OULUN AMMATTIKORKEAKOULU

Mika Huotari

MOBIILISOVELLUKSEN TOTEUTUS IONIC-SOVELLUSKEHYKSEN AVULLA

Case: Liikuntapaikka-sovelluksen kehittäminen avointa rajapintaa hyödyntämällä

MOBIILISOVELLUKSEN TOTEUTUS IONIC-SOVELLUSKEHYKSEN AVULLA

Case: Liikuntapaikka-sovelluksen kehittäminen avointa rajapintaa hyödyntämällä

Mika Huotari
Opinnäytetyö
Kevät 2020
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma, Digitaalisten palveluiden suuntautumisvaihtoehto

---

Tekijä(t): Mika Huotari
Opinnäytetyön nimi: Mobiilisovelluksen toteutus Ionic-sovelluskehyksen avulla
Työn ohjaaja: Jouni Juntunen
Työn valmistumislukukausi ja -vuosi: Kevät 2020 Sivumäärä: 40

Tämän opinnäytetyön tarkoituksena on tutkia mobiilisovelluksen toteuttamista Ionic-sovelluskehyksen avulla. Työssä käydään läpi Ionicin historiaa, tärkeimpiä ominaisuuksia sekä ympäristöön kytkeytyviä sovelluskehyksiä, kuten Angular.

Googlen kehittämän Angular-sovelluskehyksen myötä esitellään TypeScriptiä ja Angularin arkkitehtuuria, tärkeimpinä osina moduulit, komponentit ja palvelut. Palvelinpään tietolähteenä toimii avoimen datan palvelu nimeltään Lipas, joka on Jyväskylän yliopiston ylläpitämä kuntien liikuntapaikkojen tietoja sisältävä tietokanta.

Opinnäytetyössä toteutetaan Ionic- ja Angular-sovelluskehysten sekä Lipas-tietokannan tarjoaman liikuntapaikkatiedon avulla sovellusdemo mobiililaitteelle. Sovelluksen tarkoituksena on tarjota käyttäjälle palvelu, jonka kautta on mahdollista löytää tietoa paikkakuntien liikuntapaikoista. Valitusta kohteesta on mahdollista tarkastella yksityiskohtaisia tietoja sekä nähdä sen sijainti kartalla.

Asiasanat: Ionic, Angular, Android, mobiilisovellus, rajapinta

ABSTRACT

Oulu University of Applied Sciences
Degree Programme of Business Information Systems

Author(s): Mika Huotari
Title of thesis: Implementing a mobile application using Ionic Framework
Supervisor(s): Jouni Juntunen
Term and year when the thesis was submitted: Spring 2020 Number of pages: 40

The purpose of this thesis is to study the implementation of a mobile application using the Ionic application framework. The work reviews the history of Ionic, the most important features and application frameworks connected to the environment, such as Angular.

The Angular framework introduces TypeScript and Angular's architecture with modules, components and services as the most important components. The data source of the application is the open data service called Lipas maintained by the University of Jyväskylä.

With the help of the demo application, it is possible to search for municipal sports venues. The user can select a specific sport venue from which detailed information and location are displayed.

Keywords: Ionic, Angular, Android, mobile, application, interface

SISÄLLYS

1 JOHDANTO

Tämän työn tarkoitus on selvittää mobiilisovelluksen kehittäminen ideasta valmiiksi julkaistavaksi ohjelmistotuotteeksi. Opinnäytetyön tekijän aiempi mobiiliohjelmointikokemus rajoittuu lähinnä yhteen Oulun ammattikorkeakoululla käytyyn Android-ohjelmointiin keskittyneeseen Mobile programming -kurssiin. Kehitysympäristöksi valikoitui alustavan kartoituksen jälkeen Ionic-sovelluskehys, koska ympäristö tarjoaa monipuolisesti mahdollisuuksia mobiilisovelluksen kehittämiseen.

Motiivina työn tekemiselle on laajentaa ja syventää opinnäytetyön tekijän omaa ammatillista osaamista mobiiliympäristöjen parissa. Opinnäytetyön tekijällä on työkokemusta erilaisista ohjelmistoprojekteista yli kymmenen vuoden ajalta. Työhistoriaan on kuulunut erilaisia tehtäviä niin koodauksen, testauksen kuin suunnittelunkin saralla.

Henkilökohtaisena tavoitteena on parantaa ymmärrystä siitä, mitä asioita on otettava huomioon ja mitä vaiheita on käytävä läpi, kun mobiilisovellus kehitetään ideasta valmiiksi ohjelmistotuotteeksi. Opinnäytetyöprosessin myötä kehittyy jonkinlainen näkemys siitä, mitkä kehityskohteet on mahdollista suunnitella ennakkoon ja mitkä haasteet on vain pyrittävä ratkaisemaan sitä mukaa kuin kehitystyö etenee.

Koska opinnäytetyön tekijän mobiiliohjelmointikokemus on ennakkoon vähäinen, niin syntyvän ohjelmistotuotteen suhteen on hyvä asettaa realistiset vaatimukset. Ensimmäinen ohjelmistoversio tulee olemaan pelkistetty ja se tulee toimimaan lähinnä demona siitä, mitä mahdollisuuksia Ionic-sovelluskehys tarjoaa. Pitkällä tähtäimellä pyrkimys on, että tämän prosessin läpikäynti lisäisi opinnäytetyön tekijän omaa kiinnostusta tutkia aihealuetta jatkossa entistä syvällisemmin. Mahdollista on myös, että tämä projekti jää ainutkertaiseksi kokemukseksi Ionic-sovelluskehityksestä.

Opinnäytetyön kehitystä vie eteenpäin kolme kannustinta. Ensijaisena motivoijana toimii halu kehittää toimiva mobiilisovellus. Toinen mielenkiintoa herättävä kohde on sen toimintamallin havainnointi, miten valmis ohjelmistotuote toimitetaan asiakkaan ulottuville sovelluskauppaan. Kolmas kiinnostava asia on avoimen datan -palveluiden tarjonta ja niiden hyödyntäminen ohjelmistokehityksessä. Yleensä yksi tärkeimmistä ohjelmistotuotteen kehittämistä ohjaavista tekijöistä, kohderyhmäajattelu, jää tässä työssä vähemmälle huomiolle.

Jo aikaisessa vaiheessa oli selvää, että kehitystyö tullaan kohdistamaan Android-päätelaitteelle. Tätä päätöstä ohjasivat käytännölliset syyt, koska opinnäytetyön tekijällä oli käytettävissään Windows 10 -käyttöjärjestelmällä varustettu tietokone ja Android-mobiililaitteita. iOS-käyttöjärjestelmä karsiutui pois, koska sen kehitystyöhön tarvittavaa MacBook-tietokonetta ei ollut vaivattomasti saatavilla. Androidin valinnalle on nähtävissä myös selvä järkiperuste, koska se on selvästi käytetyin käyttöjärjestelmä mobiililaitteissa (Statcounter Global Stats 2020, viitattu 15.4.2020).

Kehitysympäristöä valittaessa vertailua käytiin lähinnä Googlen Android Studion ja avoimen lähdekoodin projektina syntyneen Ionic-sovelluskehyksen välillä. Android Studion avulla on mahdollista kehittää natiivisovellus, jolla saadaan parhaiten hyödynnettyä Android-käyttöjärjestelmällä varustetun mobiililaitteen ominaisuuksia. Ionic-sovelluskehyksellä taas voidaan kehittää monipuolisempi hybridisovellus, joka soveltuu käytettäväksi erilaisilla käyttöjärjestelmillä, kuten iOS ja Android, varustetuissa päätelaitteissa.

Vaikka tarkoitus oli toteuttaa sovellus Android-ympäristöön, suorituskykytekijöillä ei koettu demosovelluksen kannalta olevan juurikaan merkitystä. Tilastojen mukaan Ionic on yksi suosituimmista hybridisovellusten kehitysympäristöistä (Statista 2020, viitattu 23.6.2020). Halu tutustua monipuoliseen ja ohjelmistokehittäjien keskuudessa ilmeisen suosittuun kehitysympäristöön ratkaisi lopulta Ionic-sovelluskehyksen valinnan opinnäytetyön tutkimuskohteeksi. Ioniciin integroiduista sovelluskehyksistä valituksi tuli Angular, kun oletuksena oli, että sitä koskevaa lähdemateriaalia on paremmin saatavilla kuin uudemmille sovelluskehyksille Reactille ja Vuelle. Natiivin JavaScriptin käyttöä tässä projektissa ei ole harkittu.

Sovellusdemossa on tarkoitus keskittyä frontend-toteukseen, joten backend-toteutukseen tarvittava työn osuus on pyritty minimoimaan. Backend päätettiin toteuttaa avoimen datan rajapintoja hyödyntämällä. Avoindata.fi -sivuston avulla löytyi Jyväskylän yliopiston ylläpitämä liikuntapaikkoja koskeva sisältöpalvelu nimeltään Lipas.

Lipas REST API -rajapinnan avulla pyritään tuottamaan mobiilisovellus, joka näyttää rajatulle alueelle (esim. kunta tai maakunta) merkityt liikuntapaikat. Liikuntapaikkojen tiedot näkyisivät sovelluksessa sekä tekstinä että sijaintina kartalla. Sovelluksessa hyödynnetään avoimen yhteistyöprojektin OpenStreetMapin tuottamaa karttapalvelua.

Sovelluksen kehitystyö on tarkoitus toteuttaa kuin pieni ohjelmistoprojekti. Kun kehitysympäristö on asennettu ja sen käyttö opeteltu, toteutettava mobiilisovellus mallinnetaan muutaman käyttöliittymäkuvan avulla. Sovellusta testataan koko ajan kehitystyön ohessa. Kun ohjelmiston toiminnallisuudessa saavutetaan riittävä taso, pyritään mobiilisovellus julkaisemaan sovelluskaupassa. Projektiin liittyviä erilaisia tehtäviä hallitaan Trello-verkkopalvelun avulla.

2 IONIC-SOVELLUSKEHYS

Ionic-sovelluskehys (Ionic Framework) on avoimen lähdekoodin käyttöliittymäkirjasto, joka on tarkoitettu mobiili- ja työpöytäsovellusten sekä progressiivisten webapplikaatioiden (PWA) kehittämiseen. Ionicin alkuperäinen AngularJS:än päälle rakennettu versio julkaistiin v. 2013. Ensimmäinen beta-versio julkaistiin v. 2014. (JavaTpoint 2020, viitattu 12.5.2020.)

Ionic Framework -ekosysteemiä ohjaa kansainvälinen kehittäjäyhteisö ja avustajat, jotka tukevat sen kasvua ja helpottavat omaksumista. Ydinryhmä kehittää aktiivisesti järjestelmää ja ylläpitää sitä täysipäiväisesti. (Ionic 2020a, viitattu 11.3.2020.)

Yhteisöön kuuluu sovelluskehittäjiä yli 200 maasta ja sen käytössä on useita viestintäkanavia. Forumilla voidaan esittää kysymyksiä ja jakaa ideoita, Slackissa kehittäjät voivat tavata ja keskustella reaaliaikaisesti sekä Twitterissä on mahdollista jakaa erilaisia päivityksiä ja sisältöjä. GitHubia käytetään vikojen raportointiin ja uusien ominaisuuksien pyytämiseen ja Content authoring antaa mahdollisuuden teknisten blogien ja omien tarinoiden jakamiseen. (Ionic 2020a, viitattu 11.3.2020.)

2.1 Ionic on itsenäinen

Ionic-sovelluskehys on alustariippumaton eli sovellusten luominen ja käyttöönotto voidaan hoitaa yhdellä koodikannalla, niin iOS- ja Android -pohjaisissa natiivisovelluksissa kuin PWA-applikaatioissakin. Ionic on MIT-lisenssillä julkaistu avoimen lähdekoodin projekti. MIT-lisenssi tarkoittaa sitä, että tuotetta voidaan käyttää henkilökohtaisiin tai kaupallisiin projekteihin ilmaiseksi. Lisenssiä käyttävät myös sellaiset suositut projektit kuin JQuery ja Ruby on Rails. (Ionic 2020a, viitattu 11.3.2020.)

Ionic on rakennettu sellaisten luotettavien ja standardisoitujen webteknologioiden päälle kuin HTML, CSS ja JavaScript. Se käyttää Custom Elementsin ja Shadow DOM:n kaltaisia moderneja Web API -rajapintoja. Nämä ovat syitä, miksi Ionicilla on vakaa ohjelmointirajapinta eikä yksittäisten alustatoimittajien linjauksilla ole siihen isompaa vaikutusta. (Ionic 2020a, viitattu 11.3.2020.)

Aiemmin Ionic oli vahvasti sidoksissa Angular-sovelluskehykseen. Yksi Ionic 4 -version päätavoitteista oli poistaa komponenttien vahvat sidokset yksittäiseen ohjelmistokehykseen. Uudelleensuunnittelun tuloksena siitä tulikin itsenäinen Web Component -kirjasto, johon on integroitu uusimpia Angularin kaltaisia JavaScript-sovelluskehyksiä. (Ionic 2020a, viitattu 11.3.2020.)

Ionicia voidaan käyttää useimpien frontend-sovelluskehyksien, kuten React ja Vue, kanssa. Jotkin sovelluskehykset tarvitsevat kuitenkin lisäkirjaston pystyäkseen hyödyntämään Web Componentia. Itsenäiset ydinkomponentit voidaan ottaa käyttöön websivulla script-tagien avulla. Näin Ionicia on mahdollista käyttää itsenäisenä kirjastona esimerkiksi yksittäiseltä WordPress-sivulta. (Ionic 2020a, viitattu 11.3.2020.)

2.2 Sovelluslogiikan kehittäminen

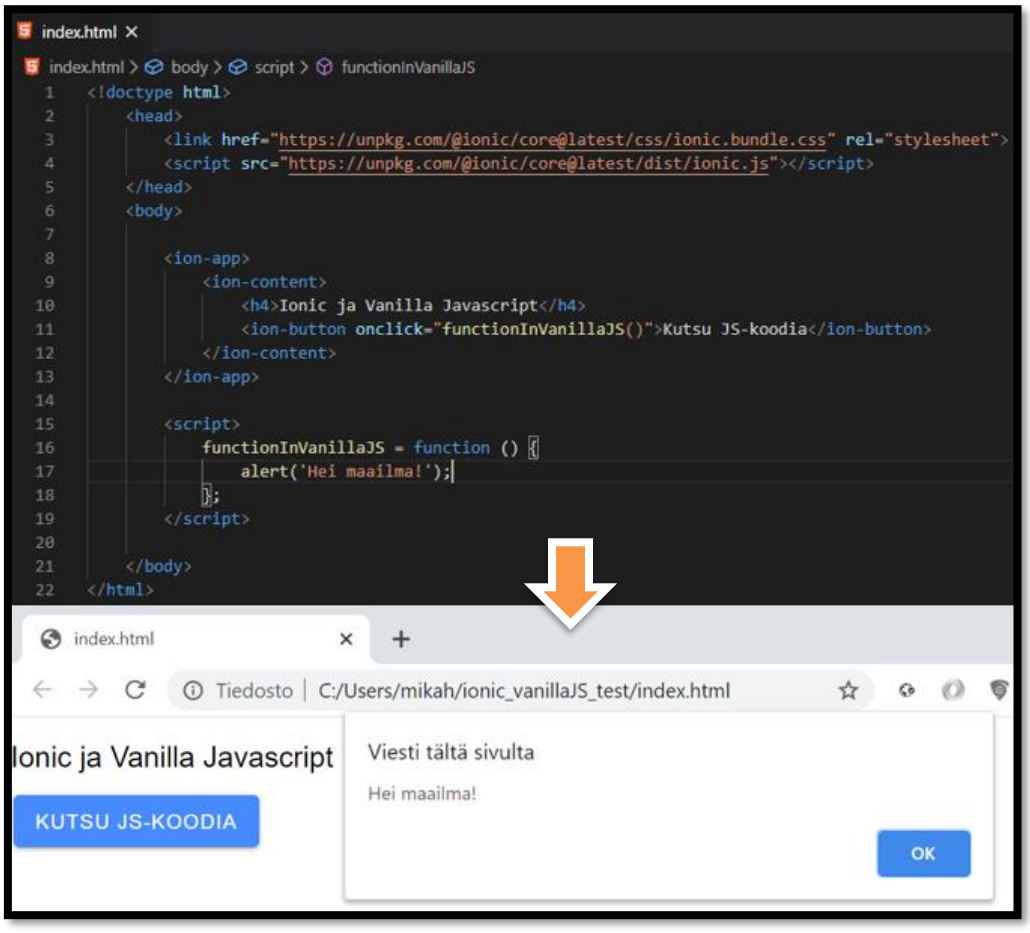
Ionic-sovellus voidaan kehittää käyttämällä natiivia JavaScript-kieltä. Tällöin sovelluslogiikka rakennetaan käyttämällä niin sanottua Vanilla JavaScriptiä (myös Vanilla JS), joka on pelkistettyä JavaScriptiä ilman lisäkirjastoja. Sovellusten kehitystyön helpottamiseksi ja vaihtoehtojen tarjoamiseksi Ioniciin on integroitu myös JavaScript-pohjaiset sovelluskehykset Vue, React ja Angular.

React on interaktiivisten käyttöliittymien tekemiseen tarkoitettu JavaScript-kirjasto. Reactin komponentit ovat lyhyitä ja toisistaan eristettyjä lähdekoodin palasia. Nämä kapseloidut komponentit kykenevät itsenäiseen tilanhallintaan ja tämä taas mahdollistaa monitahoisten käyttöliittymien luomisen. Reactin elementit ovat JavaScript-objekteja, joita on mahdollista tallentaa muuttujiin tai liikuttaa sovelluksen sisällä. (React 2020a, viitattu 13.5.2020; React 2020b, viitattu 13.5.2020.)

Vue on käyttöliittymien rakentamiseen tarkoitettu progressiivinen JavaScript-ohjelmistokehys. Kehyksen ydinkirjasto keskittyy vain näyttökerrokseen, joten se on mahdollista integroida muihin kirjastoihin tai olemassa oleviin projekteihin. Vue pystyy toimimaan myös yhden sivun sovelluksissa (Single-Page Application), kun niitä käytetään yhdessä nykyaikaisten työkalujen ja kirjastojen kanssa. (Vue.js 2020, viitattu 15.4.2020.)

Ionic-sovelluskehys tukee Angularia versiosta 6.0.0 ylöspäin. Angularissa on sisäänrakennettu työkalu helpottamaan päivitysten automatisointia ja antamaan palautetta sovelluskehittäjille rajapinnan muutoksista. Ionicista 4 eteenpäin on ollut käytössä sovellusten luomiseen ja reitityksen tarkoitettu virallinen Angular-pinomuisti, joten Ionic-sovellukset ovat yhdenmukaisia muun Angular-ekosysteemin kanssa. (Ionic 2020d, viitattu 16.4.2020.)

Kuva 1 näyttää yksinkertaisen koodin ja selainnäkymän kautta esimerkin siitä, miten natiivia JavaScriptiä voidaan käyttää Ionic-ympäristössä. Tiedoston index.html head-lohkossa esitellään kirjastot ionic.js sekä ionic.bundle.css ja body-lohkossa määritellään käytettävä JavaScript-koodi script-tagien välissä. Ionic-sovellus määritetään ion-app -lohkossa. Tästä lohkosta kutsutaan JavaScript-funktiota käyttäjän painettua ion-button -tyyppistä painiketta.

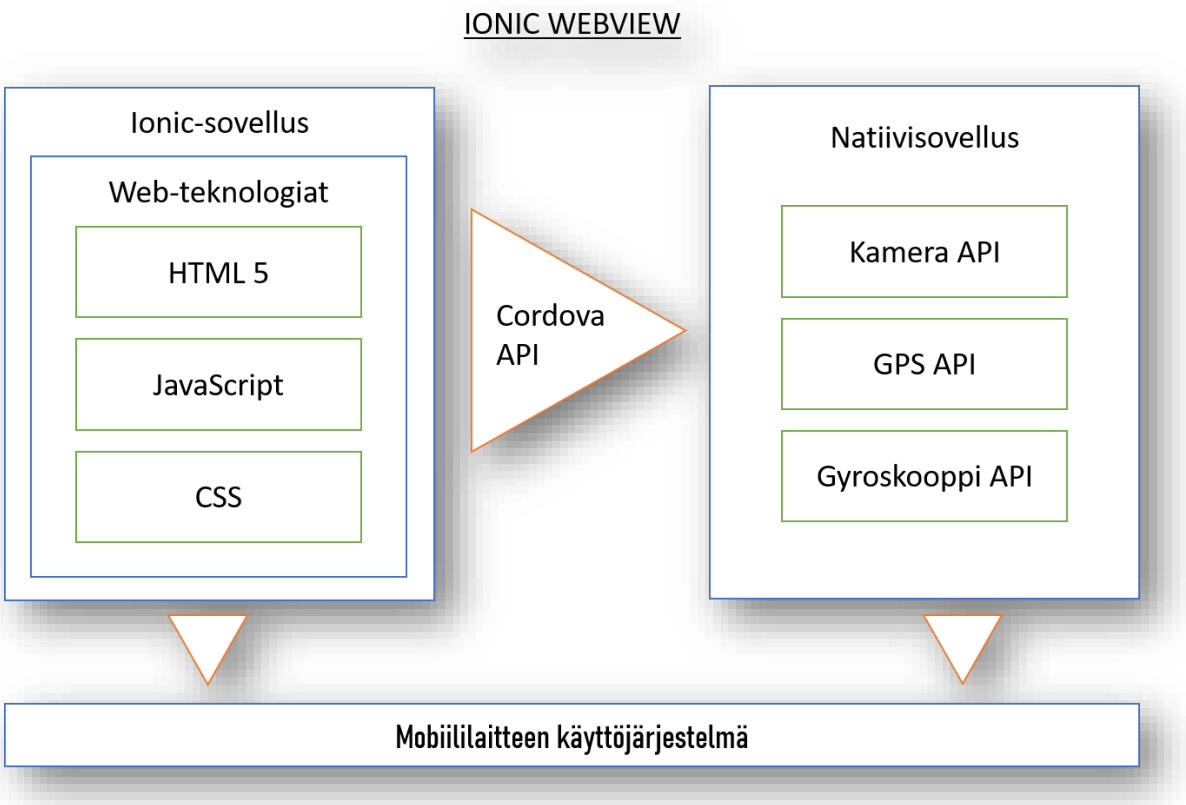


KUVA 1. Natiivin JavaScriptin käyttäminen Ionic-ympäristössä

2.3 Ionic WebView

WebView on sulautettu selain, jota natiivisovellus voi käyttää verkkosisällön esittämiseen. Natiivisovellus on kirjoitettu jollakin ohjelmointikielellä ja käyttöliittymä on varta vasten suunniteltu tietylle alustalle. Itsenäisen selainsovelluksen, kuten Google Chrome, osat ovat käyttöliittymä webelementteineen ja moottori, joka muuttaa koodit ihmisen näkemiksi pikseleiksi. (Kirupa 2020, viitattu 16.4.2020.)

Kuvassa 2 esitellään Ionic WebView:n toimintaperiaatetta. Ionic-sovellukset kehitetään webteknologioilla ja renderöidään näkymään käyttämällä WebView-komponentteja. Nykyaikaiset WebView-komponentit tarjoavat monia sisäänrakennettuja HTML5-rajapintoja laitteistoresursseille, kuten kamerat, sensorit, GPS, kaiuttimet ja Bluetooth. Ionic-sovelluksissa laitteiden rajapintoihin pääsee käsiksi siltakerroksen kautta. (Ionic 2020e, viitattu 16.4.2020.)



KUVA 2. Ionic WebView (Ionic 2020e, viitattu 16.4.2020.)

Cordova toimii rajapintana Ionic-sovelluksen ja natiivisovelluksen välillä. Ionic-sovelluksen JavaScript-metodeilla on yhteys laitteiston natiivikoodin metodeihin Cordova-sillan ansiosta. Silta mahdollistaa parametrien lähettämisen edestakaisin. Cordova sisältää tavallisesti natiivikoodin tuetuille alustoille, yhteisen JavaScript-rajapinnan ja plugin.xml-nimisen manifest-tiedoston. (Mawson, G. 2015, viitattu 16.4.2020.)

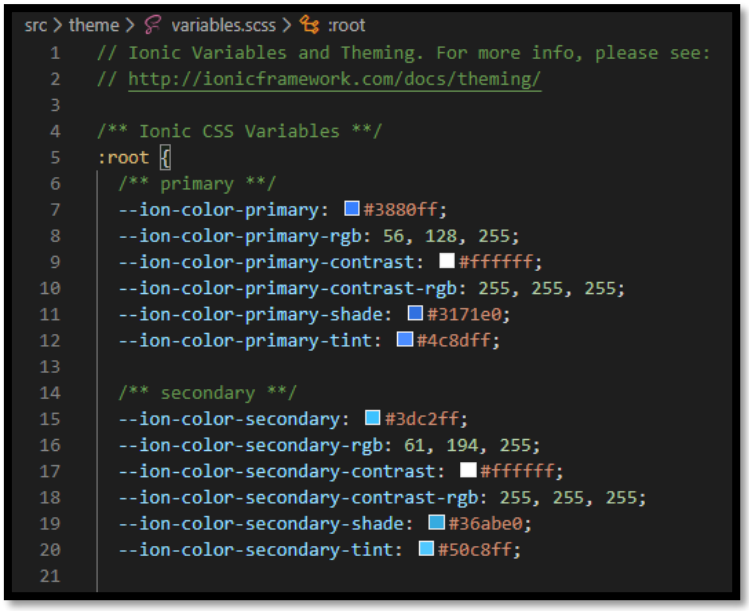
2.4 Sovelluksen ulkoasu

Ionicista löytyy Layout -dokumentaation (Ionic 2020g, viitattu 13.5.2020) mukaan elementtien asettelua helpottavia layout-pohjia erilaisiin tarpeisiin yksisivuisista asetteluista jaetun ruudun näkymiin ja modaaleihin. Seuraava luetelma listaa muutamia esimerkkejä Ionic-sovelluksen asetteluista:

- Header and Footer Layout; yksinkertaisin asettelu koostuu otsikosta ja sisällöstä (content)
- Tabs Layout; vaakatasossa olevista välilehdistä koostuva asettelu
- Menu Layout; sivuvalikko avautuu painikkeesta tai pyyhkäisemällä
- Split Pane Layout; sallii useiden näkymien näyttämisen

Ionic-sovelluskehys on kokoelma käyttöliittymäkomponentteja, jotka ovat sovelluksen rakennuspalikoina toimivia uudelleenkäytettäviä elementtejä. Komponentit ovat mukautuvia, joten niillä on mahdollista vaikuttaa koko sovelluksen ulkoasuun. Ionic on kehitetty Cascading Style Sheets -tyyliohjeiden (CSS) mukaisesti ja tämä mahdollistaa CSS-ominaisuuksien joustavuuden hyödyntämisen kehitystyössä. (Ionic 2020b, viitattu 11.3.2020.)

Ionicin CSS-tiedostoista löytyy oletustyyleinä erilaisia väriyhdistelmiä (kuva 3), mutta sovelluskehittäjillä on myös mahdollisuus korvata oletusvärit luomalla omia, esimerkiksi yrityksen brändiin yhteensopivia, värimalleja. Koko sovelluksen kattavat värit voidaan vaihtaa muokkaamalla tyylejä CSS-tiedostossa tai säätämällä komponentin color-ominaisuutta HMTL-tiedostossa (esim. <ion-toolbar color="secondary">). (Ionic 2020f, viitattu 13.5.2020.)



KUVA 3. Esimerkki Ionic Colorin esiasetetuista väriyhdistelmistä (Ionic 2020f, viitattu 13.5.2020)

Mukautuva muotoilu (Adaptive Styling) on Ionic-sovelluskehyksen sisäänrakennettu ominaisuus, joka sallii sovelluskehittäjän käyttää samaa koodikantaa useille alustoille. Jokainen Ionic-komponentti mukauttaa ulkoasunsa sille alustalle, millä sovellus pyörii. Esimerkiksi Apple-laitteet, kuten iPhone ja iPad, käyttävät Applen omaa iOS-suunnittelukieltä. Samalla tavalla Android-laitteet käyttävät Googlen kehittämää muotokieltä Material Designia. (Ionic 2020b, viitattu 11.3.2020.)

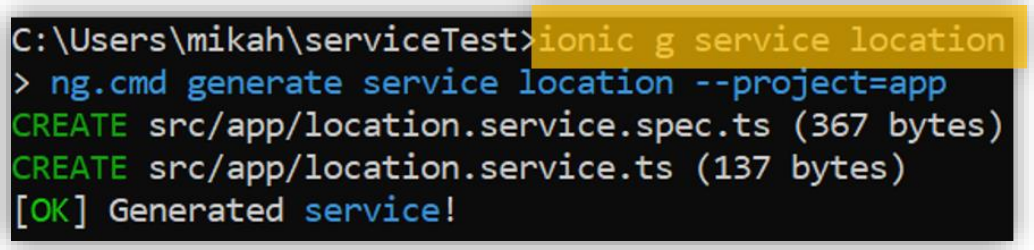
2.5 Testaaminen Ionic-ympäristössä

Kun Ionic/Angular -sovellus luodaan käyttämällä Ionicin komentorivikäyttöliittymää (command-line interface, CLI), sille otetaan automaattisesti käyttöön testausmenetelmiä, kuten yksikkötestaus (unit testing) ja päästä päähän -testaus (end-to-end testing). Testausympäristön kokoonpano on samanlainen kuin mitä käytetään Angularin CLI-työkalussa. (Ionic 2020h, viitattu 16.5.2020.)

Yksikkötesteissä testataan yhtä osaa sovelluksen koodista, kuten komponentti, sivu, palvelu tai putki, erillään muusta järjestelmästä. Testattavan osan pitäminen erossa muusta koodista saadaan aikaan injektoimalla todellisuutta jäljitteleviä mock-objekteja oikeiden riippuvuuksien sijaan. Mock-objektien avulla on mahdollista hallita riippuvuuksien tuotoksia yksityiskohtaisesti. (Ionic 2020h, viitattu 16.5.2020.)

Sivut (page) ovat Angular-komponentteja, joten sekä sivut että komponentit testataan käyttämällä Angularin Component Testing -ohjeita. Koska sivuilla ja komponenteissa käytetään sekä TypeScript-koodia että HTML-merkkikieltä, on komponentin luokan ja dokumenttioliomallin (DOM) testaaminen mahdollista. (Ionic 2020h, viitattu 16.5.2020.)

Palvelut on pyrittävä testaamaan niin, että palvelusta luodaan instanssi. Uudelle instanssille injektoidaan mock-objekteja sen verran mitä palvelulla on riippuvuuksia. Näin palvelua voidaan testata sellaisenaan. Muiden palveluiden toimintalogiikka tai niihin tulevat muutokset eivät siis vaikuta testattavan palvelun toimintaan, kunhan vain tarvittavat arvot ovat asianmukaisia. Testit ovat automaattisesti käytettävissä, kun palvelu on luotu Ionic CLI:n kautta kuvassa 4 esitetyllä tavalla. (Ionic 2020h, viitattu 16.5.2020.)



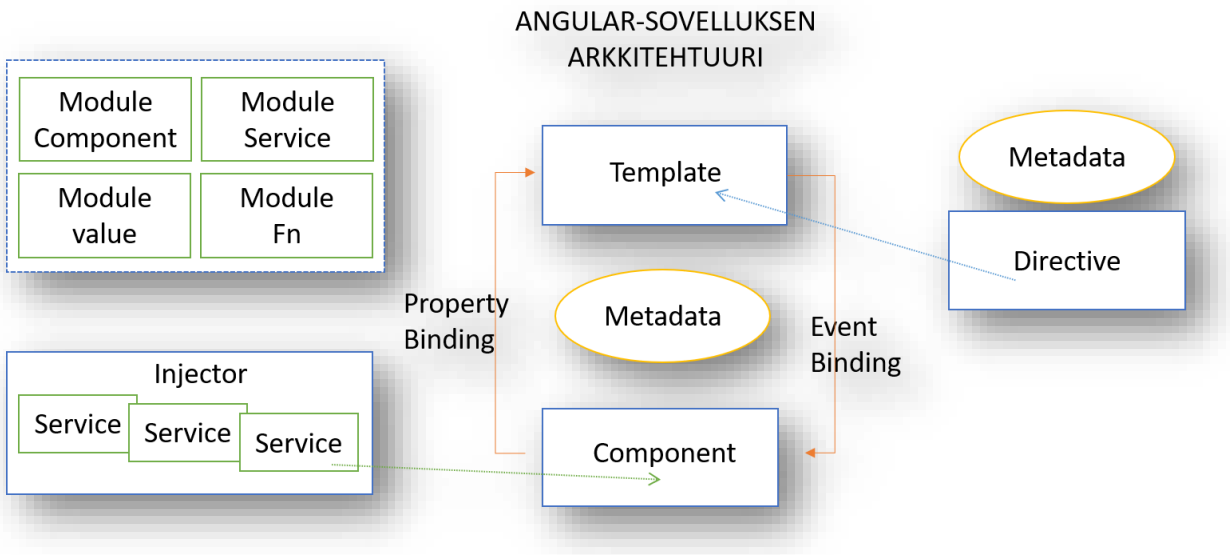
KUVA 4. *Palvelun luominen Ionic CLI:n avulla*

3 ANGULAR

Angular on Googlen kehittämä TypeScript-pohjainen JavaScript-sovelluskehys ja se julkaistiin ensimmäisen kerran vuonna 2010. Sovelluskehykseen tuli merkittävä muutos vuonna 2016 Angular 2 -version julkaisun myötä. Viimeisin vakaa versio on Angular 9, joka julkaistiin helmikuussa 2020. (Daityari 2019, viitattu 9.5.2020.)

Alun perin AngularJS:än uudelleen kirjoitetusta versiosta käytettiin nimeä Angular 2. Epäselvyyksien välttämiseksi päätettiin kuitenkin, että alkuperäisistä sovelluskehitysversioista käytetään termiä AngularJS ja versiosta 2 eteenpäin käytetään pelkistettyä termiä Angular. (Daityari 2019, viitattu 9.5.2020.)

Angular-sovelluksen arkkitehtuuri (kuva 5) tukeutuu tiettyihin peruskäsitteisiin. Sovelluksen rakennuspalikoina toimivat Angularin NgModuulit (NgModules), jotka keräävät toisiinsa liittyvät koodit toiminnallisiksi joukoiksi. NgModuulit tarjoavat komponenteille (Component) käännösympäristön. (Angular 2020b, viitattu 16.4.2020.)



KUVA 5. Angular-sovelluksen arkkitehtuuri (Angular 2020b, viitattu 16.4.2020)

3.1 Moduulit

Jokaisessa Angular-sovelluksessa on oltava vähintään juurimoduuli (root), tavanomaisesti nimeltään AppModule, joka mahdollistaa sovelluksen käynnistämiseen. Sovellus sisältää tyypillisesti myös monia toiminnallisia moduuleja. (Angular 2020b, viitattu 16.4.2020.)

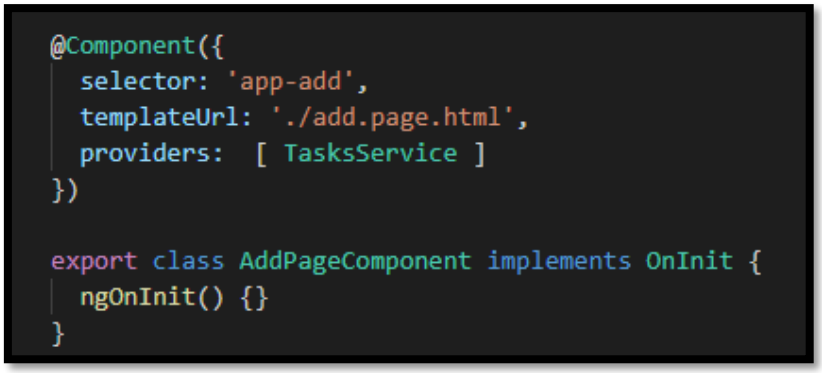
NgModuulien avulla täydennetään JavaScriptin ES2015-määrittelyn mukaisia moduuleja. NgModuuli määrittää käännöskontekstin komponenttijoukolle, joka voi olla varattu sovelluksen toimialueen, työnkulun tai ominaisuuksien käyttöön. Se voi yhdistää komponentit asianmukaiseen koodiin, kuten palveluihin, toiminnallisten yksiköiden muodostamiseksi. (Angular 2020b, viitattu 16.4.2020.)

Kuten JavaScript-moduuleilla yleensäkin, NgModuulilla on mahdollisuus tuoda toiminnallisuutta toisista NgModuuleista. Se myös sallii toisten NgModuulien hyödyntävän omaa toiminnallisuuttaan. Koodin järjestäminen erillisiksi toiminnallisiksi moduuleiksi helpottaa monimutkaisten sovellusten kehittämistä ja uudelleenkäytettävyyden suunnittelua. Tämän tekniikan avulla voidaan myös hyödyntää Lazy-loading -ominaisuutta eli moduuleja ladataan vain tarpeen mukaan. (Angular 2020b, viitattu 16.4.2020.)

3.2 Komponentit

Jokaisessa Angular-applikaatiossa on vähintään yksi komponentti. Juurikomponentti yhdistää komponenttihierarkian sivun DOM-malliin. Jokainen komponentti määrittelee luokan, joka sisältää sovellustiedot ja -logiikan. Siihen liittyy myös HTML-malli (HTML template), joka määrittelee kohdeympäristössä näytettävän näkymän. @Component-dekoraattori tunnistaa seuraavana alapuolellaan olevan luokan komponentiksi tarjoten sille mallin ja komponenttikohtaisen metadatan käytettäväksi. (Angular 2020b, viitattu 16.4.2020.)

Kuvassa 6 esitetyistä @Component-dekoraattorin asetuksista selector on CSS-valitsin, joka määrittää HTML-tagit, joiden väliin Angular lisää komponentin instanssin. Asetuksista templateUrl on komponentin HTML-mallin osoite ja providers on taulukko komponentin vaatimista palvelujen tuottajista. (Angular 2020c, viitattu 10.5.2020.)



KUVA 6. Esimerkki komponentin metadatasta

Malli (template) yhdistää HTML:ään Angularin merkinnän (markup), joka voi muokata HTML-elementtejä ennen niiden näyttämistä. Mallin direktiivit (directive) tarjoavat sovelluslogiikan ja tiedon sidonnan merkintä (HTML-mallissa: {{muuttuja}}) yhdistää sovellusdatan ja DOM-mallin. (Angular 2020b, viitattu 16.4.2020.)

Tietojen sidontaa (data binding) on kahta tyyppiä: tapahtumien sidontaa (event binding) ja ominaisuuksien sidontaa (property binding). Tapahtumien sidonta mahdollistaa sovelluksen reagoinnin DOM-mallista tulevaan käyttäjän antamaan syötteeseen päivittämällä sovellusdataa. Ominaisuuksien sidonta taas antaa DOM-mallille mahdollisuuden havaita sovellusdatassa tapahtuvat muutokset. (Angular 2020b, viitattu 16.4.2020.)

Direktiivi (directive) on luokka, jolla on @Directive-dekoraattori. Kun Angular renderöi mallejaan, ne muuttuvat dynaamisesti DOM-direktiivien ohjeistuksen mukaisesti. Direktiivejä on olemassa kolmea tyyppiä: direktiivejä mallilla laajentava komponentti, DOM-elementtien muokkauksen avulla layoutia muuttava rakennedirektiivi sekä HTML-elementin ulkoasua tai käyttäytymistä muuttava ominaisuusdirektiivi. Taulukko 1 esittää esimerkkien avulla muutamia rakenne- ja ominaisuusdirektiivejä. (Angular 2020c, viitattu 10.5.2020.)

*TAULUKKO 1. Esimerkkejä Angularin direktiiveistä (*Angular 2020d, viitattu 11.5.2020; Angular 2020e, viitattu 11.5.2020*)*

Direktiivi	Tyyppi	Kuvaus	HTML-esimerkki
NgClass	ominaisuus	Lisää ja poistaa CSS-luokkia HTML-elementissä.	<div [ngClass]=" isSpecial ? 'special' : '' "> This div is special
NgFor	rakenne	Toistaa solmun (node) jokaiselle listan kohteelle.	<div *ngFor="let item of item"> {{item.name}}
NgIf	rakenne	Ehdollisesti joko luo tai tuhoaa alinäkymiä mallista.	<app-item-detail *ngIf="isActive" [item]="item"> </app-item-detail>
NgModel	ominaisuus	Lisää kaksisuuntaisen tiedon sidonnan HTML-muotoelementtiin (form).	<label for="example ">[(ngModel)]:</label> <input [(ngModel)]="name" id="example">
NgStyle	ominaisuus	Lisää ja poista HTML-tyylejä.	<div [ngStyle]="{'font-size':'12px'}"> </
NgSwitch	rakenne	Näkymien välillä vaihtava direktiivijoukko.	 Pyörä Auto <div ng-switch-default>Kävely

3.3 Palvelut

Palvelu (Service) on laaja luokka, joka kattaa kaikki sovelluksen tarvitsemat arvot, toiminnot tai ominaisuudet. Palvelu on tyypillisesti luokka, jolla on kapea ja tarkkarajainen tarkoitus. Sen on tarkoitus tehdä hyvin jokin erityistehtävä. (Angular 2020a, viitattu 8.5.2020.)

Angular erottaa komponentit palveluista modulaarisuuden ja uudelleenkäytettävyyden lisäämiseksi. Kun erotetaan komponentin web-näkymään liittyvä toiminnallisuus muusta toiminnallisuudesta, komponenttiluokista voidaan tehdä kevyitä, joustavia ja tehokkaita. (Angular 2020a, viitattu 8.5.2020.)

Riippuvuusinjektio (Dependency injection) on kytketty Angular-sovelluskehykseen ja sitä käytetään tarjoamaan uusia komponentteja palveluilla. Komponentit kuluttavat palveluita. Kun palvelu injektoidaan komponenttiin, komponentti pääsee käyttämään kyseistä palveluluokkaa. (Angular 2020a, viitattu 8.5.2020.)

Jokaista riippuvuutta kohden on rekisteröitävä tarjoaja (Provider) sovelluksen injektorissa, jotta injektori voi käyttää tarjoajaa luomaan uusia ilmentymiä. Palvelun tarjoaja on tyypillisesti itse palveluluokka. (Angular 2020a, viitattu 8.5.2020.)

Angularin reititys (Router) on palvelu, jolla voidaan määrittää navigointipolku eri sovellustilojen välillä ja tarkastella hierarkioita sovelluksessa. Reititys tarjoaa selaimista tutun tavan navigoida: URL-osoitteen syöttäminen osoitekenttään ja hyperlinkin painallus ohjaavat käyttäjän oikealle sivulle. Selaimen taaksepäin- ja eteenpäin -painikkeet ohjaavat käyttäjän sivulta toiselle selaushistorian perusteella. (Angular 2020b, viitattu 16.4.2020.)

3.4 TypeScript on JavaScriptin yliluokka

Alun perin Microsoftin kehittämä TypeScript ei ole oma kielensä, vaan se lisää JavaScriptiin uusia ominaisuuksia, kuten vahvan tyypityksen tuen. Vahvasti tyypitetyissä kielissä jokaisen muuttujan tyyppi on määriteltävä sitä luotaessa. JavaScript ja sen päälle kehitetty TypeScript perustuu ECMAScript on standardiin. (Tarvainen, J. 2016, viitattu 8.4.2020.)

ECMAScript (ES) on skriptikielen määrittely, jonka on standardoinut Ecma International. Se luotiin JavaScriptin standardisoimiseksi, jotta itsenäisten toteutusten tekeminen helpottuisi. JavaScript on edelleen ECMAScriptin laajimmin käytetty toteutus standardin julkaisemisen jälkeen. (freeCodeCamp 2020, viitattu 13.5.2020.)

TypeScript-kieli perustuu syntaksiin ja semantiikkaan, joka on tuttu miljoonille JavaScriptin käyttöön tottuneille ohjelmistokehittäjille. Olemassa olevia koodeja ja valmiita JavaScript-kirjastoja voidaan yhä käyttää, kun TypeScript-koodia hyödynnetään kutsumalla sitä omasta JavaScript-koodista. TypeScript-ympäristö kääntää koodin pelkistetyksi JavaScript-koodiksi, joka toimii kaikilla selaimilla, Node.js-ympäristöissä tai missä tahansa JavaScript-moottorissa, josta tukee vähintään versiota 3 ECMAScriptistä. (Microsoft 2020, viitattu 8.4.2020.)

4 SOVELLUKSEN TOTEUTUS

Ionic tarjoaa useita mahdollisuuksia sovelluslogiikan toteuttamiseen mobiilisovelluksessa. JavaScript-koodin tekemiseen tässä projektissa valittu Angular-sovelluskehys oli järkiperusteinen ja laskelmoitu valinta. Se on ollut pitempään käytössä kuin React ja Vue, joten opinnäytetyön tekemistä helpottavaa dokumentaatiota on todennäköisesti löydettävissä enemmän. Angular on yhä suosittu ja laajasti käytetty sovelluskehys (gitconnected 2020, viitattu 18.5.2020).

Vaikka Ionic on hybridisovelluskehys, mobiilisovellusdemoa päätettiin kehittää pääasiallisesti vain Android-käyttöjärjestelmään keskittyen. Android on opinnäytetyön tekijälle tutumpi käyttöjärjestelmä kuin esimerkiksi iOS. Android-kehitystyötä puolsi myös se, että testausta varten oli käytettävissä useampia Android-päätelaitteita ja ohjelmistokehitystyöhön oli käytettävissä Windows 10-käyttöjärjestelmällä varustettu tietokone. iOS-kehitystyö olisi vaatinut macOS-käyttöjärjestelmällä varustetun tietokoneen, kuten Applen Macbook Air.

Mobiilisovelluksen idea oli jalostaa Jyväskylän yliopiston Lipas-tietokannasta löytyvää avointa liikuntapaikkatietoa käyttäjäystävälliseen muotoon. Sovelluksen avulla on mahdollista etsiä eri paikkakuntien liikuntapaikkoja. Käyttäjä voi halutessaan tutustua tarkemmin tietyn liikuntapaikan yksityiskohtaisiin tietoihin sekä sijaintiin kartalla.

4.1 Avoin data tietolähteenä

Kun päätös avoimen datan käyttämisestä sovelluksen tietolähteenä oli tehty, niin sen jälkeen palvelujen kartoitusta tehtiin pääsääntöisesti Avoindata.fi -sivuston avulla. Portaalin kautta löytyy runsaasti erilaisia aineistoja muun muassa terveyteen, ympäristöön, matkailuun, talouteen ja väestöön liittyen.

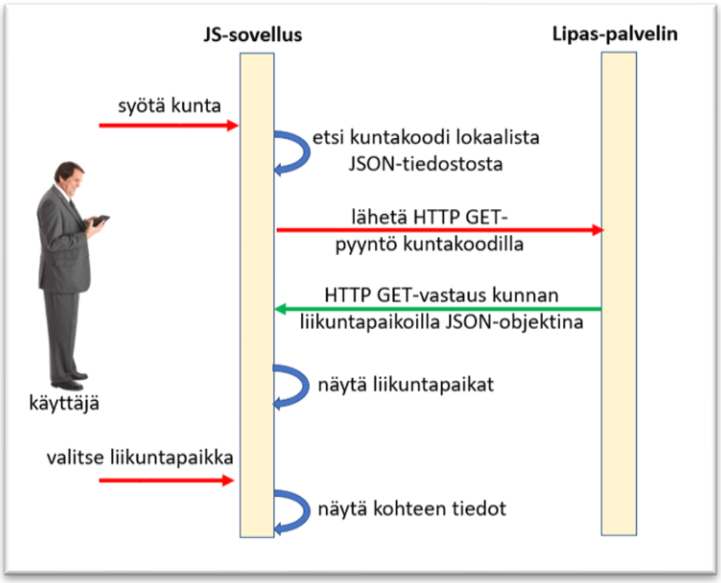
Mobiilisovelluksen sisältöön vaikutti alun perin kaksi kriteeriä: aiheen olisi kiinnostettava opinnäytetyön tekijää ja aineiston tulisi sisältää tietoa Kainuun alueesta. Lopulta Jyväskylän yliopiston ylläpitämästä Lipas-tietokannasta löytyi molemmat valintakriteerit täyttävä aineisto. Lipas sisältää

kunnallisten toimijoiden päivittämää tietoa omien paikkakuntiensa liikuntapaikoista. Tietokannasta löytyy tietoa muun muassa liikuntapaikkojen sijainneista, pinta-aloista ja yhteystiedoista.

Liikuntapaikka-aineistosta oli alkuperäisen suunnitelman mukaan tarkoitus rajata käytettäväksi pienempi maantieteellinen alue, kuten yksi kaupunki tai yhden maakunnan paikkakunnat. Toteutuksen edetessä kävi kuitenkin selväksi, ettei sovelluksen toimintaa ole järkevää rajoittaa vain tiettyä aluetta koskevaksi. Sovellus pystyisi sellaisenaan esittämään liikuntapaikat myös koko maan kattavasti, joten alueellinen rajaaminen olisi teettänyt vain lisätöitä.

Yksi väylä, jolla Lipas-palvelimelta on mahdollista hakea tietoa, on REST API -rajapinta. Tätä rajapintaa hyödynnettiin sovelluksessa niin, että ensin käyttäjä syöttää hakukenttään jonkin suomalaisen kunnan nimen. Sovellus hakee paikallisesta citycodes.json -tiedostosta annettua nimeä vastaavan yksilöllisen kuntakoodin.

Sen jälkeen Lipas-palvelimelle lähetään HTTP GET -viesti, jolla pyritään saamaan Lipas-tietokannasta kaikki paikkakuntaa koskeva liikuntapaikka-aineisto. Lipas-palvelin lähettää vastauksena pyyntöön JSON-muotoisen aineiston. Saadusta aineistosta sovellus listaa pääsivulleen liikuntapaikkojen nimet ja tyypit lajiteltuna nimen mukaiseen aakkosjärjestykseen. Tästä listasta käyttäjä voi valita itseään kiinnostavan liikuntapaikan ja valitusta liikuntapaikasta näytetään yksityiskohdat Kohteen tiedot -sivulla. Sovelluksen toimintalogiikkaa mallintaa kuva 7.



KUVA 7. *Sovelluksen käyttäjä-sovellus-palvelin -vuorovaikutus*

4.2 Työkalut

Helmikuussa vuonna 2020, opinnäytetyöprosessin alkuvaiheessa, Ionicin pääversio (major version) päivittyi versiosta 4 pykälän ylöspäin, kun muun muassa Angular 9 -tuen sisältävä 5.0.0 Magnesium -versio julkaistiin (Ionic 2020i, viitattu 17.5.2020). Liikuntapaikka-sovellus käyttää toukokuussa ilmestynyttä Ionic Framework 5.1.1-versiota (kuva 8).

```
C:\Users\mikah\git_local\SportPlaces>ionic info

Ionic:

   Ionic CLI                     : 6.7.0 (C:\Users\mikah\AppData\Roaming\npm\node_modules\@ionic\cli)
   Ionic Framework               : @ionic/angular 5.1.1
   @angular-devkit/build-angular : 0.803.26
   @angular-devkit/schematics    : 8.3.26
   @angular/cli                  : 8.3.26
   @ionic/angular-toolkit        : 2.2.0

Cordova:

   Cordova CLI       : 9.0.0 (cordova-lib@9.0.1)
   Cordova Platforms : android 8.1.0
   Cordova Plugins   : cordova-plugin-ionic-keyboard 2.2.0, cordova-plugin-ionic-webview 4.2.1, (and 5 other plugins)

Utility:

   cordova-res                          : not installed
   native-run (update available: 1.0.0) : 0.3.0

System:

   Android SDK Tools : 26.1.1 (C:\Users\mikah\AppData\Local\Android\Sdk)
   NodeJS            : v12.14.1 (C:\Program Files\nodejs\node.exe)
   npm               : 6.13.4
   OS                : Windows 10
```

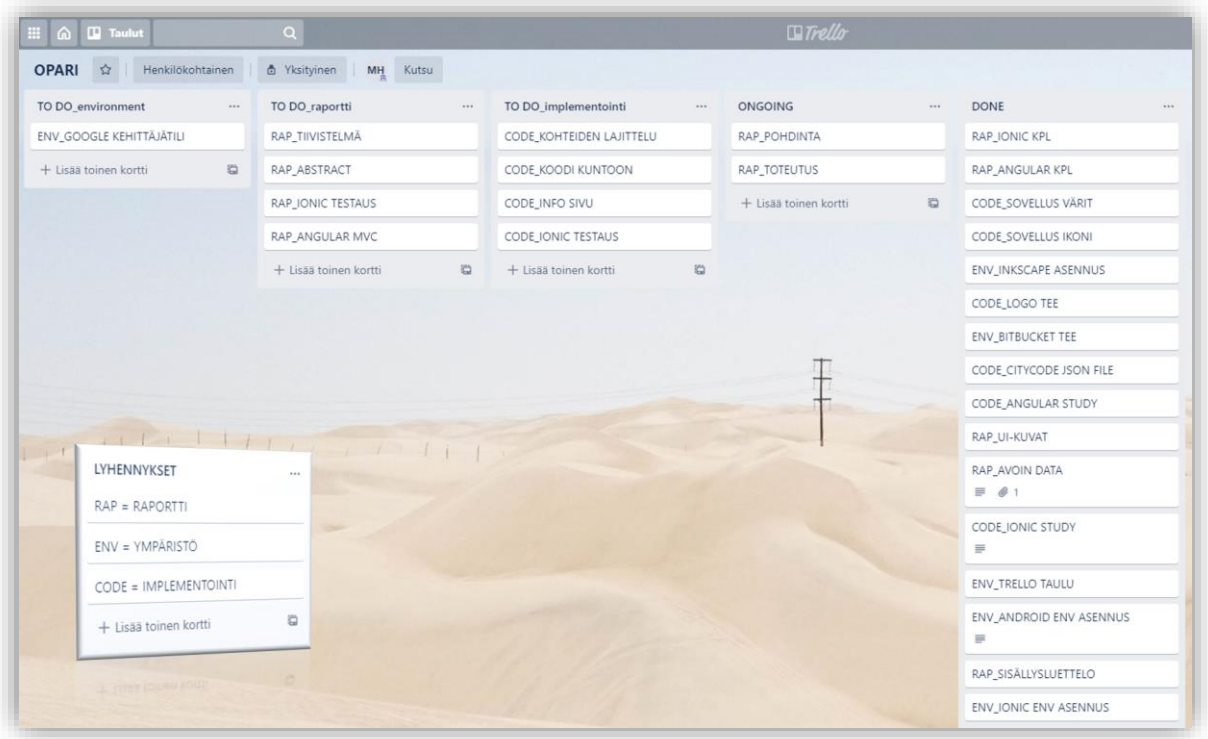
KUVA 8. Liikuntapaikka-sovelluksen ohjelmistoversiot

Lähdekoodin versionhallintaan käytettiin Git-järjestelmää, joka valittiin RhodeCode Enterprisen (2020) mukaan suosituimmaksi versionhallintajärjestelmäksi v. 2016. Liikuntapaikka-sovelluksen versionhallintalustaksi valikoitui Bitbucket. Se on monipuolinen koodinhallinnan ja tiimityöskentelyn työkalu, mutta tässä projektissa sen käyttö rajoittui lähinnä koodin tallennuspaikaksi.

Projektin tehtävienhallintaan käytettiin selainpohjaista Trello-järjestelmää. Trello on yhteistyökalu, joka soveltuu projektin organisointiin taulujen avulla. Taululla olevat tehtävät ovat ikään kuin muistilappuja, joihin voidaan liittää dokumentteja ja joita projektin jäsenten on mahdollista täydentää omilla kommenteillaan. Projektitauluja on mahdollista käyttää sijainnista riippumatta älypuhelimella ja tietokoneilla verkkoyhteyden kautta. (Trello 2020, viitattu 8.4.2020.)

Opinnäytetyötä varten perustettiin oma Trello-taulu, johon ryhmiteltiin tehtävälistat osa-alueittain, kuten ohjelmointiympäristön konfiguroimiseen, raportin laatimiseen ja sovelluksen ohjelmointityöhön liittyvät tehtävät. Tehtävälistat taas palasteltiin tehtäviin korttien avulla. Kun jotakin tehtävää

alettiin suorittamaan, kortti siirrettiin Ongoing-listalle. Tehtävän valmistuttua kortti siirrettiin Done-listan ylimmäiseksi (kuva 9).



KUVA 9. Opinnäytetyön projektinhallinta Trellossa

Työn edetessä kävi ilmi, että Bitbucket ja Trello ovat molemmat Atlassian-organisaation hallinnassa. Bitbucketista löytyikin mahdollisuus integroida tiedon tallennuspaikka (repository) Trello-taulun kanssa. Trello-lisäosan käyttö Bitbucketissa helpotti työn tekemistä jonkin verran, kun sekä version- että projektinhallinta löytyivät samasta verkkosijainnista.

Ohjelmointityön tukena käytettiin avoimen lähdekoodin Visual Studio Code -editoria (VS Code), joka on selkeä, helppokäyttöinen ja kevyt eli tietokoneen resursseja vähän kuluttava. VS Code toimii hyvin nykyaikaisella tietokoneella useiden sovellusinstanssien ollessa auki, joka taas mahdollistaa lähdekoodin tutkimisen useasta sovellusprojektista samanaikaisesti. VS Code tukee eri tekniikoiden, kuten JavaScript, JSON sekä Git, käyttöä ja siihen on saatavana runsaasti erilaisia lisäosia.

Mobiilisovelluksen logo (kuva 10) tehtiin Microsoftin PowerPoint-ohjelmalla. PowerPointilla toteutettiin myös raportin kuvien muokkaus ja yhdistely. Se on varsin kätevä työkalu grafiikan tekemiseen, koska valmiita elementtejä on saatavissa melko runsaasti, objektien asettelu ja yhdistely on suhteellisen vapaata ja objektit on helppo tallentaa, tässä tapauksessa png-muotoisina, kuvina.

KUVA 10. Logon kuva ja sijainti Liikuntapaikka-sovelluksessa

Sovellusikoni (käynnistysikoni, launcher icon) oli alun perin tarkoitus tehdä pienellä vaivalla ja nopeasti vain kopioimalla sovelluslogosta pieni osa. Tämä ratkaisu ei kuitenkaan toiminut puhelimen työpöydällä, jossa ikonin leveys ja korkeus on mobiilaitteista riippuen noin senttimetri. Oli siis tutkittava tarkemmin muiden sovelluskehittäjien ratkaisuja sen suhteen, miten pienessä tilassa saadaan olennainen informaatio esitettyä. Kevyen analyysin perusteella ratkaisu näyttää yleisesti olevan se, että ikoni luodaan yksinkertaisen kuvan, brändin mukaisen värityksen ja korkeintaan 1-5 merkkiä pitkän tekstin avulla.

Liikuntapaikka-sovelluksen png-muotoinen sovellusikoni (kuva 11) toteutettiin CorelDRAW 2020 -ohjelman avulla yhdistelemällä muotoiluelementtejä ja tekstiä. Värit ovat samat kuin logossa eli valkoinen, oranssi (#C55A11) ja sininen (#2E75B6). Ikonissa pyrittiin välittämään tekstin avulla liikkumiseen liittyvää viestiä ja pallon/pisteen avulla yritettiin luoda mielikuvaa yhtäältä pallopelistä sekä toisaalta paikkaa kuvaavasta pisteestä kartalla.

KUVA 11. Sovellusikonin kuvatiedosto ja sijainti mobiililaitteen työpöydällä

Oman sovellusikonin saa otettua käyttöön Ionic-ympäristössä korvaamalla sillä resources-kansion alta oletuksena löytyvä icon.png -tiedoston. Komentoriviltä ajettu komento *ionic cordova resources* saa Cordovan generoimaan automaattisesti eri kokoisia kuvia niin Androidin kuin iOS:in käyttöön, kun lähdetiedosto oli validi.

Vähimmäisvaatimus sovellusikonin koolle on 1024x1024 pikseliä. Splash screen -kuvan eli sovelluksen käynnistyksen yhteydessä näytettävän kuvan vähimmäiskoko on 2732x2732 pikseliä ja sille annettava nimi splash.png. Validointi on ainakin koon suhteen tarkka, sillä muutaman pikselin liian pieneksi jäänyt kuvatiedosto aiheutti generointivaiheessa ValidationError-virheen.

Kuvassa 12 esitetyt käyttöliittymäkuvat toteutettiin WireframeSketcher-sovelluksella. Ohjelmalla on mahdollista luonnostella (sketch) käsintehdyn oloisia rautalankamalleja (wireframe) sovelluksen käyttöliittymistä. Rautalankamallilla pyritään mallintamaan optimaalista elementtien määrää ja sijaintia suhteessa laitteiston näytön kokoon.

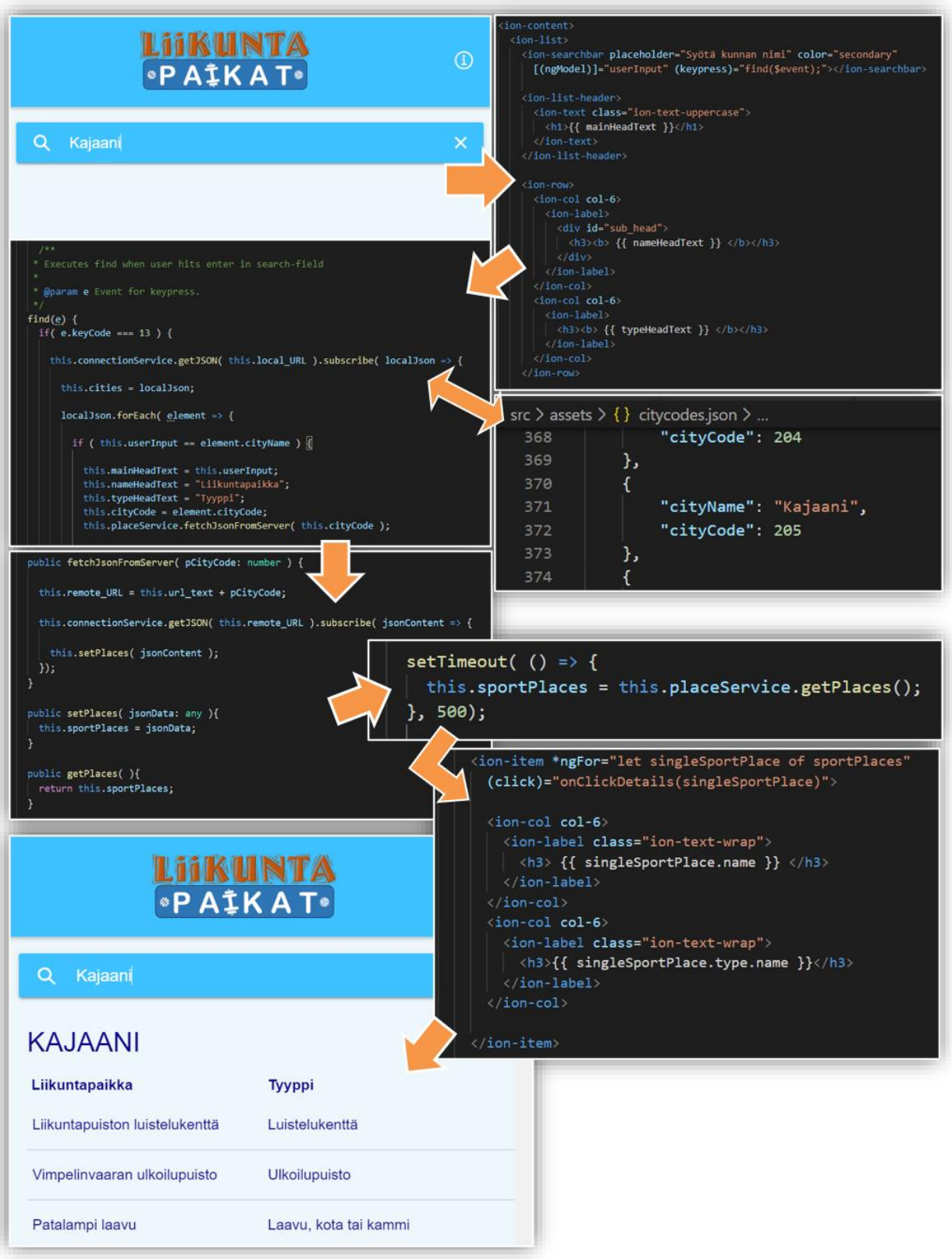
KUVA 12. Liikuntapaikka-sovelluksen käyttöliittymäkuvat

4.3 Tekninen toteutus

Liikuntapaikka-sovelluksen etusivu on toteutettu home-moduulin avulla. home.page.html-sivun hakukenttä käyttää ion-searchbar -toimintoa. Käyttäjän hakukenttään antama kuntanimi luetaan syötteestä userInput-muuttujaan. Tämän jälkeen keypress-tapahtumankäsittelijän avulla kutsutaan home.page.ts-tiedoston find()-funktiota.

JavaScript-koodi lukee kunnan kuntakoodin paikallisesta citycodes.json-tiedostosta. Kuntakoodi liitetään URL-osoitteeseen, joka toimii parametrina connectionService-palvelun getJson()-funktiossa. getJson()-funktion ja Angularin HttpClient-kirjaston avulla tilataan (subscribe) kunnan tiedot JSON-objektina Lipas-palvelimelta.

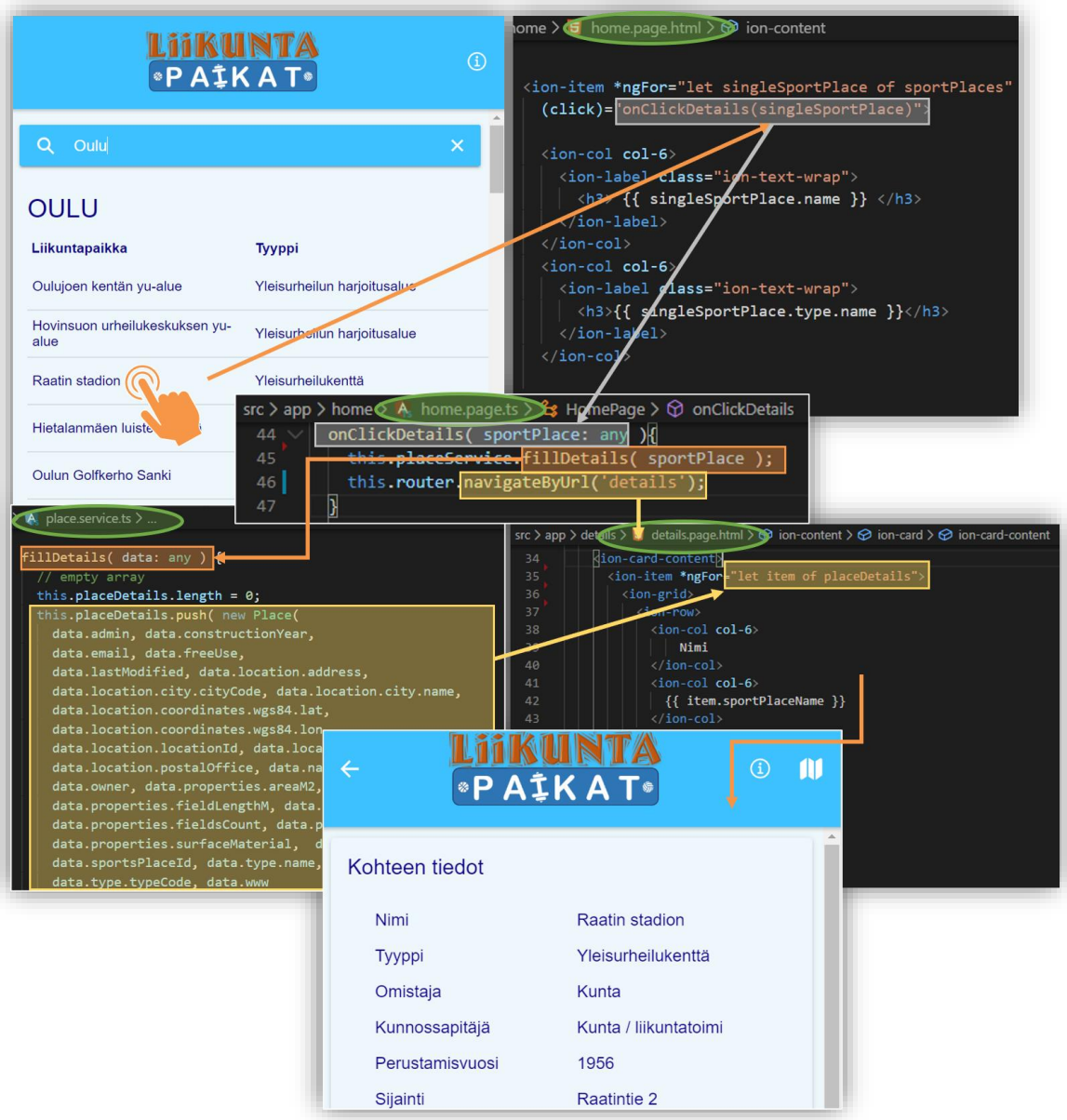
JSON-objektin sisältö sijoitetaan placeService-palvelun avulla sportPlaces-nimiseen muuttujaan, josta home-moduuli poimii liikuntapaikkojen nimet ja tyypit esitettäväksi sovelluksen etusivulla. Kuva 13 mallintaa edellä selitettyä home-moduuliin liittyvää toiminnallisuutta.



KUVA 13. Sovelluksen etusivun esittämiseen tarvittavan tiedon hakeminen ja jäsentäminen

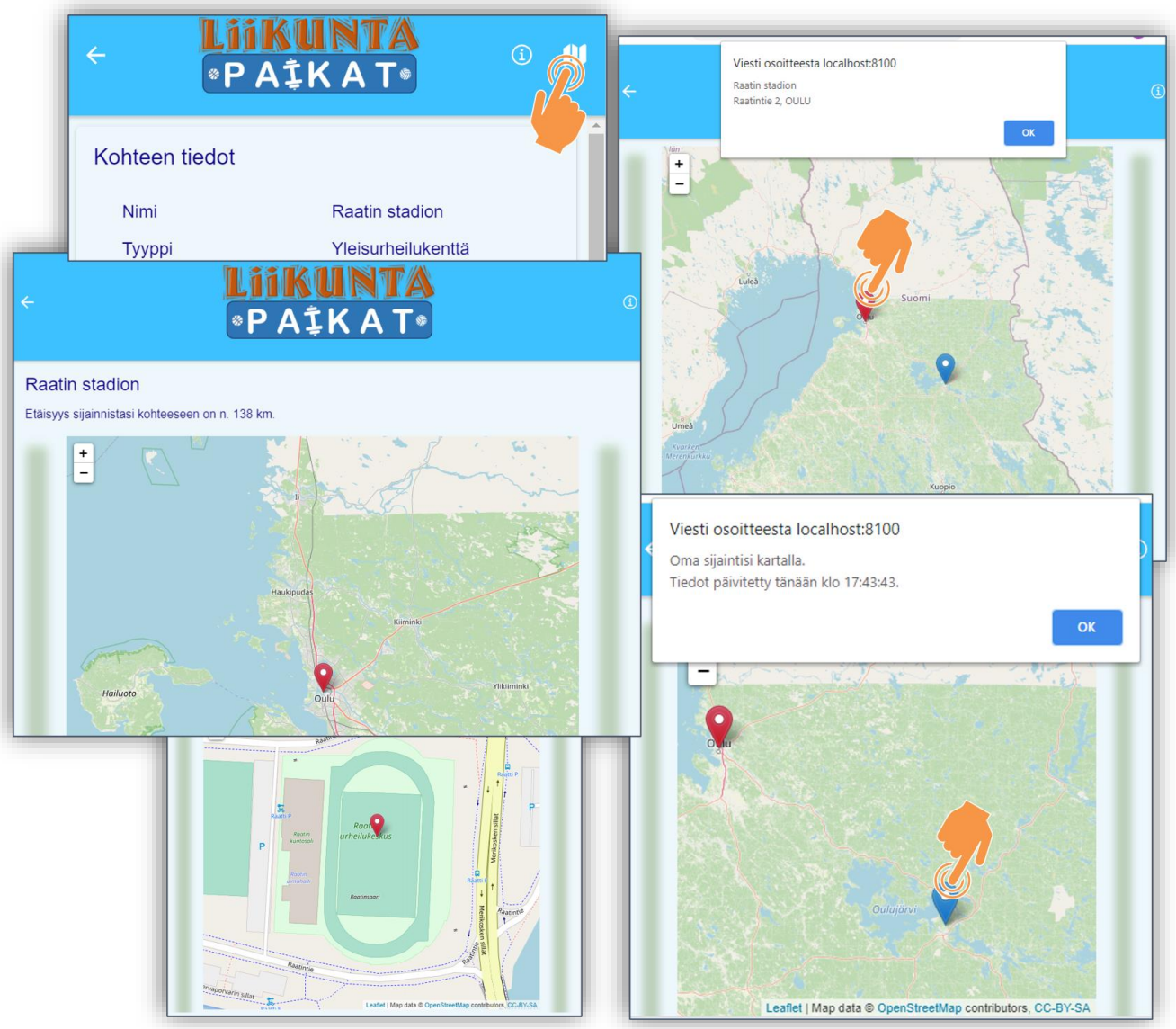
Kohteen tiedot -sivulla esitetään liikuntapaikan yksityiskohtaiset tiedot, kun käyttäjä on ensin valinnut liikuntapaikan etusivun listalta. Etusivun toteuttavan HomePage-koodin kuuntelija reagoi käyttäjän valintaan kutsumalla onClickDetails()-funktiota, joka taas kutsuu PlaceService-palvelusta löytyvää kohteen yksityiskohdat täyttävää fillDetails()-funktiota.

fillDetails()-funktio jäsentää parametrina saamansa JSON-muotoisen datan Place-tyyppiseen taulukkoon. onClickDetails()-funktion jälkimmäisen kutsun avulla vaihdetaan näkymää navigoimalla Kohteen tiedot -sivun mallintavaan details-moduuliin. Kuva 14 mallintaa edellä selvitettyä toiminnallisuutta.



KUVA 14. Kohteen tiedot -sivun täyttäminen käyttäjän valitseman liikuntapaikan tiedoilla

Kohteen tiedot -sivulta on linkki karttasivulle, jossa käyttäjä voi tarkastella kohteen sijaintia karttanäkymässä (kuva 15). Karttasivun otsikkokentässä näytetään liikuntapaikan nimi sekä etäisyys omasta sijainnista liikuntapaikkaan linnuntietä pitkin. Karttanäkymään ilmestyy 2 merkkiä (marker), joista sininen näyttää käyttäjän sijainnin ja punainen näyttää liikuntapaikan sijainnin. Kummankin merkin painallus tuo näkyviin toast-tyyppisen ilmoituksen näkymän yläreunaan.



KUVA 15. Liikuntapaikan ja käyttäjän sijainti karttanäkymässä

Karttanäkymä on toteutettu avoimen lähdekoodiin perustuvien OpenStreetMap-karttapalvelun ja JavaScript-kirjasto Leafletin avulla. OpenStreetMapia rakentavat vapaaehtoiset yhteisön jäsenet, jotka tuottavat ja ylläpitävät karttatietoja muun muassa teistä, rautateistä ja monista muista kohteista kaikkialla maailmassa (OpenStreetMap 2020, viitattu 18.5.2020).

Karttanäkymässä sijainteja ilmentävien merkkien hallinta on toteutettu Leafletin funktioiden avulla. Merkin oletusväri on sininen, mutta käyttäjäkokemuksen parantamiseksi liikuntapaikan ja käyttäjän oman sijainnin merkit haluttiin eri värisiksi. Liikuntapaikka-merkin punainen väri voitiin toteuttaa antamalla map-moduulissa iconUrl-muuttujan arvoksi leaflet-color-markers -kirjaston alta löytyvän kuvan marker-icon-2x-red.png:n sijainti.

Liikuntapaikkojen sijaintitieto oli alun perin tarkoitus hakea Lipas-aineiston coordinates-tietueesta. Kehitystyön edetessä kuitenkin ilmeni, ettei kaikille liikuntapaikoille ollut merkitty koordinaatteja. Tämä ongelma ratkesi kuitenkin lisäämällä tarkistuksia JSON-objektiin käsittelyyn hasOwnProperty()-funktion avulla. Näin voidaan selvittää onko koordinaatteja määritetty ja jos ei, niin sitten konvertoidaan osoitetieto koordinaateiksi leaflet-geosearch -projektista tuodun OpenStreetMap-Providerin search()-funktion avulla.

Sovellustietoja-sivulle (kuva 16) päästään painamalla sovelluksen yläosan info-painiketta. Sivulla on nähtävissä sovelluksen versio ja millä laitteistoilla se toimii. Lisäksi sivulla esitellään lyhyesti sovelluksen käyttämät avoimen datan -palvelut.

KUVA 16. Sovellustietoja-sivun sisältö

4.4 Testaus

Sovelluksen toimivuutta on tarkasteltu lähinnä Ad hoc -testauksena ilman testaussuunnitelmaa ja dokumentointia. Testausta suoritettiin jatkuvasti Google Chrome -selaimella sitä mukaa, kun toteutus eteni. Chromen ohjelmistokehittäjän työkaluilla on melko kätevää tutkailla web-elementtien toimintaa ja sijoittelua sekä verkkoliikennettä. Lähdekoodiin sijoitettujen testiprinttien avulla on voitu seurata muuttujien arvojen oikeellisuutta ohjelman suorituksen eri vaiheissa.

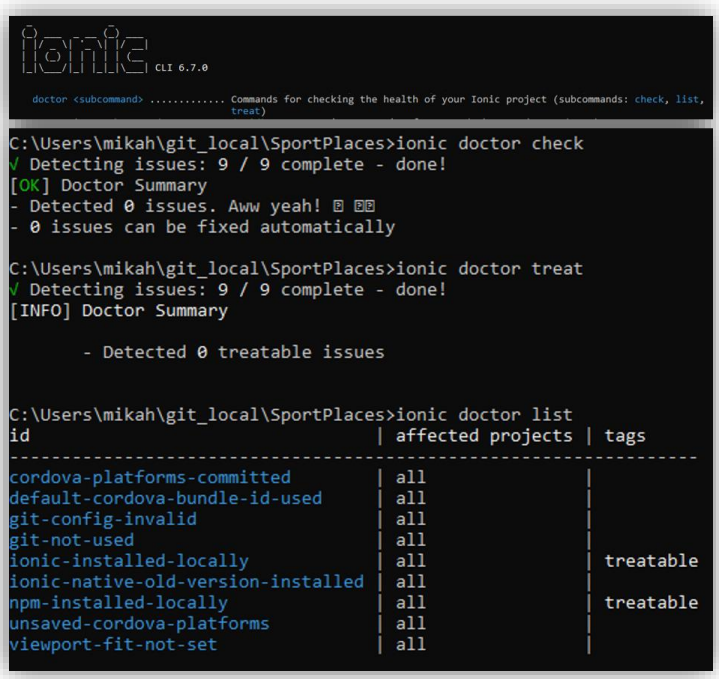
Tiedon hakemisessa palvelimelta oli aluksi ongelmia, kun sekä Chrome- että Firefox -selaimet rajoittivat liikennettä ulkopuolisen verkkoresurssin suuntaan. Syyksi selvisi Cross-Origin Resource Sharing -mekanismi (CORS), jonka avulla selaimet ja webview-komponentit rajoittavat turvallisuussyistä sovelluksista tehtäviä HTTP- ja HTTPS-pyyntöjä erilaisiin verkkoresursseihin. Rajoittamisen syynä ovat pääasiassa käyttäjän tietojen suojaaminen ja sovellusta vaarantavien hyökkäysten estäminen. (Ionic 2020c, viitattu 15.4.2020.)

Sovellustestauksen helpottamiseksi Chromelle asennettiin *CORS: Access-Control-Allow-Origin* -lisäosa. Tämä lisää HTTP-vastauksen Header-lohkoon säännön, jolla hyväksytään ulkopuolisen verkkoresurssin käyttö (MyBrowserAddon 2020, viitattu 15.4.2020). Mobiililaitetestauksessa ilmennyt CORS-ongelma ratkesi https://cors-anywhere.herokuapp.com/ -verkkosivustolta löytyvän API:n avulla, joka sisällyttää CORS-headerin HTTP-pyynnön vastaukseen.

Sovelluksen ollessa käynnissä, taustalla toimivat myös Ionicin Angular-pohjaiset testit. Ne asentuvat ympäristöön automaattisesti, kun Ionic-projekti on perustettu Ionic CLI:n avulla. Testit sijaitsevat tiedostoissa, joiden nimessä on spec-sana, esimerkiksi home.page.spec.ts. Testit otetaan käyttöön antamalla komentorivillä komento *npm test.*

Mobiilisovelluksen paikallisen JSON-tiedoston validointiin käytettiin selainpohjaista JSONLint -palvelua. jsonlint.com -osoitteesta löytyvässä palvelussa tarkistettava tiedosto kopioidaan tekstikenttään. Validate JSON -painikkeen painalluksen jälkeen, palvelu ilmoittaa Result -kentässä onnistuiko tiedoston jäsentäminen (parse).

Ionic CLI:stä löytyy komento *ionic doctor* optioilla *check, list ja treat* (kuva 17). Näillä komennoilla voidaan tarkistaa Ionic-projektin kunto (health). Toiminnon tarkoitus on havaita järjestelmässä ilmenevät ongelmat ja pyrkiä tarjoamaan ratkaisuja niiden selvittämiseksi.



KUVA 17. Ionic CLI -näkymä Ionic-projektin kuntotarkastuksen tuloksista

Liikuntapaikka-sovelluksen päätelaitetestaus tapahtui Samsung Galaxy S10+ -puhelimella, jossa on Android 9 Pie -käyttöjärjestelmä. Yhteyden rakentamiseksi puhelimen ja tietokoneen välille, tietokoneelle oli asennettava Android Studio. Puhelimesta piti ensin aktivoida *Sovelluskehittäjien asetukset*, sitten laite kytkettiin USB-kaapelilla kiinni tietokoneeseen ja lopuksi sallittiin USB-vianetsintä.

Tietokoneen komentoriviltä annettu komento *ionic cordova run android -l* käynnisti mobiisovelluksen puhelimessa. Mobiililaitteessa ajettavaa sovellusta oli mahdollista debugata samaan tapaan kuin tietokoneelta ajettaessa Chromen DevToolsissa, joka löytyi osoitteesta chrome://inspect/#devices.

4.5 Julkaiseminen

Julkaisuprosessin avulla Android-sovellukset tuodaan käyttäjien saataville. Julkaisemisessa suoritetaan kaksi päävaihetta. Ensimmäisessä vaiheessa valmistellaan applikaation julkaisuversio, jonka käyttäjä voit ladata ja asentaa omille Android-päätelaitteilleen. Julkaisuvaiheessa hoidetaan sovelluksen markkinointi, myynti ja jakelu asiakkaille. (Google Developers 2020, viitattu 3.5.2020.)

Android-sovelluksen julkaisemiseen on käytettävissä useita keinoja. Sovellus voidaan julkaista markkinapaikan kautta. Julkaiseminen voidaan tehdä myös oman nettisivun kautta tai sitten applikaatio voidaan lähettää suoraan käyttäjille. (Google Developers 2020, viitattu 3.5.2020.)

Opinnäytetyössä syntyvän mobiilisovelluksen jakelukanavaksi riittäisi tuotteen kehitysasteen perusteella suora jakelu käyttäjille. Oppimisprosessin kannalta oli kuitenkin tärkeää havainnoida, miten sovelluksen kehitysversio jalostetaan valmiiksi ohjelmistotuotteeksi jonkin markkinapaikan, tässä tapauksessa Google Playn, kriteereillä.

Käyttäjätilin rekisteröimisessä Google Play -palveluun on hyväksyttävä kehittäjien jakelusopimus ja Consolen käyttöehdot. Kehittäjän rekisteröitymismaksu on 25 dollaria ja sitä varten palveluun on liitettävä maksukortti. Ennen Google Play Consolen käyttöönottoa on syötettävä vielä yhteystietoja kehittäjäprofiilin luomiseksi. Tämän jälkeen on mahdollista luoda julkaistava sovellus, joka menee Google Playn tarkistusprosessiin.

5 POHDINTA

Opinnäytetyöaiheen miettiminen ja varsinkin työn aloittaminen oli pitkä prosessi. Aiheen kartoitus lähti käyntiin jo toukokuussa 2018, kun erään yhteistyökumppanin kanssa käytiin keskusteluja urheilujoukkueen faniapplikaation tuottamiseksi Ionic-ympäristöllä.

Noin vuosi tästä eteenpäin keskusteltiin toisessa yhteydessä mahdollisuudesta tutkia alustakohtaisen kehitysympäristön, hybridin sovelluskehyksen ja alustariippumattoman sovelluskehyksen eroja Android Studion, Ionic Frameworkin ja React Nativen kautta. Tämä näkökulma olisi laajuudessaan jättänyt kunkin osa-alueen tutkinnan enemmän pintapuoliseksi ja työstä olisi todennäköisesti tullut hyvin teoriapitoinen. Tärkein henkilökohtainen motiivi opinnäytetyön tekemiselle oli kuitenkin oppia tuottamaan ohjelmakoodia uudessa ympäristössä, joten ideaa piti vielä kypsytellä.

Toki mietinnässä oli muitakin kuin mobiilisovelluksiin liittyviä aiheita. Mutta aina ajatus on kuitenkin palannut siihen, että olisi mielenkiintoista perehtyä mobiiliohjelmoinnin saloihin. Lopulta tutkimuskohteeksi valikoituikin Ionic-sovelluskehyksellä toteutettava mobiilisovellus, joka hyödyntää avoimen datan palvelua. Näin backend-toteutuskaan ei vienyt resursseja, vaan oli mahdollista keskittyä olennaiseen eli frontendin opiskeluun, suunnitteluun ja toteuttamiseen.

Tämä suhteellisen pitkä opinnäytetyöprosessi opetti aiheen rajaamisen tärkeyttä. On tärkeää saada aiheen laajuus vastaamaan käytettävissä olevia resursseja. Mutta resurssien mitoitus on usein hankalaa ohjelmistotyössä, sillä kokemusperäinen havainto vuosien varrelta on, että ohjelmistoprojekteilla on taipumusta kasvaa työläämmiksi kuin ennakkoon on osattu arvioida.

Suurin haaste ja hidaste opinnäytetyön tekemistä ajatellen on ollut työelämästä johtuva henkilökohtainen aika- ja energiaresurssien vajaus. Kun työskentelee arkipäivisin ohjelmistotehtävien parissa, on vaikeaa motivoitua opiskelemaan uusia ohjelmointialan osa-alueita iltaisin ja viikonloppuisin. Ratkaisu resurssiontiongelmiin oli lopulta siirtyä tekemään lyhyempää työaikaa.

Opinnäytetyöprosessin myötä opinnäytetyön tekijä joutui kohtaamaan useita itselleen uusia osa-alueita, kuten mobiiliohjelmointi, Ionic- ja Angular-sovelluskehykset, avoimen datan hyödyntäminen ja julkaiseminen sovelluskaupassa. Aiheen suhteen olisi varmasti ollut löydettävissä helpompia ja

tutumpia kehityskohteita, mutta tavoite oli jo opintojen alusta lähtien yrittää oppia mahdollisimman paljon uusia asioita. Siitä tavoitteesta ei opinnäytetyönkään kohdalla tingitty.

Opinnäytetyö oli tarkoitus toteuttaa pienen ohjelmistoprojektin tavoin, jossa myös raportin laatiminen oli pilkottu projektitehtäviksi. Tämän työskentelymallin ajatuksena oli tutustua uusiin työkaluihin myös projektin- ja versionhallinnan puolelta, joka mahdollisesti kehittäisi myös työelämän taitoja. Alan seuraaminen ja itsensä kehittäminen ovat tärkeässä roolissa jatkuvasti kehittyvässä ohjelmistotyössä.

Koska uusia osa-alueita oli useita ja raportin laatiminen vei aikaa, demosovellus piti toteuttaa melko yksinkertaisessa muodossa. Tätä opinnäytetyössä tehtyä sovellusversiota voidaan pitää ensimmäisen vaiheen sovelluksena, jonka pääasiallinen tarkoitus on hakea data palvelimelta ja jalostaa siitä informaatiota käyttäjäystävälliseen esitysmuotoon. Käyttäjällä tarjotaan mahdollisuus tehdä jotakin toimia sovelluksessa.

Jatkossa sovellukseen voisi tuoda lisää käyttäjäkokemusta parantavia ominaisuuksia. Tällaisia voisivat olla suosikkipaikkojen tallentaminen muistiin, liikuntapaikkojen haku muillakin ehdoilla kuin pelkällä kunnan nimellä ja tarkempi etäisyyden mittaus eri kohteiden välillä.

Opinnäytetyön tärkeimpiä anteja oli se, että on pystynyt kehittymään sekä uuden oppijana että todennäköisesti myös työntekijänä. Tärkeä oppi on myös parantunut tietämys siitä, mitä tietolähteistä kannattaa hyödyntää, kun informaatioa on lisääntyvissä määrin tarjolla erilaisissa yhteyksissä.

LÄHTEET

Angular 2020a. Introduction to services and dependency injection. Viitattu 8.5.2020. https://angular.io/guide/architecture-services.

Angular 2020b. Introduction to Angular concepts. Viitattu 16.4.2020, https://angular.io/guide/architecture.

Angular 2020c. Introduction to components and templates. Viitattu 10.5.2020, https://angular.io/guide/architecture-components#directives.

Angular 2020d. Template syntax. Viitattu 11.5.2020, https://angular.io/guide/template-syntax.

Angular 2020e. NgClass. Viitattu 11.5.2020, https://angular.io/api/common/NgClass.

Daityari 2019. Angular vs Vue vs React: Which Framework to Choose in 2019. Viitattu 9.5.2020. https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

freeCodeCamp 2020. What's the difference between JavaScript and ECMAScript?. Viitattu 13.5.2020, https://www.freecodecamp.org/news/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5/.

gitconnected 2020. Angular vs React vs Vue: Which Is the Best Choice for 2020?. Viitattu 18.5.2020. https://levelup.gitconnected.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2020-81f577697c7e

Google Developers 2020. Publish your app. Viitattu 3.5.2020. https://developer.android.com/studio/publish.

Ionic 2020a. What is Ionic Framework? Viitattu 11.3.2020, https://ionicframework.com/docs/intro.

Ionic 2020b. Core Concepts. Viitattu 11.3.2020, https://ionicframework.com/docs/intro/concepts.

Ionic 2020c. CORS Errors. Viitattu 15.4.2020, https://ionicframework.com/docs/troubleshooting/cors.

Ionic 2020d. Ionic Angular Overview. viitattu 16.4.2020, https://ionicframework.com/docs/angular/overview#angular-version-support.

Ionic 2020e. Web View. Viitattu 16.4.2020, https://ionicframework.com/docs/core-concepts/webview.

Ionic 2020f. Colors. Viitattu 13.5.2020, https://ionicframework.com/docs/theming/colors.

Ionic 2020g. Layout/Structure. Viitattu 13.5.2020, https://ionicframework.com/docs/layout/structure

Ionic 2020h. Testing. Viitattu 16.5.2020, https://ionicframework.com/docs/angular/testing.

Ionic 2020i. Release Notes. Viitattu 17.5.2020, https://ionicframework.com/docs/reference/release-notes.

JavaTpoint 2020. What is Ionic Framework?. Viitattu 12.5.2020. https://www.javatpoint.com/what-is-ionic-framework

Kirupa 2020. Understanding WebViews. Viitattu 16.4.2020. https://www.kirupa.com/apps/webview.htm.

Mawson, G. 2015. Introduction to custom Cordova plugin development. Viitattu 16.4.2020, https://blogs.oracle.com/mobile/introduction-to-custom-cordova-plugin-development.

Microsoft 2020. TypeScript - JavaScript that scales. Viitattu 8.4.2020, https://www.typescript-lang.org.

MyBrowserAddon 2020. Allow CORS: Access-Control-Allow-origin. Viitattu 15.4.2020, https://mybrowseraddon.com/access-control-allow-origin.html.

OpenStreetMap 2020. OpenStreetMap - Tietoja. Viitattu 18.5.2020, https://www.openstreet-map.org/about.

React 2020a. React - A JavaScript library for building user interfaces. Viitattu 13.5.2020, https://reactjs.org/.

React 2020b. Tutorial: Intro to React. Viitattu 13.5.2020, https://reactjs.org/tutorial/tutorial.html.

RhodeCode Enterprise 2020. Version Control Systems Popularity in 2016. Viitattu 26.6.2020, https://rhodecode.com/insights/version-control-systems-2016.

Statcounter Global Stats 2020. Mobile Operating System Market Share Worldwide Mar 2019 - Mar 2020. Viitattu 15.4.2020. https://gs.statcounter.com/os-market-share/mobile/worldwide.

Statista 2020. Cross-platform mobile framworks used by software developers worldwide as of 2019. Viitattu 23.6.2020. https://www.statista.com/statistics/869224/worldwide-software-developer-wor-king-hours/.

Tarvainen, J. 2016. Mikä on TypeScript?. Viitattu 8.4.2020, https://symfony.fi/artikkeli/mika-on-ty-pescript.

Trello 2020. What is Trello?. Viitattu 8.4.2020, https://help.trello.com/article/708-what-is-trello.

Vue.js 2020. What is Vue.js?. Viitattu 15.4.2020. https://vuejs.org/v2/guide/#What-is-Vue-js.