jamk

Viivakoodin lukeminen mobiililaitteilla

Skannaukseen perustuvien ohjelmistokirjastojen käyttö ohjelmistokehityksessä

Sara Ainali

Opinnäytetyö, AMK
Marraskuu 2022
Tietojenkäsittelyn tutkinto-ohjelma
Tradenomi (AMK)

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Ainali, Sara

Viivakoodin lukeminen mobiililaitteilla. Skannaukseen perustuvien ohjelmistokirjastojen käyttö ohjelmistokehityksessä

Jyväskylä: Jyväskylän ammattikorkeakoulu. Marraskuu 2022, 35 sivua

Tradenomi. Tietojenkäsittelyn tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Opinnäytetyön tavoitteena oli selvittää, että voidaanko löytää parempaa skanneriohjelmistokirjastoa valmiille Angular-ohjelmistokehyksellä rakennetulle projektille, joka käyttää avoimen lähdekoodin skanneriohjelmistokirjastoa viivakoodien lukemiseen.

Työ toteutettiin neljässä vaiheessa. Ensimmäisenä valittiin kolme skanneriohjelmistokirjastoa, jotka tukivat työssä laadittua kriteeristöä. Seuraavaksi näistä kolmesta aiemmin valitusta kirjastosta päädyttiin vertailun ja dokumentaatioiden avulla vain yhteen kirjastoon, jota kolmannessa vaiheessa lähdettiin käyttöönottamaan valmiin projektin pohjassa. Käyttöönoton jälkeen uutta kirjastoa testattiin ja verrattiin projektin vanhaan kirjastoon ja selvitettiin, että saadaanko uudella kirjastolla parempia tuloksia kuin vanhalla kirjastolla.

Tuloksissa päädyttiin siihen, että uusi kirjasto tuottaa vanhaan kirjastoon verrattuna parempia tuloksia testatuissa ympäristöissä. Testaus suoritettiin eritavoin valaistuissa ympäristöissä ja eri pakkausmateriaaleilla pohjustetuilla viivakoodeilla. Vaikka todettiin, että uusi kirjasto toimii rajatuissa ympäristöissä paremmin kuin vanha, tarvitaan silti enemmän testaamista ja koodin muokkaamista projektikohtaisemmaksi, jotta tiedetään tarkemmalla varmuudella, että uusi kirjasto ei vain tuota parempia viivakoodien tuloksia, vaan se toimii kokonaisuudessaan paremmin projektissa.

Avainsanat (asiasanat)

Ohjelmistokirjasto, Angular, Viivakoodi, Skanneri

jamk

Description

Ainali, Sara

Decoding barcodes with mobile devices. Use of scanning-based software libraries in software development

Jyväskylä: JAMK University of Applied Sciences, November 2022, 35 pages

Bachelor of Business Administration. Bachelor's Degree Programme in Business Information Technology. Bachelor’s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The aim of the thesis was to find out whether there is a better alternative to a software library for a finished project built with the Angular software framework, which uses an open-source scanner software library for reading barcodes.

The work was carried out in four stages. First, using a set of criteria, three libraries were selected that supported the criteria developed in the thesis. Next step was to choose only one library from these three libraries by means of comparison and examination. All the libraries' own documentations were used for the examination. In the third stage the one chosen library was deployed to the project. After deployment, the new library was tested and compared to the old library in the project, and it was determined whether the new library produces better results than the old library.

The results concluded that the new library produces better results in the tested environments compared to the old library. The testing was performed in differently lit environments and with barcodes primed with different packaging materials. Although the new library works better in limited environments than the old one, more testing and code manipulation is needed to be more project-specific, to know with better certainty that the new library not only produces better barcode results, but also works better within the project.

Keywords/tags (subjects)

Software library, Angular, Barcode, Scanner

Sisältö

Kuviot

1 Johdanto

Viivakoodien lukeminen onnistuu nykyään helposti mobiililaitteiden kameran avulla. Ohjelmointikehityksessä sovelluksiin voidaan implementoida valmiita skanneriohjelmistokirjastoja, jotka käyttävät mobiililaitteiden kameraa viivakoodien tulkitsemiseen ja lukemiseen. Eli ohjelmistokehittäjän ei tarvitse lähteä kehittämään skanneria itse, vaan hän voi hyödyntää valmiita avoimen lähdekoodin skanneriohjelmistokirjastoja, joita muut ohjelmistokehittäjät ylläpitävät.

Avoimen lähdekoodin skanneriohjelmistokirjastoja löytyy esimerkiksi GitHubista. Näiden kirjastojen käytössä, käytänteissä ja toiminnallisuuksissa voidaan havaita eroavaisuuksia ja eri kirjastojen kanssa voi ilmaantua erinäisiä ongelmia, jotka vaikuttavat skannauksien tuloksiin negatiivisesti. Ongelmia voi syntyä esimerkiksi eri mobiililaitteiden yhteensopivuudessa kirjaston kanssa; skanneri voi olla hidas tai lukea usein väärin. Kirjaston dokumentointi on heikkoa ja kirjastoa on siksi hankala käyttää. Tutkimusta tarvitaan sen takia, jotta löydettäisiin parempi skanneriohjelmistokirjasto Angular-ohjelmistokehyksellä rakennettuihin sovelluksiin ja sovelluksiin, jotka toimivat selainalustoilla.

Opinnäytetyössä keskitytään skannerin tuloksien luotettavuuteen ja lukemisen nopeuteen eri valoisissa ympäristöissä. Tutkimusta lähdetään selvittämään etsimällä kevyen kriteeristön avulla vähintään kolme eri skanneriohjelmistokirjastoa, jotka tukevat rajattuja teknologioita. Rajatut teknologiat ovat Angular ja Progressive Web App eli PWA. Kriteeristön avulla valittuja kirjastoja verrataan keskenään ja tämän vertailun avulla valitaan vain yksi kirjasto, jota lähdetään empiirisesti käyttöönottamaan Allergiaskanneri nimisen projektin pohjassa. Käyttöönoton jälkeen uutta kirjastoa testataan ja verrataan Allergiaskannerin nykyiseen kirjastoon. Vertailulla selvitetään tuottaako uusi kirjasto parempia tuloksia kuin nykyinen kirjasto.

Tutkimuksen selvittäminen on tarpeellista, koska Allergiaskanneri-projektissa nykyisen skanneriohjelmistokirjaston kanssa on ollut ongelmia ja tulokset eivät aina ole olleet toivottuja. Tämän opinnäytetyön avulla yritetään selvittää löytyisikö nykyiselle kirjastolle parempi vaihtoehto. Allergiaskanneri on projekti, jota on kehitetty Jyväskylän ammattikorkeakoulun Ticorporate opintojaksoilla. Se on sovellus, jonka tarkoituksena on helpottaa ja nopeuttaa allergikoiden kaupassakäyntiä, niin ettei käyttäjän tarvitse itse lukea elintarvikkeiden ainesosalistaa, vaan sovellus

tekee tämän heidän puolestaan ja kertoo, mikäli skannattu tuote sisältää hänelle epäsopivia ainesosia. Tämä opinnäytetyö on osa tietojenkäsittely tutkinto-ohjelmaa. Opinnäytetyön aihe on kehitetty itsenäisenä työnä, eikä sillä ole toimeksiantajaa.

2 Viivakoodit ja niiden käyttötarkoitukset

Tässä luvussa kerrotaan viivakoodeista ja niiden käyttötarkoituksesta sekä historiasta. Luvussa perehdytään myös erilaisiin viivakoodityyppeihin.

2.1 Viivakoodeista yleisesti

Viivakoodeja hyödynnetään monella toimialalla tuotannon helpottamisen ja nopeuttamisen takaamiseksi. Se toimii yksilöllisenä tunnistimena muun muassa elintarviketuotteille. Viivakoodin tunnistaa yleisesti sen eri levyisistä ja paksuisista valkoisista ja mustista viivoista tai pisteistä. Nämä symbolit kuvastavat koneellisesti tarkistettavissa olevia numero ja kirjain yhdistelmiä, joita voidaan lukea, tallentaa sekä käsitellä erinäisten tietojärjestelmien ja lukijoiden avulla. (GS1-standardit viivakoodeille ja RFID-tunnisteille n.d.)

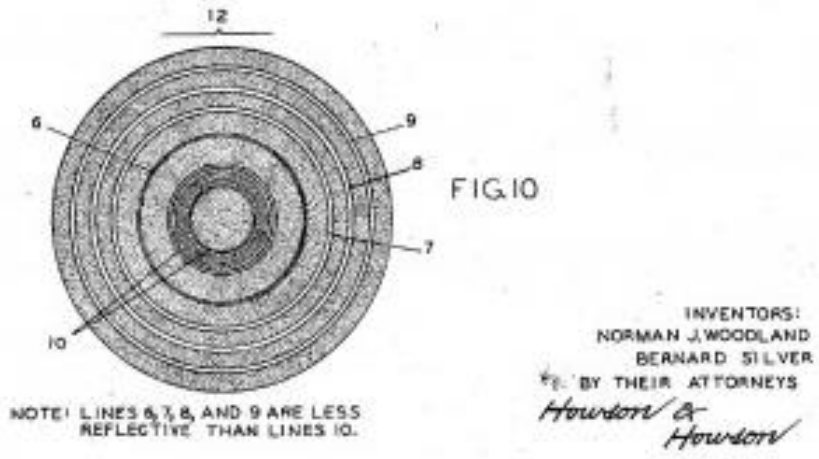
Viivakoodit käyttävät standardeja eli yhdenmukaisia sääntöjä ja tapoja tuotteiden yksilöintiin. GS1 on maailmanlaajuinen organisaatio, joka kehittää ja hallinnoi käytetyimpiä tuotetietojen sekä yksilöinnin standardeja, joista tunnetuin on EAN-viivakoodi. GS1-standardit tukevat yhteisiä tapoja tuotteiden, toimitusketjun tapahtumien ja paikkojen yksilöintiin, tunnistamiseen sekä tiedon jakamiseen ja helpottavat tiedon jakamista eri toimijoiden välillä. (Paremman huomisen puolesta – supervoimana standardit n.d.)

2.2 Viivakoodien historia

Vuonna 1940 Bernard Silver, silloinen opiskelija Philadelphian Drexel yliopistossa, kuuli sattumalta päivittäistavarateollisuuden tarpeesta automatisoidulle tuotteiden tunnistukselle lisääntyneen kysynnän takia. Silver kertoi kuulemastaan ystävälleen Norman Joseph Woodlandille, jotka yhdessä päättivät aloittaa kehittämään ilmenneelle ongelmalle ratkaisua. (Drobnik 2015, luku A.1.)

Hiekkaan piirrellessään Woodland huomasi, kuinka eri paksuiset viivat voisivat edustaa eri numeroita morsekoodin tyyppisesti. Tästä Woodland ja Silver saivat lähtökohtaisen idean viivakoodille ja jatkoivat sen kehitystä seuraavat yhdeksän vuotta. Useiden hakemuksien jälkeen vuonna 1952 keksinnölle myönnettiin viimein US-patentti. (Drobnik 2015, luku A.1.)

Patentoitu lähestymistapa oli tehdä viivakoodeista ympyrän muotoisia, jotta koodia pystyttäisiin lukemaan miltä suunnalta tahansa. Sille annettiin englanninkielinen epävirallinen nimi bull's-eye symbol (ks. kuvio 1). George J. Laurer kuitenkin huomasi, että kyseinen viivakoodi suttaantui helposti painatuksessa ja teki siitä näin lukukelvottoman. Ratkaisuksi Laurer suunnitteli lineaarisen viivakoodin. (Drobnik 2015, luku A.2.)



Kuvio 1. Bull's-eye symbolin patentti

Vuonna 1973 lehdistötiedotteessa julkaistiin Universal Product Code lyhennettynä UPC. 26. kesäkuuta 1974 Joe Woodland skannasi ensimmäisen UPC-koodilla merkatun elintarvikepakkauksen. (Drobnik 2015, luku A.2.)

2.3 Lineaariset viivakoodit

Lineaariset tai 1D-viivakoodit ovat viivakoodityypeistä vanhimpia. Lineaarisia viivakoodejatyyppejä on kahdenlaisia ja ne ovat englanninkielisiltä nimiltään numeric-only ja alpha-numeric. Numeric-only koostuu nimensä mukaan pelkästään numeroista ja alpha-numeric koostuu sekä numeroista että kirjaimista (Stazzone 2022). Lineaarinen viivakoodi rakentuu eri paksuisista valkoisista ja mustista pystyviivoista ja sen pituus kuvastaa kuinka paljon informaatiota kyseinen viivakoodi sisältää. Tunnetuin lineaarisen viivakoodin tyyppi on EAN. Muita mainittavia lineaarisen viivakoodin tyyppejä ovat esimerkiksi Code-39, Code-128 ja Codabar. (Types of Barcodes: Choosing the Right Barcode 2015.)

European Article Number eli EAN (ks. kuvio 2) on maailmanlaajuisesti standardoitu viivakoodi, mutta sitä käytetään pääasiassa vain Euroopassa. EAN-koodeja on sisällöltään eri pituisia, jotka ovat nimeltään EAN-13 ja EAN-8. EAN-13 koostuu nimensä mukaan 13 numeron sarjasta ja EAN-8 kahdeksan numeron sarjasta. Näistä kahdesta EAN-13 on yleisemmin käytetty pakkausmerkintä ja EAN-8 hyödynnetään pääasiassa vain pienempi kokoisten tuotteiden merkkaamiseen. Yhdysvalloissa EAN-koodia vastaa UPC eli Universal Product Code. (Types of Barcodes: Choosing the Right Barcode 2015.)

Kuvio 2. EAN-13 viivakoodityyppi

2.4 Kaksiulotteiset viivakoodit

Kaksiulotteinen viivakoodi eli 2D-viivakoodi koostuu nimensä mukaan kaksiulotteisista symboleista ja muodoista. 2D-viivakoodit sisältävät lineaarisiin viivakoodeihin verrattuna enemmän tietoa. Niiden suurin hyöty on myös siinä, että viivakoodin revetessä, se on edelleen lukukelvollinen. 2D-viivakoodityyppejä ovat muun muassa Data Matrix, PDF417, Maxicode ja QR-koodi (ks. kuvio 3). (Scandit 2015.)

Kuvio 3. Esimerkki QR-koodista, joka ohjaa Wikipedian etusivulle

Nykypäivänä elintarviketuotteet käyttävät lineaarisia viivakoodeja tuotteiden tunnistukseen, mutta tähän on suunnitelmissa muutos. Viimeistään vuoteen 2027 mennessä tarkoituksena on siirtyä 2D-viivakoodien käyttöön tuotteiden pakkausmerkinnöissä maailmanlaajuisesti. Tämä ei kuitenkaan tarkoita sitä, että EAN-koodit poistetaan tuotteiden pakkauksista heti muutoksen astuessa voimaan, vaan molemmat viivakoodityypit ovat käytössä samanaikaisesti sujuvan siirtymän takaamiseksi. (2D-viivakoodit käyttöön viimeistään 2027 2022.)

3 Web-sovelluskehitys

Tässä luvussa kerrotaan web-kehityksen periaatteista ja tutustutaan syvemmin Angular-ohjelmistokehykseen ja sen periaatteisiin.

3.1 Web-kehityksestä yleisesti

Web-kehityksen kysyntä on kasvanut viime vuosien aikana huomattavasti. Web-kehityksellä viitataan verkkosivujen ja -sovelluksien luomiseen ja ylläpitoon selaimella. Front-end ja back-end ovat web-kehityksen tyyppejä, mistä front-end kehityksessä keskitytään verkkosivun asiakaspuoleen eli siihen mitä käyttäjä sivustolla näkee. Back-end-kehityksessä taas keskitytään siihen, miten ohjelmisto toimii sivuston tai sovelluksen takana. (What is web development n.d.)

Web-kehitys keskittyy verkkosivustoihin ja sovelluksiin, joita ajetaan Internetissä. Internet tarvitsee protokollia laitteiden väliseen kommunikointiin. Protokollat tukevat yhteisiä käytänteitä ja sääntöjä, jotka ohjaavat tiedonsiirtoa eri laitteiden välillä. Protokollien avulla esimerkiksi varmistetaan, että data lähetään oikeaan osoitteeseen ja datan asettelu on vaadittavassa muodossa. (Blank 2004, luku 2.) TCP/IP on sarja kommunikaatio protokollia, joiden tarkoituksena on yhdistää verkkolaitteita Internetissä. TCP/IP selvittää, millä tavalla dataa siirretään internetissä kahden laitteen välillä. TCP/IP pohjautuu sanoista Transmission Control Protocol/Internet Protocol, jotka voivat toimia myös erillisinä protokollina. (Shacklett ym. 2021.) HTTP eli Hypertext Transfer Protocol on standardi protokolla, mikä ajaa selaimen ja palvelimen välistä kommunikointia. HTTP seuraa asiakaspalvelinmallia, jossa asiakas tekee pyynnön palvelimelle ja palvelin lähettää vastauksen asiakkaalle. (HTTP 2021.) DNS eli Domain Name System puolestaan muuttaa IP-osoitteet luettavampaan ja helposti muistettavampaan muotoon (What is DNS n.d).

Käytetyimmät ohjelmointikielet web-kehityksessä ovat HTML, CSS ja JavaScript. Tämä ei kuitenkaan tarkoita, etteikö kehityksessä pystyttäisi käyttämään muita ohjelmointikieliä. (What is web development n.d.) Kaiken pohjana toimii HTML-dokumentti, joka määrittelee sivun rakenteen ja sisällön sekä kertoo mitä tiedostoja sen täytyy ladata verkon yli palvelimelta. CSS on tyylitiedosto, joka määrittelee miltä dokumentti ja sen elementit näyttävät. JavaScript-tiedostot ovat skript-tiedostoja, jotka toteuttavat interaktiot ja muut selainpäässä ajettavat toiminnallisuudet. (Tapala 2016.)

3.2 Angular-ohjelmistokehys

Ohjelmistokehys on web-kehityksen työväline, mikä tarjoaa valmiita komponentteja ja ratkaisuja nopeamman kehityksen takaamiseksi. Kehykset ottavat ohjelmistokirjastojen muodon ja sen tarkoituksena on yksinkertaistaa kehityksen kulkua. (Software Framework 2018.)

Opinnäytetyön toteutuksessa käytetään Angular-ohjelmistokehystä, koska Allergiaskanneri-projekti on rakennettu sitä käyttäen. Siksi tässä luvussa syvennytään pelkästään Angular-ohjelmistokehyksen perusteisiin. Angular on ohjelmistokehitysalusta, joka on rakennettu TypeScriptille. Se on Googlen kehittämä ja ylläpitämä ohjelmistokehys, joka helpottaa verkkosovellusten kehittämistä. Misko Hevery ja Adam Abrons aloittivat Angularin kehittämisen sivuprojektina vuonna 2009. Sen tarkoituksena oli rakentaa sovelluksia HTML-tunnisteita eli tageja hyödyntäen. Nimensä Angular saikin HTML-tunnisteita ympäröivien kulmasulkujen myötä. (Kumar 2019, luku 1.)

Angularin arkkitehtuuri koostuu muun muassa komponenteista, moduuleista ja palvelimista. Komponentit (engl. component) Angular-projekteissa rakentavat sovelluksen tai sivuston ulkoasun ja sisältävät ohjelmiston logiikkaa. Jokainen komponentti koostuu muun muassa TypeScript-luokasta, joka määrittelee logiikan, HTML-dokumentista, joka määrittelee sen, että mitä sivulla tulee näkymään ja vaihtoehtoisesta tyylitiedostosta (ks. kuvio 4). Komponentilla voi olla lapsikomponentteja eli komponentteja, joita ajetaan toisen komponentin sisällä. Angular sisältää myös Data Binding-määritelmän, joka on vapaasti suomennettuna tietojen sidontaa. Tietojen sidonta kertoo, kuinka tieto liikkuu komponentin TypeScript-luokan ja HTML-dokumentin välillä. Tiedon sidonta tyyppejä on kolme ja ne ovat Interpolation, Property ja Event Binding. (Kumar 2019, luku 2.)

```
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-example-component',
5    templateUrl: './example-component.component.html',
6    styleUrls: ['./example-component.component.css']
7  })
8  export class ExampleComponentComponent implements OnInit {
9
```

Kuvio 4. Angular komponentin TypeScript-luokka

Komponentit eivät kommunikoi suoraan keskenään, joten niiden välissä käytetään usein palvelinta (engl. service). Palvelin on luokka, joka ajaa vain yhtä tiettyä toimintaa Angular-projektissa (ks. kuvio 5). Palvelimen luokan toiminta voi olla mitä tahansa. Palvelimessa käytetään @Injectable-määritelmää, joka merkkaa luokan käytettäväksi riippuvuutena (engl. dependency). Riippuvuudet ovat palvelimia tai objekteja, joita luokka tarvitsee funktioiden suorittamiseen (Dependency injection in Angular 2022). Komponenttien välisen kommunikoinnin lisäksi palvelinta voidaan käyttää esimerkiksi kirjautumisen tai tunnistautumisen skenaarioissa. (Kumar 2019, luku 11.)

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class ExampleServiceService {
7
```

Kuvio 5. Angular palvelin-tiedoston koodipohja

Angular moduuli (engl. module) on käyttöönoton alajoukko. Jokaiseen luotuun Angular projektiin kuuluu automaattisesti juurimoduuli, mutta moduuleja voidaan luoda tarvittaessa lisää. Useiden moduulien avulla projektia voidaan jakaa pienempiin osiin ryhmittämällä esimerkiksi yhteensopivat komponentit ja direktiivit. Moduulien hyötyä nousee esille esimerkiksi HTML-dokumenttien

jäsentämisen aikana. Normaali tapauksessa jokaista HTML-tunnistetta verrataan kaikkiin projektissa oleviin komponentteihin tai direktiiveihin. Moduulin avulla pystytään kertomaan projektille suoraan, missä yhteensopivat komponentit sijaitsevat. (Angular Modules and NgModule - Complete Guide 2022.) Angular moduuleilla on oma modulaarisuus järjestelmä NgModules. NgModules on luokka, joka määrittää injektorin (engl. injector) ja kääntäjän (engl. compiler) sekä auttaa järjestämään toisiinsa liittyvät komponentit ja dokumentit yhteen. (ks. kuvio 6) (NgModules 2022.)

```
1   import { NgModule } from '@angular/core';
2   import { BrowserModule } from '@angular/platform-browser';
3   import { AppComponent } from './app.component';
4
5   @NgModule({
6     declarations: [AppComponent],
7     imports: [BrowserModule],
8     providers: [],
9     bootstrap: [AppComponent]
10  })
11  export class AppModule { }
```

Kuvio 6. Angular moduuli-tiedoston koodipohja

Vaikka tässä luvussa keskitytään pelkästään Angular-ohjelmistokehykseen, on se kuitenkin vain yksi ohjelmistokehitysalustoista. Muita mainittavia ohjelmistokehyksiä, joita voidaan hyödyntää web-kehityksessä ovat muun muassa React ja Vue.

3.3 Progressive Web App (PWA)

Progressive Web App (lyhyemmin PWA) on Googlen lanseeraama määritelmä sovelluksille, jotka on toteutettu web-kehityksen tekniikoilla ja sovelluksille, jotka toimivat suoraan selaimella ilman, että niitä on tarvetta ladata sovelluskaupoista kuten Googlen Play kaupasta. PWA-sovelluksia on kuitenkin mahdollista viedä sovelluskauppoihin. (Mikä on progressiivinen verkkosovellus (PWA) ja mitä etuja se tarjoaa? 2020.) Jeremy Keith (2017) on esittänyt, että Progressive Web Appien teknninen määritelmä on verkkosivusto, jota tarjotaan HTTPS:n yli palvelutyöntekijän (engl. service worker) ja manifest-tiedoston kanssa.

Berriman ja Russell (2015) ovat dokumentoineet yhdeksän tarkentavaa ominaisuutta PWA sovelluksille, jotka ovat seuraavanlaiset:

1. Reagoiva eli sovellus mukautuu näytön koon mukaisesti.
2. Yhteydestä riippumaton eli sovellus toimii myös offline-tilassa tai huonolla verkkoyhteydellä.
3. Sovelluksen kaltainen ohjelma.
4. Tuore eli sovellus päivittyy taustalla aina uusimpaan versioon.
5. Turvallinen.
6. Löydettävissä eli verkkosivustot ilmoittavat olevansa sovelluksia yksinkertaisen manifest-tiedoston avulla, jota hakukoneet ja selaimet käyttävät uusien sovelluksien etsimiseen.
7. Interaktiivinen eli sovellus pystyy lähettämään käyttäjälle ilmoituksia ja muistutuksia.
8. Ladattavissa eli sovelluksen pystyy ladata laitteen aloitusnäytölle niin halutessaan.
9. Linkitettävissä eli sovellusta pystytään jakamaan helposti linkin avulla ja sitä voidaan käyttää ilman sovelluksen lataamista.

Yleensä sovellukset toteutetaan tietyillä teknologioilla ja nämä tietyt teknologiat sopivat vain tietyille sovellusalustoille. PWA-sovelluskehityksen myötä selaimella toimivat sovellukset voivat toimia eri alustoilla, vaikka sovellus olisi rakennettu vain yhdellä ja samalla teknologialla. Näin voidaan vähentää koodauksen ja resurssien määrä. (Mikä on progressiivinen verkkosovellus (PWA) ja mitä etuja se tarjoaa? 2020.)

Grigsbyn (2018) mukaan useat yritykset ovat kiinnostuneita mahdollistamaan sovelluksen kaltaisien käytänteiden hyödyntämisestä web-kehityksessä ja, että nämä yritykset haluavat hyödyntää verkon omia käytänteitä sovelluksien luomiseen, vaikka sovellukset sijaitsisivat selaimen ulkopuolella. Ongelmaksi kuitenkin ilmeni se, että vaikka sovellukset rakennettiin käyttäen verkkoa ne eivät olleet osa verkostoa. (Grigsby 2018, luku 1.)

3.4 Avoin lähdekoodi osana kehittämistä

Avoin lähdekoodi (eng. open source code) on tapa jakaa, kehittää ja muokata koodia, joka on kaikille avoimesti saatavilla. Kaikki voivat vapaasti osallistua ohjelmiston kehitystyöhön ja näin edesauttaa ohjelmistojen hyvää tietoturvaa ja korkeaa laatua. Lähdekoodi on se osa ohjelmistoa, jota

normaali ohjelmiston käyttäjä ei näe ja minkä avulla toteutetaan ohjelmiston toiminta. Ohjelmistokehittäjät, joilla on pääsy ohjelmiston lähdekoodin, pystyvät muokkaamaan ja korjaaman ohjelmiston toimintaa. (What is open source n.d.)

Avoin lähdekoodi käsite korvasi alkuperäisen free software termin (vapaa ohjelma), koske se koettiin harhaanjohtavana. Christine Petersonin (2018) mukaan ”Aiemman termin ’free software’ ongelma ei ollut poliittinen konnotaatio, vaan se, että – uusille ohjelmistokehittäjille - sen näennäinen keskittyminen hintaan oli häiritsevä”.

3.5 Ohjelmistokirjastot

Ohjelmistokirjasto on ohjelmistokehityksessä käytettävä kirjasto, joka koostuu valmiista tiedostoista ja avoimesta lähdekoodista, jota kuka vain ohjelmistokehittäjä voi halutessaan muokata tai hyödyntää omissa projekteissaan hänen haluamallaan tavalla. (What is open source n.d.)

Ohjelmistokirjastot käyttävät lisenssejä, jotka tukevat ohjelmistokirjaston kehittäjän ja käyttäjän sujuvaa yhteistyötä. Ohjelmistokirjastojen käyttö on yleensä ilmaista, mutta sisältää rajoituksia, joita ohjelmistokirjaston käyttäjän tulee aina noudattaa. Siksi lisenssit täytyy ottaa huomioon, jotta ohjelmistoa käytetään vain tarpeisiin joihin, kehittäjä on ohjelmiston tarkoittanut. Ennen ohjelmiston käyttöä tai jakamista, lisensseihin kannattaa tutustua huolella. (Bandi 2019.)

Lisenssityypeistä yleisimpiä ovat MIT License ja Apache License 2.0. MIT License on luvallinen lisenssi, jonka ainut vaatimus on alkuperäisen tekijänoikeus ja lisenssi todennuksen säilyttäminen ohjelmiston kopiossa. Apache License 2.0 on myös luvallinen lisenssi, joka vaatii alkuperäisen tekijänoikeus ja lisenssi todennuksen säilyttämisen ohjelmiston kopiossa sekä kaikkien huomion arvoisten muutoksien dokumentointi. (Bandi 2019.)

Tässä opinnäytetyössä keskitytään ohjelmistokirjastoihin, joiden käyttötarkoituksena on skannata mobiililaitteen kameran avulla viivakoodeja. Skanneriohjelmistokirjastot voivat joko tuottaa tai lukea viivakoodeja. Skanneriohjelmistokirjaston avulla haetaan yhteys laitteen kameraan ja kameran videokuvan avulla luetaan viivakoodi ja palautetaan siitä saatu tulos. Skanneri tulkitsee liikkuvaa tai seisovaa kuvaa ja etsii siitä tunnettuja muotoja, jotka tässä tapauksessa ovat mustia ja valkoisia

pystyviivoja tai pisteitä. Tarvitut muodot löydettyään se tulkitsee ne ja antaa tulokseksi numero sarjan, jonka viivakoodi sisältää.

4 Tutkimusasetelma

Tässä luvussa käydään läpi opinnäytetyön taustaa, tavoitteita ja rajausta ja kerrotaan, miten opinnäytetyön tutkimusvaiheet toteutetaan sekä mitä tutkimusmenetelmää käytetään ja mikä opinnäytetyön tavoitteena on.

4.1 Tausta, tavoitteet ja rajaukset

Opinnäytetyön aihe on kehitetty Allergiaskanneri-projektissa ilmenneen ongelman pohjalta. Projektia on kehitetty Jyväskylän ammattikorkeakoulun Ticorporate-opintojaksojen toteutuksilla Demo Lab ja Product Lab. Allergiaskanneri on sovellus, minkä tarkoituksena on helpottaa ja nopeuttaa allergikoiden kaupassa käyntiä. Sovelluksesta löytyy valmis valikoima allergioita, joita voidaan asettaa päälle käyttäjien omien allergioiden mukaan. Sovellus tarjoaa myös skannerin, jonka avulla luetaan elintarvike tuotteiden viivakoodeja. Skannauksen jälkeen sovellus analysoi, mikäli skannattu tuote sisältää päälle asetettuja allergioita ja ilmoittaa käyttäjälle saadun tuloksen. (ks. kuvio 7)

Kuvio 7 Allergiaskanneri-sovelluksen etusivu

Allergiaskanneri projektissa käytetään valmista skanneriohjelmistokirjastoa, jonka avulla luetaan tuotteiden viivakoodit. Skanneriohjelmistokirjaston valinnassa ja käytössä ilmeni kuitenkin ongelmia, ja tulokset eivät aina olleet toivottuja. Nykyinen kirjasto on välillä hitaanpuoleinen ja pienetkin ympäristölliset ongelmat kuten vähäinen valo heikentävät skannauksen tuloksia huomattavasti. Ongelmia on myös ilmennyt kirjaston yhteensopivuudessa eri laitteiden välillä.

Tätä opinnäytetyötä varten ei kuitenkaan lähdetä ratkaisemaan kaikkia yllä mainittuja ongelmia, vaan keskitytään vain osa ongelmien tutkimiseen. Yhteensopivuutta eri laitteiden välillä ei lähdetä tutkimaan, koska sen toteutus muiden ongelmien lisäksi olisi aikaa vievämpää toteuttaa. Ongelma, joihin tullaan keskittymään, on nykyisen kirjaston hitaus ja epävarmuus, sekä ympäristöllinen toimivuus eli skannerin lukemisen suorituskyky erilaisissa ympäristöissä. Ympäristöillä tarkoitetaan eri valoisia tiloja. Opinnäytetyöllä ei ole toimeksiantajaa, joten työ on kehitetty itsenäisenä työnä oman mielenkiinnon herättämänä.

4.2 Tutkimuksen toteutus

Tutkimus toteutetaan vaiheissa, joita on yhteensä kolme. Ensimmäisessä vaiheessa laaditaan kriteeristö kolmen skanneriohjelmistokirjaston valintaan. Kriteeristö ei ole tarkka, vaan se on suuntaa antava. Ohjelmistokirjastoja etsitään GitHubin kautta, koska se tarjoaa yksinkertaisen ja nopean tavan etsiä vaadittavia ohjelmistokirjastoja. Kirjastojen valinnassa otetaan huomioon esimerkiksi kirjaston dokumentointi, aktiivisuus ja käyttäjämäärä. Huomion arvoista on myös se, että kirjasto tukee opinnäytetyössä esiintyviä tekniikoita. Ensimmäisen vaiheen lopussa valitaan kriteeristön avulla vertailtavat skanneriohjelmistokirjastot. Vertailtavia kirjastoja valitaan kolme kappaletta.

Ohjelmistokirjastojen valinnan jälkeen siirrytään toiseen vaiheeseen. Toisessa vaiheessa tutustutaan valittujen kirjastojen toiminnallisuuksiin syvemmin ja vertaillaan niiden käytänteitä ja selvitetään käyttömahdollisuuksia esimerkiksi Angularin kanssa kirjastojen omien dokumentointien ja GitHubista löydettävän ongelmien seuranta -välilehden avulla. Tutkintaan esimerkiksi sitä, että onko kirjastoa hankala käyttää tai liittää toiseen projektiin, millaisia ongelmia kirjaston kanssa esiintyy. Toisen vaiheen lopussa valitaan yksi skanneriohjelmistokirjasto, jota lähdetään empiirisesti testaamaan Allergiaskanneri-projektissa.

Kolmannessa vaiheessa käydään lyhyesti läpi valitun skanneriohjelmistokirjaston asennus ja käyttöönotto Allergiaskanneri-projektin pohjassa ja testataan sen toiminnallisuutta projektin yhteydessä. Testaamisvaiheessa selvitetään, mikäli uusi kirjasto tuottaa parempia tuloksia kuin vanha kirjasto.

4.3 Tutkimusmenetelmä

Opinnäytetyö kehitetään laadullisena eli kvalitatiivisena tutkimuksena. Tutkimusosuus aloitetaan luomalla kriteeristö, jonka avulla valitaan kolme skanneriohjelmistokirjastoa. Valittuihin kirjastoihin tutustutaan syvemmin ja niitä vertaillaan keskenään. Vertailun jälkeen valitaan vain yksi kirjasto, joka voisi soveltua Allergiaskanneri-projektiin sen nykyistä kirjastoa paremmin.

Tutkimusmenetelmäksi valikoitui laadullinen tutkimus, koska opinnäytetyön tarkoituksena on etsiä ja testata toista ohjelmistokirjaston ratkaisua Allergiaskanneri-projektissa. Tutkimusta lähetään avaamaan tutkimuskysymyksien pohjalta ja näihin kysymyksiin vastataan opinnäytetyön toteutuksessa.

Laadullisella tutkimuksella pyritään ymmärtämään ilmiötä eli selvitetään mistä tutkittavassa ilmiössä on kyse. Tutkimuksen tarkoituksena ei ole kerätä yleistävää tietoa ilmiöstä tai aiheesta vaan keskitytään enemmän yksittäisen tutkimuksen aineistoon. Pääsääntönä laadulliselle tutkimukselle on se, että mitä vähemmän tutkittavasta ilmiöstä tiedetään, sitä todennäköisemmin on kyseessä laadullinen tutkimus. Laadullisen tutkimuksen tavoitteena on tutkittavan ilmiön kuvaaminen ja ymmärtäminen. (Kananen 2017, 32–33, 35.)

Tilanteita, joissa voidaan käyttää laadullista tutkimusta ovat seuraavanlaisia: ilmiöt, joilla ei ole ennestään kerättyä tietoa tai tutkittavasta ilmiöstä halutaan saada syvällisempi näkemys. Ilmiöstä halutaan luoda uusia teorioita ja hypoteeseja. Tutkimuksessa käytetään triangulaatiota eli monimenetelmäistä tutkimusasetelmaa, jossa käytetään erilaisia lähestymistapoja ilmiön ymmärtämiseksi. Tilanne voi olla myös se, että tutkittavasta ilmiöstä halutaan selkeä kuvaus. (Kananen 2017, 33.) Tämän opinnäytetyön tutkimus soveltuu tilanteeseen, jossa halutaan luoda uusia teorioita ja hypoteeseja.

Laadullisella tutkimuksella on ominaisia piirteitä, joita tutkimuksessa voidaan nostaa esille. Muutamia mainittavia piirteitä ovat esimerkiksi se, että tutkimus tehdään aidoissa ympäristöissä ja tilanteissa kuten skannerin testaaminen laitteella hämärässä huoneessa tai tilassa. Tutkimusaineisto on yleensä monilähteistä kuten dokumentaatiota tai kuvia. (Kananen 2017, 34.)

Laadullisessa tutkimuksessa keskitytään paljon tutkimuksen prosessiin, merkitykseen ja ilmiön ymmärtämiseen tekstien ja kuvien avulla. Siksi menetelmän tutkimus tyyli on kuvailevaa ja tutkimuksessa edetään usein yksittäisistä havainnoista seuraaviin havaintoihin ja lopulta tuloksiin. (Kananen 2017, 36.) Tämän opinnäytetyön tutkimus toteutetaan vaiheittain eli siirrytään havainnosta toiseen.

Ilmiötä johdatetaan ja lähdetään ratkaisemaan tutkimuskysymyksien kautta. Tutkimuskysymyksiin vastaamalla ratkaistaan tutkimuksen ongelma.

4.4 Tutkimuskysymykset

Opinnäytetyön tutkimuskysymykset ovat:

1. Millaisia skanneriohjelmistokirjastoja on olemassa vaadituille teknologioille?
2. Kuinka asennetaan ja käyttöönotetaan uusi kirjasto valmiissa projektissa?
3. Saadaanko uudella skanneriohjelmistokirjastolla parempia tuloksia?

Ensimmäiseen tutkimuskysymykseen vastataan etsimällä valmiita skanneriohjelmistokirjastoja GitHubista kriteeristön avulla. Kriteeristöllä varmistetaan, että kirjasto on yhteensopiva Allergiaskanneri-projektin kanssa. (ks. Kirjastojen valinta, luku 5.1; Kirjastojen analysointi ja vertailu, luku 5.2.)

Toiseen tutkimuskysymykseen vastataan asentamalla ja käyttöönottamalla uusi kirjasto ja kerrotaan sen prosessista (ks. Kirjaston asennus ja käyttöönotto, luku 5.3).

Kolmanteen tutkimuskysymykseen vastataan uuden kirjaston käyttöönoton jälkeen. Sekä uutta että vanhaa kirjastoa testataan viivakoodeilla, jotka on printattu erilaisille pakkausmateriaaleille.

Testaus tehdään myös eri valoisissa ympäristöissä ja selvitetään tuottaako uusi kirjasto testaus vaiheessa parempia tuloksia kuin vanha kirjasto. (ks. Käyttöönotetun kirjaston testaus ja vertaaminen, luku 5.4.)

5 Ohjelmistokirjaston valinta, asentaminen ja käyttöönotto

Tässä luvussa laaditaan kriteeristö kirjastojen valintaan ja vertaillaan kriteeristön avulla valittuja kirjastoja keskenään. Vertailun pohjalta valitaan lopulta vain yksi kirjasto, jota lähdetään käyttöönottamaan opinnäytetyön empiirisessä vaiheessa.

5.1 Ohjelmistokirjastojen valinta

Ensimmäisenä lähdetään toteuttamaan opinnäytetyön ensimmäistä vaihetta eli kirjastojen valintaa. Ensimmäisen vaiheessa laaditaan kevyt kriteeristö, minkä pohjalta valitaan kolme skanneriohjelmistokirjastoa, jotka tukevat kriteeristöä. Kriteeristö koostuu viidestä osasta, jotka kerrotaan ovat seuraavanlaisia; Kirjastolla täytyy olla omaa dokumentaatiota, joka tekee kirjaston asentamisesta sujuvampaa ja antaa käyttäjälle alustavan ymmärryksen kirjaston käyttömahdollisuuksista. Kirjaston täytyy tukea lineaarisia viivakoodeja, joita ovat muun muassa EAN-13 ja EAN-8. On myös tärkeää, että kirjasto antaa tukea vaadittaville teknologioille, jotka ovat tässä opinnäytetyössä Angular ja Progressive Web App. Kirjaston valinnassa täytyy ottaa huomioon, että kirjasto on edelleen aktiivisessa toiminnassa eli kirjastolle tehdään ajankohtaisia päivityksiä. Viimeisenä kriteerinä on, että kirjastolla on suhteellinen määrä käyttäjien antamia tähtiä. Suhteellinen tässä tapauksessa on yli 100 tähteä.

Kirjastoja etsitään GitHubin avulla, koska GitHub tarjoaa kaikki tarvittavat tiedot kriteeristöä varten. Etsimisessä käytetään GitHubin omaa hakukenttää, johon hakusanoina käytetään ”barcode scanner”, ”ngx barcode scanner” ja ”barcode reader”. Tämän lisäksi haun lajitteluksi asetetaan tähtien määrä, joita GitHubin käyttäjät pystyvät kirjastoille antamaan. Tähtien määrällä voidaan seurata kirjaston suosiota suuruusjärjestyksessä. Yhdessä hakusanoista käytetään lyhennettä ngx, joka viittaa Angulariin.

Parhaimmat tulokset kirjastoihin löytyivät hakusanalla ”ngx barcode scanner”, koska se sisältää suoran viittauksen Angulariin ja antaa siksi jo hakuvaiheessa tiedon siitä, että kirjasto tukee tässä

opinnäytetyössä vaadittavia teknologiota. Koska Angular on web-kehitykseen luotu ohjelmistokehityksen työkalu, tiedetään suoraan, että kirjasto antaa tukea myös PWA:lle.

Kokonaisuudessaan kirjastoja löytyi hakusanoilla useita satoja, mutta useat niistä eivät olleet suuressa suosiossa tai käytössä. Suurin osa tarjotuista kirjastoista oli myös haaroitettu muista suosituimmista kirjastoista. Kirjastojen tukemat viivakoodityypit löytyivät suoraan kirjastojen omista dokumentoinneista. Dokumentaatiot löytyvät suoraan kirjastojen omilta GitHub-sivuilta.

GitHubin avulla löydettiin kolme kirjastoa, jotka tukevat kriteeristön vaatimuksia. Kirjastot ovat ZXing, html5-qrcode ja Quagga. Seuraavassa luvussa analysoidaan ja syvennytään valittuihin kirjastoihin tarkemmin ja tutustutaan, että miten kyseiset kirjastot tukevat kriteeristöä.

5.2 Kirjastojen analysointi ja vertailu

Tässä luvussa kerrotaan mitkä kolme kirjastoa täyttävät kaikki edellisen luvun kriteeristön vaiheet ja analysoidaan kirjastojen tarjontaa pääosittain kirjastojen tarjoaman dokumentoinnin avulla. Luvun lopussa vertaillaan kirjastoja keskenään ja vertailun avulla valitaan vain yksi kirjasto, jota lähdetään käyttöönottamaan Allergiaskanneri-projektin lähdekoodin pohjaan empiirisesti.

Ensimmäinen valittu skannerikirjasto on nimeltään ZXing. ZXing on monimuotoinen avoimen lähdekoodin kirjasto, joka soveltuu useiden lineaaristen ja kaksiulotteisien viivakoodien käsittelyyn. Kirjasto käyttää Apache 2.0 lisenssiä. Alustavasti ZXing on tehty Java-kielellä, mutta ensimmäisen kirjaston suosion kasvaessa, on kirjastosta kehitetty useita versioita eri ohjelmointikielille, jotka ovat yhteensopivia eri teknologioiden kanssa. ZXingille löytyy myös haaroitettu (engl. forked) kirjasto, joka on kehitetty suoraan Angular-projekteille. Sivukirjasto löytyy GitHubista nimellä zxing/ngx-scanner. zxing/ngx-scanner tarvitsee muita ZXing kirjaston haaroja toimiakseen, mutta dokumentointi sisältää selkeän ohjeistuksen kaikkiin vaadittaviin ZXing-kirjaston osiin, joita sen käyttöönotto vaatii. (zxing-js/library n.d.)

ZXing-kirjasto otetaan käyttöön projektin HTML-dokumentissa. Tätä varten ZXingilla on oma HTML-tunniste (engl. tag), jolle löytyy omia valmiita attribuutteja (ks. kuvio 8). Kun ZXing HTML-tunniste lisätään projektin HTML-dokumenttiin, ottaa se yhteyden laitteen kameraan MediaDevices-rajapinnan sisältämän getUserMedia-metodin avulla. HTML-tunniste sisältää myös ratkaisun

viivakoodien lukemiseen. HTML-tunnisteen attribuuttien avulla voidaan lisätä uusia ominaisuuksia skannerille, kuten luettavien viivakoodityyppien rajoittamisen. Valmiita attribuutteja on yhteensä 11, mutta attribuutteja voidaan luoda tarvittaessa lisää. Enable-attribuutti on yksi esimerkki valmiista attribuutista, jonka tehtävänä on aloittaa ja lopettaa skannaaminen. Muita attribuuttien toiminnallisuuksia ovat esimerkiksi laitteen salamavalon käyttöönotto, kameraa ei löydy, skannauksen onnistuminen tai epäonnistuminen jne. (zxing-js/library n.d.)

ZXing-kirjasto on käytössä Allergiaskanneri-projektin nykyisessä versiossa. Kirjasto otettiin mukaan tähän vaiheeseen, koska muita kriteeristä tukevia ohjelmistokirjastoja ei löytynyt.

```
7   <zxing-scanner
8     [device]="currentDevice"
9     (scanSuccess)="update($event)"
10    [formats]="allowedFormats"
11    [torch]="torchService.torchEnabled"
12    (camerasFound)="onCamerasFound($event)"
13    (permissionResponse)="onHasPermission($event)"
14    (torchCompatible)="torchService.onTorchCompatible($event)"
15  ></zxing-scanner>
```

Kuvio 8. ZXing-kirjaston HTML-tunniste ja attribuutteja

Seuraava kirjasto on Quagga. Alkuperäinen Quagga-kirjasto ei ole enää jatkuvassa ylläpidossa, mutta kyseisestä kirjastosta on kehitetty jatkoprojekti nimeltään Quagga2, joka on edelleen kehityksessä. Alkuperäisen Quagga-kirjaston dokumentointi ohjaa Quagga2-kirjastoon, joten voidaan päätellä, että kirjaston alkuperäinen kehittäjä on mukana uudessa kehityksessä tai tukee Quagga2-kirjaston kehitystä. Tässä opinnäytetyössä käytetään jatkossa sanaa Quagga, vaikka puhutaan Quaggan uudemmasta versiosta eli Quagga2-kirjastosta. Kirjaston lisenssi on MIT License. (ericblade/quagga2 n.d.)

Quagga-kirjastolle löytyy haara nimeltään ngx-barcode-scanner, joka hyödyntää Quagga-kirjastoa ja se on kehitetty suoraan Angular projekteille. Quaggan dokumentointi ohjaa suoraan ngx-barcode-scanner haaraan. Quaggalle löytyy ZXingin tavoin oma HTML-tunniste, jota käytetään HTML-

dokumentissa (ks. kuvio 9). Quaggan dokumentoinnista löytyy alustava ohjeistus kirjaston käyttöönottoon. (ericblade/quagga2 n.d.) Quagga on ennenkin ollut lyhyesti käytössä Allergiaskanneri-projektissa, mutta ilman ngx-barcode-scanner haaraa.

```
14  <barcode-scanner-livestream
15    type="ean"
16    (valueChanges)="onValueChanges($event)"
17    (started)="(onStarted)">
18  </barcode-scanner-livestream>
```

Kuvio 9. Quagga-kirjaston HTML-tunniste

Viimeinen kirjasto on nimeltään html5-qrcode. Vaikka kirjaston nimessä käytetään suoraan viittausta kaksiulotteisen viivakoodin tyyppiin, soveltuu kirjasto nykyään myös lineaaristen viivakoodien lukemiseen. Tämän kirjaston dokumentoinnissa ei selkeästi mainita sen yhteensopivuutta Angulariin, mutta se tukee useita muita ohjelmistokehitysalustoja ja ei suoraan sulje pois Angularia. Kirjaston oma dokumentointi on laajaa ja kertoo kirjaston käyttöönoton vaatimuksista yleisesti, mutta ei tarjoa suoraa ohjeistusta Angular-projekteihin. html5-qrcode käyttää ZXing-kirjastoa viivakoodien tulkintaan. Kirjaston lisenssi on Apache License 2.0. (mebjas/html5-qrcode n.d.)

html5-qrcode kirjastoa ei oteta käyttöön tässä opinnäytetyössä, koska verrattuna muihin kirjastoihin, se ei anna suoraa ohjeistusta ja tukea Angular-projekteille ja se käyttää ZXingin toteutusta viivakoodien tulkintaan. ZXing ja Quagga tarjoavat paljolti samoja attribuutteja ja kirjaston käyttäminen projektissa on hyvin samantapaista. Empiirisessä osuudessa lähdetään kuitenkin asentamaan ja käyttöönottamaan Quagga-kirjastoa, koska ZXing-kirjaston ngx-scanner haaraa on osa nykyistä Allergiaskanneri-projektia. Quaggan ngx-barcode-scanner haara ei ole ennestään ollut käytössä projektissa. Jotta nykyistä ja tätä tutkimusta varten valittua kirjastoa voidaan vertailla keskenään tutkimuksen lopussa, valikoituu uudeksi kirjastoksi Quagga.

5.3 Kirjaston asennus ja käyttöönotto

Seuraavaksi asennetaan ja otetaan käyttöön Quagga-kirjasto Allergiaskanneri-projektissa. Asennusprosessi käydään läpi vaiheittain. Vaiheet ovat pääasiassa tiedostokohtaisia muutoksia.

Kirjastojen asentaminen aloitetaan npm-komennolla install, minkä avulla asennetaan kaksi vaadittua kirjastoa, jotka ovat quagga2 ja ngx-barcode-scanner (ks. kuvio 10). Allergiaskannerissa on käytössä Angular-ohjelmistokehyksen 11 versio, joten asennuksessa täytyy ottaa huomioon asennettujen kirjastojen yhteensopivuus Angular version kanssa. Quaggan dokumentaatio tarjoaa vaatimattoman taulukon, joka esittelee versioiden yhteensopivuudet Angular 10 ylöspäin.

```
$ npm install @ericblade/quagga2@1.2.6 ngx-barcode-scanner@0.2.0
```

Kuvio 10. Komento kirjaston asennukseen

Kirjaston asentamisen jälkeen importoidaan BarcodeScannerLivestreamModule-luokka oman projektin moduuli-tiedostoon (ks. kuvio 11). Moduuli-tiedosto pitää sisällään Angularin oman modulaarisuus järjestelmän nimeltään NgModules. NgModules on luokka, mihin importoidaan BarcodeScannerLiveStreamModule-luokka Quagga-kirjaston käyttöönottoa varten. NgModules importoinnin avulla havainnoidaan importoidun luokan omat komponentit, jotta niitä pystytään käyttämään projektissa myöhemmin. Tämä vaihe suoritetaan aina kun asennetaan ulkoisia kirjastoja.

```
18  import { BarcodeScannerLivestreamModule } from 'ngx-barcode-scanner';
19
20
21  @NgModule({
22    imports: [
23      BarcodeScannerLivestreamModule,
```

Kuvio 11. Kirjaston importointi moduuli-tiedostossa

Moduuli-tiedoston jälkeen siirrytään komponentin TypeScript-tiedostoon, johon myös importoidaan ensimmäisenä tarvittavat luokat ja dekoraattorit (engl. decorator) ja otetaan ne käyttöön saman komponentin luokassa mihin importoinnit tehtiin. Alustavat importoinnit, joita Quagga-kirjaston käyttöönotto vaatii ovat ViewChild ja BarcodeScannerLiveStreamComponent (ks. kuvio 12).

```
17  import { Component, AfterViewInit, ViewChild } from '@angular/core';
18  import { BarcodeScannerLivestreamComponent } from 'ngx-barcode-scanner';
```

Kuvio 12. Kirjaston importoinnit komponentti-tiedostossa

Importoidut luokat ja rajapinnat otetaan käyttöön komponentin omassa TypeScript-tiedoston luokassa. ViewChild-dekoraattori palauttaa BarcodeScannerLivestreamComponent-luokan. Palautuksen jälkeen voidaan kutsua BarcodeScannerLivestreamComponent-luokan metodeja ja ottaa ne käyttöön komponentissa. Dekoraattorin palautettu luokka asetetaan myös barcodeScanner-muuttujan arvoksi.

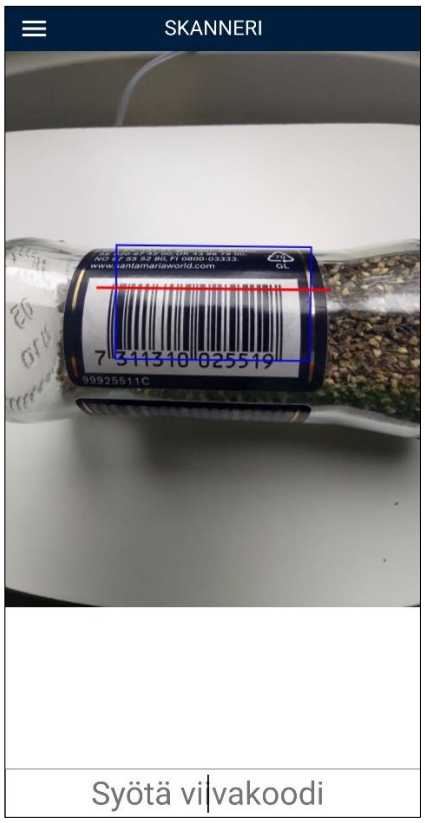
Projektin omassa luokassa luodaan kaksi uutta metodia onValueChanges ja onStarted. onValueChanges-metodi seuraa skannaus tulosta ja palauttaa sen. onStarted-metodi ilmoittaa kehittäjälle konsolissa, että kameran käyttöönotto on tapahtunut. onValueChanges palauttaa myös konsoliin luetun viivakoodin tuloksen. ngAfterViewInit-metodin sisällä alustetaan skannerin käyttöönotto. (ks. kuvio 13.)

```
26  export class ScannerComponent implements AfterViewInit {
27
28    @ViewChild(BarcodeScannerLivestreamComponent)
29    barcodeScanner: BarcodeScannerLivestreamComponent;
30
31    barcodeValue = '';
32
33    ngAfterViewInit() {
34      this.barcodeScanner.start();
35
36      this.ChechkAllergies();
37    }
38
39    onValueChanges(result) {
40      this.barcodeValue = result.codeResult.code;
41    }
42
43    onStarted(started) {
44      console.log(started);
45    }
```

Kuvio 13. Komponentin TypeScript-luokan metodit ja muuttujat

HTML-dokumentissa otetaan käyttöön Quaggan kustomoitu HTML-tunniste. HTML-tunnisteessa asetetaan viivakoodityypin tunnus Type-attribuuttiin, millä pystytään rajaamaan sitä, että mitä viivakoodityyppejä skannerin tulee tunnistaa. Type-attribuutin arvoksi asetetaan EAN. valueChanges ja started attribuuttien arvoiksi asetetaan komponentti luokassa luodut metodit onValueChanges ja onStarted. (ks. kuvio 9)

Kirjaston asennuksen ja käyttöönoton jälkeen projektissa on toimiva skanneriohjelmisto, joka avaa laitteen kameran ja lukee EAN-viivakoodeja. Skannerin kameran asettelu on kuitenkin tässä vaiheessa epämääräinen, sillä se ei automaattisesti skaalaudu näyttöön sopivaksi. (ks. kuvio 14)



Kuvio 14 Skanneri Qugga-kirjaston käyttöönoton jälkeen

5.4 Käyttöönotetun kirjaston testaus ja vertaaminen

Uuden kirjaston asennuksen ja käyttöönoton jälkeen lähdetään testaamaan uuden ja vanhan kirjaston toiminnallisuutta eli viivakoodien lukemista ja analysoidaan, mikäli uudella kirjastolla saadaan parempia tuloksia. Kirjastojen testaus suoritetaan muutamalla mobiililaitteella eri valoisissa ympäristöissä. Ympäristöjen valoisuudet, joissa testaus suoritetaan ovat luonnollinen valolähde, keinotekoinen valonlähde, heikosti valaistu ympäristö, missä ei ole suoraa valonlähdettä ja viimeisenä hämärä ympäristö.

Kirjaston tuloksien seurannalla ja tulosteilla on eroja testaus vaiheessa. Vanha kirjasto muuttaa viivakoodin tulosta livenä eli kirjasto lukee viivakoodia jatkuvalla syötteellä ja jos uusi tulos eroaa vanhasta, muuttuu tuloksen arvo uuteen tulokseen. Vanha kirjasto taas antaa vain yhden näkyvän tuloksen, vaikka todennäköisesti sama prosessi tapahtuu syötteen takana. Vanha kirjasto, joka on käytössä Allergiaskannerissa ja johon uutta kirjastoa verrataan, on ZXing. ZXing-kirjastosta mainittaessa käytetään tästä eteenpäin termiä vanha kirjasto ja Quaggasta termiä uusi kirjasto.

Viivakoodeja testattiin kolmella eri pakkauksella. Ensimmäinen viivakoodi oli tasaisella alustalla valkoisella pohjalla mustilla viivoilla kartonkipakkauksessa. Toinen viivakoodi oli pyöreällä alustalla, valkoisella pohjalla, mustilla viivoilla ja mattapintainen. Kolmas viivakoodi oli kiiltävällä alustalla, valkoisella pohjalla, tumman sinisillä viivoilla ja viivat oli aseteltu tiiviisti yhteen.

Ensimmäisen viivakoodin lukeminen luonnon valossa toimi molemmilla kirjastoilla hyvin. Luettu viivakoodi oli oikein ja tapahtui ensimmäisen sekunnin sisällä. Sama toistui kaikissa valoisissa ympäristöissä. Ainoa kerta, jolloin viivakoodin lukeminen ei onnistunut oli vanhalla kirjastolla kokonaan hämärässä ympäristössä.

Toisen viivakoodin lukeminen onnistui uudella kirjastolla oikein ja nopeasti kaikissa ympäristöissä. Vanha kirjasto luki luonnon valossa hyvin, mutta keinotekoisesti valaistussa ja heikosti valaistussa ympäristöissä tulos oli oikein, mutta hieman uutta kirjastoa hitaampi. Hämärässä tilassa tulos oli myös oikein, mutta vielä muita ympäristöjä hitaampi.

Kolmannen viivakoodin lukemisessa oli molemmilla kirjastoilla eniten haasteita. Uusi kirjasto luki viivakoodin oikein valoisemmissa tiloissa, mutta tulos oli epävakaa eli se muuttui sekunnin välein oikeasta tuloksesta väärään tulokseen ja takaisin oikeaan ja taas väärään. Hämärässä ympäristössä lukeminen ei onnistunut ollenkaan ja tulokset olivat useammin virheellisiä kuin oikeellisia. Vanhan kirjaston tulokset eivät olleet parempia. Aluksi lukeminen ei onnistunut missään ympäristössä ollenkaan eli skanneri ei antanut minkäänlaisia tuloksia. Hetken odottelun jälkeen valoisissa tiloissa saatiin lopulta tuloksia, mutta suurin osa tuloksista oli kuitenkin virheellisiä.

Yhteenvetona testauksen tuloksista voidaan tässä vaiheessa sanoa, että uusi kirjasto tuottaa parempia tuloksia kuin vanha kirjasto. Suurin ero tuloksissa näkyi hämärässä ympäristössä, mistä uusi kirjastoa suoriutui hieman paremmin. Uusi kirjasto onnistui myös paremmin kiiltäväpohjaisen ja tiiviimmin asetellun viivakoodin lukemisessa.

Lukutuloksien lisäksi uusi kirjasto tarjoaa valmiin ominaisuuden missä kirjasto näyttää visuaalisesti käyttäjälle löytääkö skanneri video syötteestä viivakoodia vai ei ja lukeeko se sitä. (ks. kuvio 15) Vanhalla kirjastolla ei ole kyseistä ominaisuutta valmiina.

Kuvio 15. Quaggan skannauksen visuaalinen näkymä ja siitä saatu tulos

6 Tulokset

Tässä luvussa kerrotaan tiivistetysti tutkimuksessa saaduista tuloksista ja vaiheista.

6.1 Kirjaston valinta ja analyysi

Toteutuksen ensimmäisessä aliluvussa valittiin kolme skanneriohjelmistokirjastoa tässä opinnäytetyössä luodun kriteeristön avulla. Valittuja kirjastoja vertailtiin keskenään pintapuolisesti niiden dokumentoinnin avulla. Toista kriteeristöä yhden kirjaston valintaan ei luotu, koska ensimmäinen kriteeristö takaa, että kaikki kirjastot ovat yhteensopivia Allergiaskannerin kanssa.

Kolmen kirjaston joukosta valittiin yksi kirjasto, jota lähdettiin myöhemmin asentamaan Allergiaskanneri-projektin koodipohjaan. Asennettavaksi kirjastoksi valikoitui Quagga. Quagga-kirjasto valikoitui, koska sen dokumentointi antaa selvän ohjeistuksen Angularin kanssa toimimiseen. Quaggan dokumentointi antaa myös ohjeita kirjaston alustavaan käyttöönottoon. Quagga-kirjaston ns. pääkirjastoa on ennenkin kokeiltu Allergiaskanneri projektissa, mutta sen sivukirjastoa ngx-barcode-scanner, joka on suunnattu suoraan Angular projekteille, ei ole ennen ollut käytössä.

6.2 Asennus, käyttöönotto ja testaaminen

Quagga-kirjaston dokumentointi antaa alustavan ohjeistuksen kirjaston käyttöönottoon Angular projekteissa ja tätä ohjeistusta seuraamalla kirjasto saatiin otettua käyttöön Allergiaskanneri projektissa onnistuneesti. Alustavan asennuksen jälkeen lopputulos ei kuitenkaan vielä näyttänyt ulkoisesti täysin toivotulta, koska kamera ei automaattisesti skaalautunut laitteen näyttöön sopivaksi vaan otti haltuun vain osan näytön tilasta (ks. kuvio 14). Kameran skaalaukseen ei löytynyt suoraa toimivaa ratkaisua Quaggan dokumentoinnista.

Uuden ja vanhan kirjaston testaus suoritettiin lukemalla eri viivakoodeja eri valoisissa ympäristöissä. Testauksen jälkeen tultiin siihen tulokseen, että uusi kirjasto toimii vanhaa kirjastoa paremmin. Erityisesti hämärässä ympäristössä lukeminen onnistui uudella kirjastolla paremmin. Vanhalla kirjastolla oli myös haasteita lukea viivakoodia, jos viivakoodi oli printattu kiiltävälle pohjalle.

Ylimääräisenä huomiona nousi esiin se, että uudessa kirjastossa lukutulos muuttui nähtävästi, mikäli edeltävä tulos ei vastannut uutta tulosta. Huomioitavaa on myös se, että usein huonommin valaistuissa ympäristöissä Quaggan ensimmäiset tulokset olivat usein virheellisiä. Vanha kirjasto luki vain yhden tuloksen ja palautti siitä saadun arvon. Viivakoodien jatkuva lukeminen ja sen tuloksen näyttäminen ei ole kuitenkaan tarpeellista Allergiaskanneri-projektissa, koska Allergiaskanneri vastaanottaa aina vain yhden viivakoodin ja palauttaa sitä viivakoodia vastaavan elintarviketuotteen tiedot. Ratkaisuna tälle ongelmalle olisi ajan rajoittaminen eli jos viivakoodin tulos ei muutu tietyn ajan sisällä, palautetaan tulos vasta sen jälkeen.

Toisena isona huomiona nousi esiin se, että Quagga-kirjastoon on sisäänrakennettu ominaisuus, joka näyttää visuaalisesti vihreillä tai sinisillä ja punaisilla väreillä mitä kirjasto lukee kamera syötteessä. Tämä auttaa antamaan käsityksen skannerin kokoaikaisesta käytöstä ja antaa käyttäjälle suoran näytön siitä, että skanneri on toiminnassa. Allergiaskannerin nykyinen kirjasto ei tarjoa kyseistä ominaisuutta. Ominaisuus voi olla hyödyllinen tilanteissa, joissa skannaus saattaa kestää kauemmin ja käyttäjä on epätietoinen siitä, mikäli skanneri on ollenkaan toiminnassa. Tämä ominaisuus tekee Quaggasta käyttäjäystävällisemmän.

7 Pohdinta

Opinnäytetyön tavoitteena oli etsiä kriteeristön avulla kolme uuttaa skanneriohjelmistokirjastoa, joista lopulta valittiin vain yksi, jota lähdettiin empiirisesti asentamaan, käyttöönottamaan ja testaamaan valmiissa Allergiaskanneri-projektin pohjassa. Tuloksena löydettiin yksi uusi kirjasto, joka tuki laadittua kriteeristöä ja uusi kirjasto otettiin käyttöön projektissa. Testauksen jälkeen tultiin siihen tulokseen, että uusi kirjasto antaa parempia tuloksia kuin vanha kirjasto.

Opinnäytetyön tulokset jäivät osittain heikoiksi. Empiirisen osuuden testaaminen olisi vaatinut enemmän testaamista ja aineistoa, jotta olisi voitu päästää luotettavampiin tuloksiin. Vaikka uusi kirjasto toimii testauksen perusteella rajatuissa ympäristöissä paremmin kuin vanha, tarvitaan myös enemmän koodin muokkaamista projektikohtaisemmaksi, jotta saadaan selville tarkemmalla varmuudella, että uusi kirjasto ei vain tuota parempia viivakoodien tuloksia, vaan se toimii kokonaisuudessaan paremmin projektin sisällä. Kirjastojen etsimisessä, valinnassa ja analysoinnissa onnistuttiin hyvin kriteeristön avulla. Kirjastojen dokumentointien avulla saatiin hyvin kirjasto kohtaista aineistoa, minkä avulla kirjastojen valinta ja vertailu onnistui hyvin.

Opinnäytetyössä ei ollut haasteita eettisyyden ylläpidossa, koska työssä ei käsitelty henkilötietoja tai salassa pidettävää tietoa. Muille Allergiaskannerin kehittäjille ilmoitettiin sanallisesti, että opinnäytetyön aihe on kehitetty Allergiaskannerin pohjalta. Kaikki aineisto mitä opinnäytetyössä on kerätty, on vapaasti käytettävissä.

Opinnäytetyön tutkimuksen tuloksia voidaan hyödyntää Angular-projekteissa, joissa tarvitaan skanneria viivakoodien lukemiseen. Tutkimus tarjoaa vaihtoehtoja skanneriohjelmistokirjaston valintaan. Tuloksia voidaan hyödyntää myös Allergiaskanneri-projektissa, jos sitä halutaan lähteä tulevaisuudessa jatkokehittämään. Tämän työn kirjoittamisen aikana Allergiaskanneri-projektia ei ole kehitetty eteenpäin.

Lähteet

2D-viivakoodit käyttöön viimeistään 2027. 2022. Artikkeli 2D-viivakoodien käyttöönotosta kaupallisuudessa GS1 Finland Oy:n verkkosivuilta. Viitattu 30.3.2022. https://gs1.fi/fi/uutiset/2d-viivakoodit-kayttoon-viimeistaan-2027-0

Angular Modules and NgModule - Complete Guide. Blogiteksti Angular verkkosivulta. Viitattu 1.11.2022 https://blog.angular-university.io/angular2-ngmodule/

Angular Routing. 2022. Dokumentti Angularin verkkosivuilta. Viitattu 8.4.2022. https://angular.io/guide/routing-overview

Bandi, M. 2019. All About Open Source Licenses. Artikkeli FOSSA, Inc verkkosivulla. Viitattu 23.3.2022. https://fossa.com/blog/what-do-open-source-licenses-even-mean/

Blank, A. 2004. TCP/IP Foundations. San Francisco ; London: Sybex. Viitattu 7.4.2022. https://janet.finna.fi, Skillsoft Books ITPro.

Dependency injection in Angular. 2022. Dokumentti Angularin verkkosivuilta. Viitattu 8.4.2022. https://angular.io/guide/dependency-injection

Drobnik, O. 2015. Barcodes with iOS: Bringing Together the Digital and Physical Worlds. Shelter Island: Manning. Viitattu 9.2.2022. https://janet.finna.fi, Skillsoft Books ITPro.

ericblade/quagga2. N.d. Github tietolähde. Viitattu 4.10.2022. https://github.com/ericblade/quagga2

Grigsby, J. 2018. Progressive Web App. A Book Apart 2018. Viitattu 20.4.2022. https://janet.finna.fi, Skillsoft Books ITPro.

GS1-standardit viivakoodeille ja RFID-tunnisteille. N.d. Artikkeli GS1 Finland Oy:n verkkosivulla. Viitattu 8.2.2022. https://gs1.fi/fi/ohjeet/yritystunniste/gs1-standardit-viivakoodeille-ja-rfid-tunnisteille

HTTP. 2021. Dokumentti HTTP protokollasta MDN verkkosivustolla. https://developer.mozilla.org/en-US/docs/Web/HTTP

Kananen, J. 2017. Laadullinen Tutkimus Pro Graduna Ja Opinnäytetyönä. Jyväskylä: Jyväskylän ammattikorkeakoulu. Viitattu 16.11.2022. https://janet.finna.fi, Booky

Keith, J. 2017. What is Progressive Web App? Blogiteksti Jeremy Keithin verkkosivulta. Viitattu 20.4.2022. https://adactio.com/journal/13098

Kumar, D. 2019. Angular Essentials: The Essentials Guide to Learn Angular. Delhi: BPB Publications. Viitattu 8.4.2022. https://janet.finna.fi, Skillsoft Books ITPro.

mebjas/html5-qrcode. N.d. Github tietolähde. Viitattu 4.10.2022 https://github.com/mebjas/html5-qrcode

Mikä on progressiivinen verkkosovellus (PWA) ja mitä etuja se tarjoaa? Blogi kirjoitus evermade verkkosivulla. Viitattu 1.11.2022https://www.evermade.fi/fi/artikkeli/mika-on-progressiivinen-verkkosovellus-pwa-edut/

NgModules. 2022. Dokumentti Angularin verkkosivulta. Viitattu 4.10.2022. https://angular.io/guide/ngmodules

Paremman huomisen puolesta – supervoimana standardit. N.d. Artikkeli GS1 Finland Oy:n toiminnasta verkkosivulla. Viitattu 21.2.2022. https://gs1.fi/fi/gs1-finland

Peterson, C. 2018. How I coined the term 'open source'. opensource.com verkkosivut. Viitattu 21.2.2022. https://opensource.com/article/18/2/coining-term-open-source-software

Russell, A. 2015. Progressive Web Apps: Escaping Tabs Without Losing Our Soul. Artikkeli PWA käytänteistä Alex Russelin verkkosivuilta. Viitattu 20.4.2022. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/

Shacklett, M. E., Novotny, A & Gerwig, K. 2021. TCP/IP. Viitattu 7.4.2022. https://www.techtarget.com/searchnetworking/definition/TCP-IP

Software Framework. 2018. Dokumentti ohjelmistokehyksistä techopedian verkkosivulla. Viitattu 8.4.2022. https://www.techopedia.com/definition/14384/software-framework

Stazzone, S. 2022. Guide to Barcode Types and Standards: 1D, 2D Barcode Symbologies, Requirements, and Standards-Issuing Entities. Artikkeli viivakoodityypeistä Camcode organisaation verkkosivulla. Viitattu 9.2.2022. https://www.camcode.com/blog/guide-to-barcode-types-standards/

Tapala, K. 2016. Mitä web-kehitys on. Artikkeli Karhu Helsinki Oy:n verkkosivuilla. Viitattu 7.4.2022. https://www.karhuhelsinki.fi/blogi/mita-web-kehitys

Types of Barcodes: Choosing the Right Barcode. 2015. Artikkeli viivakoodityypeistä Scandit organisaation verkkosivulla. Viitattu 9.2.2022. https://www.scandit.com/blog/types-barcodes-choosing-right-barcode/

What is DNS. N.d. Dokumentti DNS nimipalvelujärjestelmästä AWS Inc verkkosivulla. Viitattu 7.4.2022. https://aws.amazon.com/route53/what-is-dns/

What is open source. N.d. opensource.com verkkosivut. Viitattu 16.3.2022. https://opensource.com/resources/what-open-source

What Is Web Development. N.d. Dokumentti web-kehityksen periaatteista BrainStation Inc verkkosivuilta. Viitattu 7.4.2022. https://brainstation.io/career-guides/what-is-web-development

zxing-js/library. N.d. Github tietolähde. Viitattu 4.10.2022. https://github.com/zxing-js/library