

Andronio, N., Zanero, S., & Maggi, F. (2015). HelDroid: Dissecting and detecting mobile ransomware. *International Workshop on Recent Advances in Intrusion Detection,* 382-404.

API. (2019). Application programming interface. Luettu 18.2.2019 https://en.wikipedia.org/wiki/Application_programming_interface

Art and Dalvik. (2017). Android source. Luettu 24.4.2017 https://source.android.com/devices/tech/dalvik/

AsyncTask. (2019.) AsyncTask Android API level 28. Luettu 18.2.2019 https://developer.android.com/reference/android/os/AsyncTask

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T. M. (2017, April). Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. In *WEBIST* (pp. 344-351).

BSD-lisenssi. (2017). BSD-lisenssi on vapaa ohjelmistolisenssi. https://fi.wikipedia.org/wiki/BSD-lisenssi

Blueprints. (2019). Modular applications with Blueprints. Luettu 25.2.2019 http://flask.pocoo.org/docs/1.0/blueprints/

Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, (1), 64-69.

Bugiel, S., Davi, L., Dmitrienko, A., Fischer, T., & Sadeghi, A. (2011). Xmandroid: A new android evolution to mitigate privilege escalation attacks. *Technische Universitt Darmstadt, Technical Report TR-2011-04*

Burns, J. (2008). Developing secure mobile applications for android. *Developing Secure Mobile Applications for Android,*

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *IEEE Software, 25*(1)

Certbot. (2019). Automatically enable HTTPS on your website, with EFF's Certbot. Luettu 28.2.2019 https://certbot.eff.org/

Chin, E., Felt, A. P., Greenwood, K., & Wagner, D. (2011). Analyzing inter-application communication in android. *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services,* 239-252.

Davi, L., Dmitrienko, A., Sadeghi, A., & Winandy, M. (2010). Privilege escalation attacks on android. *International Conference on Information Security,* 346-360.

Duck DNS. (2019) Duck DNS free dynamic DNS hosted on AWS. Luettu 28.2.2019 http://www.duckdns.org/

Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B., Cox, L. P., . . . Sheth, A. N. (2014). TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS), 32*(2), 5.

Enck, W., Ongtang, M., & McDaniel, P. (2009). Understanding android security *IEEE Security & Privacy Magazine, 7*(1), 50-57. doi:10.1109/MSP.2009.26

Fahl, S., Harbach, M., Muders, T., Baumgrtner, L., Freisleben, B., & Smith, M. (2012). Why eve and mallory love android: An analysis of android SSL (in) security. *Proceedings of the 2012 ACM Conference on Computer and Communications Security,* 50-61.

Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). Android permissions demystified. *Proceedings of the 18th ACM Conference on Computer and Communications Security,* 627-638.

Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). A survey of mobile malware in the wild. *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices,* 3-14.

Flask. (2017). Flask microframework. Luettu 5.9.2017 http://flask.pocoo.org/

Gartner Worldwide Sales. (2017). Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016. Luettu 26.4.2017 http://www.gartner.com/newsroom/id/3609817

Gaunt, M. (2016). Service Workers: an introduction. Luettu 19.02.2019 https://developers.google.com/web/fundamentals/primers/service-workers/

Gaunt, M. & Kinlan, P. (2019). The Web App Manifest. Luettu 20.2.2019 https://developers.google.com/web/fundamentals/web-app-manifest/

Grace, M. C., Zhou, W., Jiang, X., & Sadeghi, A. (2012). Unsafe exposure analysis of mobile in-app advertisements. *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks,* 101-112.

Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012). Riskranker: Scalable and accurate zero-day android malware detection. *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services,* 281-294.

Grinberg, M. (2018). Flask web development: developing web applications with python. " O'Reilly Media, Inc.".

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75-105.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120-127.

Intent. (2017). Android Developer. Luettu 7.4.2017 https://developer.android.com/reference/android/content/Intent.html

JSON. (2019). JavaSript Object Notation. Luettu 18.2.2019 https://json.org/

Kasanen, E., Lukka, K., & Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research, 5*, 243.

Laplante, P. A. (2013). *Requirements engineering for software and systems* CRC Press.

Leffingwell, D. (2010). *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional.

LePage P. (2017). Your First Progressive Web App. Luettu 19.2.2019 https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/

Let's Encrypt. (2019). Let's Encrypt is free, automated, and open Certificate Authority. Luettu 28.02.2019 https://letsencrypt.org/

Luo, T., Hao, H., Du, W., Wang, Y., & Yin, H. (2011, December). Attacks on WebView in the Android system. In *Proceedings of the 27th Annual Computer Security Applications Conference* (pp. 343-352).

Nunamaker Jr, J. F., Chen, M., & Purdin, T. D. (1990). Systems development in information systems research. *Journal of Management Information Systems, 7*(3), 89-106.

Octeau, D., McDaniel, P., Jha, S., Bartel, A., Bodden, E., Klein, J., & Le Traon, Y. (2013). Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. *Proceedings of the 22nd USENIX Security Symposium,* 543-558

OneSignal. (2019). OneSignal is a free push notification service for web and mobile apps. Luettu 18.3.2019 https://github.com/OneSignal/OneSignal-Website-SDK

Paetsch, F., Eberlein, A., & Maurer, F. (2003, June). Requirements engineering and agile software development. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* (pp. 308-313). IEEE.

PHA Classification. (2017). Android Security Team: Google Android Security PHA Classification. Luettu 9.4.2017 https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_PHA_classifications.pdf

Platform architecture. (2017). Android Developer. Luettu 6.4.2017 https://developer.android.com/guide/platform/index.html

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45-77.

Permission groups. (2017). Android Developers. Luettu 1.4.2017 https://developer.android.com/guide/topics/permissions/requesting.html

Python. (2020). Python is programming language. Luettu 2.11.2020 https://www.python.org/

Robot Framework. (2019). Generic test automation framework. Luettu 26.2.2019 https://robotframework.org/

Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON, , 26*(8) 1-9.

Sarwar, G., Mehani, O., Boreli, R., & Kaafar, D. (2013). On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices. *National ICT Australia,*

SELinux.(2017). Android Developers. Luettu 18.4.2017 https://source.android.com/security/selinux/

Schlegel, R., Zhang, K., Zhou, X., Intwala, M., Kapadia, A., & Wang, X. (2011). Soundcomber: A stealthy and context-aware sound trojan for smartphones. *Proceedings of the 18th Annual Network and Distributed System Security Symposium* (Vol. 11, pp. 17-33).

Shabtai, A., Fledel, Y., & Elovici, Y. (2010). Securing android-powered mobile devices using SELinux. *IEEE Security & Privacy, 8*(3), 36-44.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). "Andromaly": A behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems, 38*(1), 161-190.

Smalley, S., & Craig, R. (2013). Security enhanced (SE) android: Bringing flexible MAC to android. In *Network and Distributed System Security Symposium, 310* 20-38.

Sqlalchemy. (2019). The Python SQL Toolkit and Object Relational Mapper. https://www.sqlalchemy.org/

Thomas, D. R., Beresford, A. R., & Rice, A. (2015). Security metrics for the android ecosystem. *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices,* 87-98.

Törrö, T. (2017) Android ohjelman suunnittelu ja määrittely käyttäen konstruktiivista suunnittelua. Kandidaatti -tutkielma. Oulun yliopisto, Oulu.

User story. (2017). User stories: an agile introduction. Luettu 09.05.2017 http://www.agilemodeling.com/artifacts/userStory.htm

uWSGI. (2019). The uWSGI project. Luettu 27.06.2019 https://uwsgi-docs.readthedocs.io/en/latest/

Virtualenv. (2019). Virtualenv is a tool to create isolated Python environments. Luettu 21.9.2019 https://virtualenv.pypa.io/en/latest/

Web_framework. (2020). Software framework that is designed to support web applications. Luettu 2.11.2020 https://en.wikipedia.org/wiki/Web_framework

WebView. (2017). Public class WebView. Luettu 27.05.2017 https://developer.android.com/reference/android/webkit/WebView.html

Wei, F., Roy, S., & Ou, X. (2014, November). Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1329-1341). ACM.

Xu, Z., & Zhu, S. (2012). Abusing notification services on smartphones for phishing and spamming. *In 6th USENIX Workshop of Offensive Technologies (WOOT12),* 1-11.

Zhang, X., Hu, W., & Jin, S. (2017). Google Play Apps Infected with Malicious IFrames. Lainattu 18.4.2017, http://researchcenter.paloaltonetworks.com/2017/03/unit42-google-play-apps-infected-malicious-iframes/

Zhou, Y., & Jiang, X. (2012). Dissecting android malware: Characterization and evolution. *Security and Privacy (SP), 2012 IEEE Symposium On,* 95-109.

Zhou, W., Zhou, Y., Jiang, X., & Ning, P. (2012). Detecting repackaged smartphone applications in third-party android marketplaces. *Proceedings of the Second ACM Conference on Data and Application Security and Privacy,* 317-326.

Liite A. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.user.application ">
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"

        android:icon="@mipmap/ic_launcher_ruokalista"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.NoActionBar">

        <activity android:name=".MainActivity"
            android:configChanges="orientation|screenSize"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Liite B. MainActivity.java

Tässä on MainActivity luokan kokonaisuus.

```
package com.example.user.application;


import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;


public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        WebView myWebView = (WebView) findViewById(R.id.webview);
        myWebView.loadUrl("ip or domain");
        myWebView.setWebViewClient(new MyWebViewClient());
        WebSettings webSettings = myWebView.getSettings();
        webSettings.setJavaScriptEnabled(true);
    }


    private class MyWebViewClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String
url) {
```

```
            if (Uri.parse(url).getHost().equals("ip or domain"))
            {
                return false;
            }
            else if (url.startsWith("ip or domain")){
                return false;
            }
            else
            {
                Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                startActivity(intent);
                return true;
            }

        }
    }
}
```

Liite C. activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools"

    android:id="@+id/activity_main"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:paddingBottom="@dimen/activity_vertical_margin"

    android:paddingLeft="@dimen/activity_horizontal_margin"

    android:paddingRight="@dimen/activity_horizontal_margin"

    android:paddingTop="@dimen/activity_vertical_margin"

    tools:context="com.example.user.application.MainActivity">

    <WebView

        android:id="@+id/webview"

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:layout_alignParentTop="true"

        android:layout_alignParentLeft="true"

        android:layout_alignParentStart="true" />

</RelativeLayout>
```

Liite D. URL-lista

[1] https://www.raspberrypi.org/
[2] http://www.appanalysis.org/
[3] https://flask.palletsprojects.com/en/1.1.x/
[4] https://werkzeug.palletsprojects.com/en/1.0.x/
[5] https://jinja.palletsprojects.com/en/2.11.x/
[6] https://httpd.apache.org/
[7] https://www.nginx.com/
[8] https://www.iis.net/
[9] https://letsencrypt.org/
[10] https://certbot.eff.org/
[11] https://wsgi.readthedocs.io/en/latest/what.html
[12] https://uwsgi-docs.readthedocs.io/en/latest/
[13] https://gunicorn.org/
[14] https://www.sqlite.org/index.html
[15] https://www.sqlalchemy.org/
[16] https://redis.io/
[17] https://onesignal.com/
[18] https://www.pfsense.org/
[19] https://www.duckdns.org/
[20] https://docker.com
[21] https://netlify.com
[22] https://www.ssllabs.com/ssltest/
[23] https://postman.com
[24] https://github.com/GoogleChrome/lighthouse
[25] https://www.sitespeed.io/
[26] https://locust.io/