Sampo Silvennoinen

Hybridisovelluskehitys Android-laitteille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

6.11.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Sampo Silvennoinen Hybridisovelluskehitys Android-laitteille 39 sivua 6.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Asiakasyrityksen toimitusjohtaja Yliopettaja Kari Salo

Insinöörityön tarkoituksena oli tutkia hybridisovelluskehitystä Android-laitteille ja tehdä arvio teknologian soveltumisesta asiakasyrityksen käyttöön. Pyrkimyksenä oli toteuttaa laaja katsaus eri teknologioihin, ja sen pohjalta valita yksi hybridisovelluskehys esimerkkisovellusprojektiin. Esimerkkisovelluksen tarkoitus oli kattaa asiakasyrityksen yleisimmät käyttötarpeet ja toimia pohjana lopulliselle arviolle teknologian soveltuvuudesta.

Työ toimii laajahkona tietopakettina hybridisovelluskehityksestä ja sen vahvuuksista ja heikkouksista natiivisovelluksiin ja web-sovelluksiin nähden.

Esimerkkisovellus toteutettiin vaaditulla tavalla käyttäen pohjana Supersonic-hybridisovellussovelluskehystä. Supersonicin lisäksi hyödynnettiin suosittuja HTML5-tekniikoita, kuten AngularJS ja Bootstrap. Lopputuloksena oli myös ilman internetyhteyttä toimiva hybridisovellus, joka vastaa toiminnaltaan natiivisovellusta.

Insinöörityö tarjosi paljon lisätietoa asiakasyritykselle ja uutta kokemuspohjaa työn tekijälle. Työssä havaittiin hybridisovellusten olevan sopiva vaihtoehto asiakasyrityksen kehittämille natiivisovelluksille, mutta huonosti soveltuviksi korvaamaan yrityksen web-sovellukset. Yksinkertaisten natiivisovellusten korvaajana hybridisovellus säästää aikaa, mutta yksinkertaisen web-sovelluksen korvaajana hybridi on tarpeettoman monimutkainen vaihtoehto.

Avainsanat	hybridisovellus, HTML5, Supersonic, JavaScript, AngularJS

Author Title Number of Pages Date	Sampo Silvennoinen Hybrid application development for Android devices 39 pages 6 November 2015
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	CEO of the client company Kari Salo, Senior Lecturer

The purpose of this thesis was to study hybrid application development for Android devices and evaluate its potential worth for the client company. The aim was to implement a comprehensive review of different approaches to creating a hybrid app and then choose the most suitable technology for developing an example application. The requirements for the example application state that it must cover the most usual needs the company has for an application and thus function as a basis for evaluating the possible advantages of hybrid application development. An extra point of interest was developing offline functions for handling the lack of internet connection.

The example application was developed using Supersonic as a hybrid application framework. In addition to Supersonic the application also makes use of popular HTML5 technologies, like AngularJS and Bootstrap. The end result is an offline-capable application which acts similarly to a native application.

The thesis offered a large amount of new data to the client company and also provided valuable new experience to the researcher of the thesis. The thesis concludes that hybrid applications are a good alternative to simple native applications, but cannot as easily replace simple web applications. Hybrid application development saves time when compared to a native application, but is unfortunately too complicated to replace web applications.

Keywords	Hybrid app, HTML5, Supersonic, JavaScript, AngularJS

Sisällys

Lyhenteet

HTML5	Hypertext Markup Language, verkkosivujen luomiseen tarkoitetun kuvauskielen uusin versio.
CSS3	Cascading Style Sheets, verkkosivuille kehitetty merkintäkieli, joka määrittää dokumentin tyylin.
PHP	PHP: Hypertext Preprocessor, ohjelmointikieli, jota sovelletaan etenkin verkkosivujen palvelinpäässä.
AJAX	Asynchronous JavaScript and XML. Laajempi termi kuvaamaan epäsynkronista tiedonsiirtoa palvelimelta asiakassovellukseen.
JSON	JavaScript Object Notation, tiedonvälitykseen tarkoitettu tallennusmuoto, jota käytetään etenkin palvelimen ja sovelluksen väliseen tiedonkulkuun.
XML	Extended Markup Language, sekä dokumentin sääntöjen määritykseen että tiedonvälitykseen tarkoitettu merkintäkieli.
MVC	Model-View-Controller, ohjelmistoarkkitehtuurityyppi, joka erottaa mallin, näkymän ja käsittelijän logiikat toisistaan.
API	Application programming interface, ohjelmointirajapinta, lista sääntöjä ja protokollia, joiden avulla saadaan tietoa ulkoisesta lähteestä.
SDK	Software development kit, tarjoaa työkaluja ja kirjastoja ohjelman kirjoittamiseen tiettyyn ympäristöön.
CLI	Command line interface, eli Windowsin komentoriviin pohjaava käyttöliittymä.
SPA	Single-page application, yhden sivun sovellus. Web-sovellus, jossa kaikki logiikka tapahtuu yhdellä sivulla monen sivun navigaation sijasta.
SQL	Structured query language, kieli relaatiotietokantojen hallintaan ja muokkaamiseen.

1 Johdanto

Insinöörityön tarkoituksena on tutkia hybridisovelluskehitystä Android-laitteille ja tehdä arvio teknologian soveltuvuudesta asiakasyrityksen käyttöön. Asiakasyritys on viihde-elektroniikan valmistaja ja maahantuoja, jonka tuoteportfolioon kuuluu myös Android-pohjainen digitaaliseen kyltitykseen ja rekisterinkeruuseen soveltuva Mediastand-konsepti. Mediastand-konseptiin kuuluva sovelluskehitys on perinteisesti pohjannut web-sovelluksiin, mutta yritys harkitsee muita teknologioita jatkokehittäessään konseptia.

Asiakasyrityksen tarjoama Mediastand-konsepti koostuu Android-tabletista, metallisesta esittelystandista sekä asiakkaalle räätälöidystä ohjelmistosta ja ulkoasusta. Yleisimpiä käyttötarkoituksia ovat muun muassa esittelystandit, infostandit, asiakaspalvelupisteet, digitaaliset esittelyalustat tai alustat kilpailuihin osallistumiseen. Lopputulos riippuu asiakkaan tarpeesta ja toiveesta – yksinkertaisimmillaan Mediastand voi tapahtumassa esittää asiakkaan jo olemassa olevaa verkkosivua tai verkkokauppaa, mutta se voi myös yhtä hyvin olla asiakkaalle tehty uusi sovellusratkaisu. Asiakasvaatimusten kasvaessa ja uusia haasteita kohdattaessa myös vaatimukset Mediastandiin räätälöitäviin ohjelmistoihin kasvavat.

Tähän mennessä ratkaisut ovat olleet web-sovelluksia, jotka lukitaan kolmannen osapuolen tuottaman kioskisovelluksen taakse. Lukitsemisen tarkoituksena on estää käyttäjää harhailemasta pois halutusta sisällöstä, jolloin Mediastand toimii myös ilman valvontaa. Web-sovellukset ovat riittäneet monille asiakkaille, mutta työn tarkoituksena on arvioida hybridisovelluksia rinnalle täydentämään web-sovellusten puutteita. Useamman kuin yhden asiakkaan toiveet ovat kuulostaneet haastavilta web-sovelluksina, mutta kuitenkin turhan vähäpätöisiltä natiivisovelluksen kehittämiseksi. Natiivisovelluksen kehitysaika ja päivitettävyyden haasteet toisivat lisäkustannuksia myös asiakkaille.

Tyypillinen Mediastandin käyttöympäristö on messuhalli, jossa suuren liikenteen vuoksi paikallinen WiFi on ruuhkautunut ja myös muut datayhteydet kärsivät. Yksi tarkasteltava näkökanta on hybridisovellusten kyky toimia offline-tilassa.

2 Mobiilisovellukset

Mobiililaitteiden laskentatehon ja suosion kasvun myötä myös laitteiden käyttötarkoitus on laajentunut. Aiemmin mobiililaitteiden sovelluksiksi voitiin pääosin laskea kalenterin tai kontaktiluettelon kaltaiset puhelimen käyttöä helpottavat sovellukset tai yksinkertaiset pelit. Internetyhteyden vakiintuminen puhelimissa ja siirtyminen kohti älypuhelimia loi mobiilisovelluksille kysynnän ja markkinat, jotka paikoittain ylittävät itse puhelinmarkkinoissa liikkuvan rahan. Laaja sovellustarjonta ja sen luomat mahdollisuudet ovat siirtäneet painopistettä pois soittamisesta ja tekstiviesteistä, tai tietyissä tapauksissa jopa korvanneet ne uusilla sovellusratkaisuilla. Ero tietokonesovellusten ja mobiilisovellusten välillä pienenee jatkuvasti. [1, s. 337–339.]

Uusi laiteympäristö luo kuitenkin aina omat haasteensa ja toimintamallinsa. Älypuhelinten alalla yksikään käyttöjärjestelmä ei ole selvä kilpailun voittaja, joten myöskään sovelluskehitys älypuhelimille ei ole vakiintunut yhden alustan standardien mukaiseksi. Applen iOS-mobiilisovelluskehitys tehdään Xcodella käyttäen joko Objective-C- tai Swift-kieltä [2], Androidin sovellukset kirjoitetaan XML:n ja Javan yhdistelmällä [3], Windows Phone taas suosii C#-ohjelmointikieltä [4]. Alustojen sisällä tapahtuu myös pienemmän mittakaavan kilpailua, sillä keskustelua ja väittelyä on käyty esimerkiksi web-sovellusten eduista natiivisovelluksiin nähden.

Mikäli mobiilikehitysmaailman jakaa sekä kilpaileviin käyttöjärjestelmiin että vielä käyttöjärjestelmän sisäisiin kiistoihin loogisimmasta tavasta luoda sovelluksia, päätyy nopeasti katsomaan pirstoutunutta kehittäjäkuntaa. Pirstoutuminen taas johtaa väistämättä siihen, että laajaa käyttäjäkuntaa tavoitellessa tulee äkkiä hallitakin sovelluskehitys monella eri alustalla. Mobiilisovellukset ovat vakiintuneesta asemastaan huolimatta yhä kuvaannollinen villi länsi, missä paljon voi muuttua lyhyessä ajassa. [5, s. 8–9.]

2.1 Natiivisovellukset

Natiivisovellukset ovat perinteisin sovellustyyppi mobiililaitteissa. Ne on alusta asti rakennettu nimenomaan suunnitellulle alustalleen eivätkä välttämättä ole helposti käännettävissä muille alustoille. Suoraan päätelaitteen käyttöjärjestelmän huomioonottava suunnittelu tarjoaa natiivisovelluksille usein helpon pääsyn loppukäyttäjän laitteen ominaisuuksiin.

Android-käyttöjärjestelmän sovelluskehitys tehdään virallisia teitä noudattaen Android SDK:n kautta. Toiminnallisuus sovelluksiin ohjelmoidaan Sun Microsystemsin kehittämällä Java-ohjelmointikielellä. Java on vahvasti olio- eli luokkapohjainen ohjelmointikieli, joka on maailman mittakaavassa hyvin laajassa käytössä eri alustoilla [6, s. 4]. Sovelluksen sisältöä ja ulkoasua taas määritetään XML-tiedostoilla [6, s. 23]. Java pyrkii toteuttamaan filosofiaa, että kerran kirjoitettuaan ohjelman Javalla voi sen helposti siirtää toiseen laitteeseen tai ympäristöön, mutta tämä ei käytännössä toteudu mobiilisovellusten tapauksessa. Vaatimus edellä mainittuun ohjelmistokoodin siirtämiseen vaatii, että päätelaite tukee Javan virtuaalikonetta, joka suorittaa kaikki Javalla kirjoitetut ohjelmat. Mobiilisovelluksissa ongelmaksi muodostuu se, että suosituimmista mobiilikäyttöjärjestelmistä vain Android pohjaa Javaan. Mobiililaitteet myös asettavat omia vaatimuksiaan, eikä esimerkiksi pelkkään Javaan ja XML:ään pohjaavia sovelluksia näe erityisen paljoa Androidin ulkopuolella. [7.]

Javaa pidetään kohtuullisen helppona kielenä aloittelijoille, mutta sen rakenne on luonnostaan hyvin pitkää ja moni muu kieli on ilmaisukykyisempi pienemmällä merkkimäärällä. Vielä äskettäin suosituin yhdistelmä oli käyttää Eclipseä ja Android SDK:ta, mutta viralliseksi työkaluksi on noussut Android Studio. Ohjelma on kuitenkin vielä sen verran tuore, että luotettavaa arviota sen vahvuuksista ja heikkouksista ei ole tehty.

2.2 Web-sovellukset

Web-sovellukset ovat internetselaimessa toimivia sovelluksia, joiden toiminta vaihtelee laajuudeltaan hyvin paljon. Suurin ero natiivisovelluksiin verrattuna on, että web-sovelluksia ei asenneta itse laitteeseen tai kirjoiteta ottaen huomioon päätelaitteen tyyppi. Web-sovellukset sijaitsevat etäpalvelimella, ja niiden sisältöön päästään käsiksi vasta web-selaimella. Ero web-sovelluksella ja internetsivulla on joko pieni tai olematon; käytännössä eron määrittää se, tarjoaako sivu toiminnollisuutta. [8.]

Koska web-sovellukset on tehty toimimaan internetselaimissa, ne eivät ota natiivisovelluksen tavoin kantaa päätelaitteen ominaisuuksiin tai käyttöjärjestelmään. Eri selaimissa on pieniä eroja, mutta valtavirran teknologiaa käytettäessä tavallinen internetsivu tai web-sovellus toimii lähes missä tahansa modernissa selaimessa. Alustariippumattomuus on suuri etu natiivisovellukseen nähden, sillä riippumattomuus laitteesta kasvattaa sovelluksen potentiaalisen käyttäjäkunnan moninkertaiseksi. Haittapuolena on se,

että perinteisesti web-sovellukset eivät yllä samaan suorituskykyyn eikä niillä ole niin laajaa ominaisuuskirjoa kuin natiivisovelluksilla. Web-sovellukset ovat myös auttamattomasti riippuvaisia internetyhteydestä, mutta käyttötarkoituksesta riippuen natiivisovellukset saattavat toimia täysin ilman internetiä.

Toinen suuri ero web-sovelluskehityksessä on myös käytetyt teknologiat. Myös Javalla voi tehdä web-sovelluksia, mutta suosio on kääntynyt kohti yksinkertaisempia ja kevyempiä kieliä. Suosituin yhdistelmä web-sovelluksen tekemiseen on HTML5, JavaScript ja CSS. Kielinä niitä kaikkia pidetään Javaa aloittelijaystävällisempinä ja oppimiskäyrä on pääosin vähemmän jyrkkä.

2.3 Hybridisovellukset

Hybridisovellukset ovat yhdistelmä natiivisovelluksia ja web-sovelluksia. Tyypillisesti lopputulos on natiivisovellukselta näyttävä päätelaitteelle asennettava sovellus, mutta sovelluksen takana oleva teknologia on enemmänkin web-sovelluksiin pohjaava. Hybridisovelluksia on eriasteisia: osa on enemmän natiivisovellusta, osa on enemmän web-sovellusta. Myös teknologioita näiden kahden yhdistämiseen on monia, mutta tyypillisesti kyseessä on web-kielillä kirjoitettu sovellus, joka on kolmannen osapuolen ohjelmistokehystä käyttäen kääritty ohueen ”natiivikuoreen”.

Hybridisovellusten suurin vahvuus on kohtuullinen alustariippumattomuus; web-kielillä kirjoittaessa loppulaitteen käyttöjärjestelmällä on vähemmän väliä. Perinteinen näkemys myös on, että web-sovelluskehittäjiä on helpompi löytää ja/tai kouluttaa kuin natiivisovelluskehittäjiä. Ongelma hybridisovelluksissa kuitenkin on se, että ne eivät välttämättä pääse natiivin lailla käsiksi kaikkiin laitteen ominaisuuksiin yhtä helposti ja myös suorituskyky on pääosin heikompi. Hybridisovellus on siis kompromissi kahden teknologian välillä, ja sen hyödyllisyys riippuu paljon sovelluksen lopullisesta käyttötarkoituksesta ja vaatimuksista. [9.]

3 Hybridisovellukset

Hybridisovelluksen luonteesta tai komponenteista ei ole olemassa yhtä tiukkaa teknistä määritelmää kuin esimerkiksi natiivisovelluksesta. Metodeja hybridisovelluksen tuotta-

miseen on monia erilaisia, ja myös lopputulokset voitaisiin tarpeen tullen jakaa erilaisiin kategorioihin.

Määritelmällisesti hybridisovellus on kirjoitettu jollain muulla ohjelmointikielellä kuin alustan määrittämällä natiivisovelluskielellä (Androidin tapauksessa Java). Tämä ei kuitenkaan tarkoita, että lopputulos on aina sovellus, joka suoritetaan käyttäen jotain muuta kieltä. Yksi malli on tehdä hybridisovellus perinteisellä web-kielellä kuten HTML:n JavaScriptin yhdistelmällä, mutta kääntää se kokoamissovelluksella Javaksi. Englannin kielen ilmaus ”compiled hybrid app”, eli käännetty tai koottu hybridisovellus, kuvaa näitä tapauksia. Kielet harvoin kuitenkaan kääntyvät täydellisesti muodosta toiseen, etenkään kun kyseessä on tulkitusta kielestä käännettyyn tai koottuun kieleen kääntäminen.

Käännettyä sovellusta selvästi suositumpi metodi ovat erilaiset WebView-komponentit. WebView’t ovat sovelluksessa olevia näkymiä, joiden sisältö on perinteistä web-sisältöä. Käytännössä ne ovat sovelluksen osioita, joihin on puolestaan upotettu selainmoottori imitoimaan tavallista internetselainta. Kyseessä on siis yhä natiivisovellus, mutta sen sisällä WebView-komponentti eli WebView-luokka lataa internetsisällön, jonka se pyrkii näyttämään mahdollisimman paljon tavallisen selaimen tavoin [10]. Periaatteen tasolla toiminnan pitäisi olla hyvin paljon sovelluskehitystä yksinkertaistava: sisällön voi tuottaa web-kielillä ja upottaa kaiken Androidin natiiviin WebView-ominaisuuteen, jolloin syntyy parhaimmillaan vaikutelma aidosta sovelluksesta. Käytännössä teknologiassa on havaittu puutteita natiiviin verrattuna, kuten heikompi suorituskyky. Teknologiana WebView on osa Androidin omaa järjestelmää, joten myös Androidin versio voi vaikuttaa siihen, kuinka moderni versio WebView’stä loppukäyttäjän laitteessa on. Tämä on ilmiselvä ongelma, mikäli hybridisovelluksella pyrkii hakemaan etua natiiviin verrattuna tarjoamalla vastaavan suorituskyvyn kaikille alustoille pienemmällä työmäärällä. WebView’n suosiosta riippumatta ongelmat hitauden suhteen eivät ole huomattavasti muuttuneet, vaan sen kumoamiseen on pyritty kehittämään oikoteitä oikeiden ratkaisujen sijasta. [11.]

WebViewt ovat myös osa natiiveja Android-sovelluksia, joten ajoittain raja hybridin ja natiivin välillä hämärtyy. Esimerkiksi pieni osa natiivisovelluksen näkymistä saattaa olla WebView-komponentteja, jotka esittävätkin web-sisältöä.

3.1 Teknologiat

Hybridisovellusten teknologiakomponentit voidaan jakaa kolmeen osaan: sovellukseksi kokonaisuuden käärivä natiivikuori, sovelluksen oman toiminnan määrittävät web-kielillä kirjoitetut elementit sekä vapaaehtoisena lisänä sovelluksen ulkoinen palvelin. Natiivikuoresta vastaavat käytännössä erilaiset kääntäjät ja kehitysympäristöt, jotka upottavat esimerkiksi HTML5-sovelluksen ympäristöön, joka toimii Androidin perspektiivistä natiivisovelluksen tavoin. Natiivikuori ei liity siis vain lopputuloksen käärimiseen sovellukseksi, vaan se toimii myös kehitysympäristönä, joka tarjoaa web-kielille paremman mahdollisuuden kommunikoida päätelaitteen kanssa. Kehitysympäristöstä riippuen tämä natiivikuori voi olla joko yksinkertainen WebView-käärijä tai sitten hyvinkin laaja paketti joka pyrkii tekemään paljon omaa optimointia.

Määritelmällisesti hybridisovellukseen ei tarvitse kuulua laitteen ulkopuolista palvelinta tarjoamaan osan sovelluksen vaatimasta datasta, mutta se on erittäin yleistä. Tyypillinen ratkaisu on erillinen ohjelmointirajapinta, jonne sovellus voi tehdä palvelupyyntöjä. Lähes kaikki sovellukset käyttävät jonkinlaista tietokantaa, jotka usein sijaitsevat etäpalvelimella laitteen oman muistin sijasta.

3.1.1 HTML5-merkintäkieli

HTML5 on uusin versio pitkäaikaisesta Hypertext Markup Language -standardista, jonka pohjimmainen tarkoitus on esittää www-sisältöä. HTML5 on etenkin pyrkimys yhdenmukaistaa ja modernisoida www-sisältöä ja vastata nykyisiin suurempiin vaatimuksiin web-median suhteen [12]. Merkintäkielen tuoreimman version on tarkoitus korvata HTML 4, XHTML ja osittain myös Adoben Flash. Web-sivut ovat perinteisesti staattisia, mutta HTML5 on askel kohti interaktiivisempia kokonaisuuksia. Käytännössä tämä on askel tavallisista sivuista kohti web-sovelluksia, staattisesta kohti asynkronista [13, s. 219–222].

HTML5 tarjoaa edeltäjiinsä nähden huomattavasti enemmän ominaisuuksia, mutta on myös käsitteenä päässyt kasvamaan alkuperäisen määritelmänsä ulkopuolelle. Usein puhuttaessa esimerkiksi HTML5-sovelluksista tai sivuista tarkoitetaan ratkaisua, jossa on HTML5:n lisäksi myös huomattavasti CSS3-elementtejä ja tyypillisesti jotain laajempaa JavaScript-kirjastoa. Tämä pätee etenkin web- ja hybridisovelluksista puhuttaessa. Hybridisovelluksissa HTML5 vastaa monin tavoin natiivisovellusten puolelta XML-

layoutia, mutta tarjoaa myös itsessään toiminnollisuutta. Sovelluksen pohjan ja ulkonäön rakentaminen HTML5:n ja CSS3:n yhdistelmällä on myös huomattavasti yksinkertaisempaa kuin aidon natiivisovelluksen ulkonäön viimeistely.

3.1.2 CSS3-tyylitiedostot

CSS3 on Cascading Style Sheets -tyylitiedostojen tuorein standardi. Tyylitiedostot yhdessä HTML5-pohjan kanssa määrittävät web-sivujen -ja sovellusten ulkonäön. Alun perin CSS:n vahvuus oli yhdellä tyylitiedostolla useampaan verkkosivun alasivuun vaikuttaminen, mutta nykyisillä CSS3-ominaisuuksilla kyetään myös korvaamaan paljon graafista työtä CSS3-tiedostoilla. Web-sivujen alkuperäisen staattisen ja koruttoman olemuksen vuoksi läpinäkyvyydet, pyöreät kulmat ja haastavammat muodot on perinteisesti jouduttu tuottamaan kuvankäsittelyohjelmilla ja lisäämään oikeaan kohtaan sivun ulkoasua staattisina kuvina. CSS3 tuo ratkaisun moniin näistä ongelmista ja tarjoaa huomattavasti muokattavamman ja helpomman tavan hioa sivun tai sovelluksen ulkoasua.

CSS3 on tuonut mukanaan myös paljon kolmannen osapuolen tekemiä kirjastoja ja sovelluskehyksiä. JavaScriptin tavoin alkaa olla melkein tyypillisempää käyttää yhtä suosituista, laajemmista kirjastoista kuin kirjoittaa kaikki tyylitiedostonsa itse. Etenkin mobiililaitteiden suosion kasvu on luonut uutta painetta suunnitella web-sivut siten, että ne toimivat hyvin myös vaihtelevankokoisilla näytöillä. Useimmat CSS-sovelluskehykset nimenomaan pyrkivätkin tarjoamaan valmiita ratkaisuja responsiiviseen suunnitteluun, jossa sivusto itse reagoi loppukäyttäjän päätelaitteen näytön kokoon [13, s. 10–14]. Internetiä selataan yhä useammin mobiililaitteen kautta, joten ulkoasun suunnittelu pelkkää pöytäkonetta ajatellen rajaa ja vaikeuttaa käyttäjäkunnan palvelemista. Niin sanottu ”mobile first”, eli ”mobiili ensin”, on yksi suunnittelumalli, joka kääntää asetelman päälaelleen ja aloittaa suunnittelun ja optimoinnin pienemmistä laitteista ja laajentaa suurempia näyttöjä kohden [13, s. 56–57].

Suosituimpia CSS3-sovelluskehyksiä ovat Bootstrap, Foundation, Semantic UI ja Skeleton, joista Bootstrap on tällä hetkellä suosituin [14]. Koska responsiivinen suunnittelu on perusvaatimuksia lähes mille tahansa web-kehittäjälle, ei ole yllättävää, että suosituimpiin kuuluvat keskittyvät etenkin mobiiliyhteensopivuuteen. Bootstrap lupaa ratkaisuja responsiivisiin mobiilisivuihin [15], Foundation lupaa samaa suurimmalla kehitysnopeudella [16], Semantic UI panostaa käyttöliittymiin [17] ja Skeleton pyrkii tarjoa-

maan kevyen ratkaisun projekteihin, jotka eivät kaipaa suurta sovelluskehystä toimiakseen [18].

3.1.3 JavaScript- ja HTML5 -sovelluskehykset

HTML5:n myötä JavaScriptin jo ennestään suuri rooli on laajentunut entisestään. Kun sivut alkavat muistuttaa sovelluksia, tarvitsevat kehittäjät myös enemmän työkaluja toimintojen tekemiseen. JavaScript-kirjastojen ja laajempien HTML5-sovelluskehysten erottaminen toisistaan muistuttaa käsitteenä veteen piirrettyä viivaa, mutta käytännössä kummankin tarkoitus on joko helpottaa tai laajentaa HTML5:n ja JavaScriptin kykyä tuottaa sovelluksia. Tyypillinen tavoite JavaScript-sovelluskehykselle voi olla esimerkiksi HTML5:n ”korjaaminen” lähemmäs muiden ohjelmointikielten suunnittelumalleja tai virtaviivaistaa ja yhtenäistää eri ominaisuuksien luomista. Nykyisessä web-kehityksessä on käytännössä osattava vähintään yhtä suuremmista sovelluskehyksistä ja mieluusti tulee olla kokemusta myös muista. [19, s. 2–6.]

JavaScript-kirjastot ja -sovelluskehykset vaihtelevat kooltaan ja ominaisuuksiltaan erittäin paljon: osa tarjoaa ratkaisun yhteen ongelmaan, osa pyrkii olemaan hyvä kaikessa ja ”se oikea tapa” tehdä web-sovelluksia. Esimerkiksi kevyt Hammer.JS tarjoaa valmiit ratkaisut kosketuseleiden toteuttamiseen sivuun tai sovellukseen [20] siinä missä Ember.js:n kaltaiset laajemmat kehykset pyrkivät tuottamaan ratkaisun kaikkeen SPA:n tekemiseen liittyen [21].

jQuery on suosituin vapaan lähdekoodin JavaScript-kirjasto. Se pyrkii yksinkertaistamaan JavaScriptia tarjoamalla oikoteitä käytetyimpiin ominaisuuksiin ja lisäämällä huomattavasti omia toimintoja. Sen laaja suosio ja helpotettu käytettävyys ovat jopa saaneet kritiikkiä osakseen, sillä jQueryn käyttö saattaa poiketa paljonkin tavallisen JavaScriptin käytöstä. Nyrkkisääntö kuitenkin on, että kirjastot ja sovelluskehykset saattavat vaihtua vuosien myötä, mutta JavaScript pysyy ennallaan. Pelkkä jQueryn osaaminen ei välttämättä takaa oppien siirtämistä muihin kirjastoihin. jQueryn suosiosta kuitenkin kertoo jo se, että esimerkiksi työpaikkailmoitukset eivät usein listaa vaatimuksiinsa jQuerya ollenkaan, sillä se on jo käytännössä JavaScriptin synonyymi [19, s. 483–387.]

Muita suosituimpia sovelluskehyksiä ovat AngularJS, BackBone, Ember ja React. Käytännössä kaikkiin termeihin törmää väistämättä lukiessaan moderneista sovelluskehyk-

sistä. Useimmat noudattavat MVC-suunnittelumallin johdannaisia, kuten MVVC, ja pyrkivät olemaan skaalattavia, laajennettavia ja helposti ylläpidettäviä. Jokainen soveltuu myös alustasta riippuen hybridisovelluskehitykseen, joten niillä on myös tärkeä rooli siirrettäessä web-sovellusten osaamista hybridisovelluksiin. [22.]

3.2 Kääntäjät

Kääntäjät tarjoavat kehitysympäristön ja mallin sovelluksen kääntämiseksi valmiiksi Android-sovellukseksi. Eri palveluntarjoajat tarjoavat erilaisia ratkaisuja erilaisin hinnoittelupolitiikoin ja ominaisuuksin. Hybridisovelluskehitys on yhä suhteellisen aikaisessa vaiheessa muihin sovellustyyleihin verrattuna, joten selviä suuntalinjoja tai ohjeita optimaalisen sovelluksen tekemiseen ei ole. Eri kääntäjien näkemys hybridisovelluskehityksen työnkulusta ja lopputuloksesta vaihteleekin huomattavasti. Osa kehitysympäristöistä myös perustuu toiseen suosittuun kehitysympäristöön, jonka ”virheitä ja puutteita” uusi ympäristö ilmoittaa paikkaavansa paremmalla vaihtoehdollaan. Harvat kääntäjät myöskään ovat ilmaisia taikka avoimen lähdekoodin ratkaisuja, joten kääntäjää arvioidessa voi olla haastavaa arvioida ominaisuuksien ja mainospuheiden paikkansapitävyyttä.

Käytännössä kaikkia kääntäjiä yhdistää kuitenkin yksi seikka: itse kehitys tehdään HTML5-tekniikoilla. Tietyt alustat voivat tarjota myös vaihtoehtoisia kieliä, mutta pohjimmiltaan kyse on web-tekniikoista. Moni alusta kuitenkin pyrkii parantamaan käytettävyyttä tai korjaamaan web-kielien puutteita sovelluskehityksessä tarjoamalla integroituna osana omia lisäyksiä tai kirjastoja. Tyypillinen esimerkki on omat HTML-entiteetit tai valmiit CSS-kirjastot, jotka luovat oikoteitä erilaisten mobiilisovelluksille tyypillisten elementtien luomiseen. Parhaimmillaan tämä ei vain tarjoa oikoteitä kehittäjille, joilla on taustaa web-tekniikoista, vaan myös luo vastineita Java-kehittäjille tutuille osille natiivipuolelta.

Sovelluskehittäjän perspektiivistä yksi suurimpia rajoituksia onkin juuri koodin kääntäminen sovellukseksi – useimmat kääntäjät eivät tarjoa mahdollisuutta kääntää projektia valmiiksi sovellukseksi omalla työkoneella, vaan toiminto on esimerkiksi pilvipalvelun päässä. Useimmissa tapauksissa tämä ei ole kehittäjälle ongelma, mutta se lisää uuden harkintakriteerin sopivaa kääntäjää valitessa: voidaanko palvelun luottaa pysyvän toiminnassa ja samanhintaisena. Aina vain yleistyvä yritysmalli on ensin luoda kysyntä

palvelulle joko halvalla hinnalla tai ilmaisuudella ja vasta myöhemmin tehdä tuotto hinnannostolla asiakasmäärän noustua. Osa kääntäjistä tarjoaakin lupauksia, kuten takuun tiettyjen palveluiden pysymisestä ilmaisina myös tulevaisuudessa. Nämä takuut eivät kuitenkaan varmista, että itse yritys pysyisi toiminnassa tulevaisuudessa.

Seuraavaksi esitellään muutamia eri lähestymistapoja hybridisovellusten kääntämiseen. Vaihtoehtoja on monia esiteltyjen lisäksikin.

3.2.1 PhoneGap-sovelluskehys

PhoneGap on suosituimpia kehitysalustoja ja kääntäjiä hybridisovelluskehitykseen. Se perustuu vapaaseen lähdekoodiin ja on saavuttanut huomattavan suosion yli 400 000 käyttäjänsä keskuudessa. PhoneGapin (tai Apache Cordovan) toiminta on oppikirjaesimerkki hybridisovelluksesta: HTML ja CSS määrittävät sovelluksen visuaalisen puolen ja JavaScript määrittää sovelluslogiikan. PhoneGapilla sovellus kyetään kääntämään kaikille suosituimmille alustoille, eli iOS, Android ja Windows Phone. [23.]

PhoneGapia ja Cordovaa käytetään usein synonyymeina toisilleen, mutta käytännössä niillä on huomattava ero: Cordova on PhoneGapin pohjalla lepäävä teknologia, mutta Cordova itse ei ole riippuvainen PhoneGapista. Cordovaa käyttävät myös muut alustat, mutta PhoneGap on puhekielessä jo vakiintunut sen tunnetuimmaksi käyttäjäksi ja usein ”PhoneGap yhteensopivuus” tarkoittaakin käytännössä Cordovaan nojaamista.

PhoneGap myös helpottaa päätelaitteen ominaisuuksiin pääsyä omalla JavaScript-API:llaan, joka tarjoaa luotettavamman tavan sovellukselle kommunikoida laitteen fyysisten osien kanssa. Myös perinteiset web-tekniikat tarjoavat kohtalaisen kyvyn käyttää esimerkiksi mobiililaitteen kameraa, mutta yhteensopivuusongelmia ilmenee etenkin vanhempien Android-versioiden kanssa. PhoneGapin API tarjoaa ratkaisun, jonka tulisi ohittaa kyseiset ongelmat ja taata sujuvampi kommunikaatio elementtien välillä. [24.]

PhoneGap tarjoaa mahdollisuuden koodin kääntämiseen sovellukseksi sekä pilvipalvelun kautta että paikallisesti. Jälkimmäinen vaatii enemmän toimenpiteitä, kuten eri alustojen SDK:ien lataamisen. Pilvipalveluna toimiva PhoneGap Build tosin tarjoaa vain yhden ilmaisen sovelluksen kehittämisen ja vaati siitä eteenpäin pienen kuukausimaksun.

Monen muun kehitysympäristön toiminta joko pohjaa vahvasti PhoneGapiin tai on rakennettu sen päälle. Syinä tähän ovat avoin lähdekoodi ja alustan suuri suosio. Osa kehitysympäristöistä pyrkiikin markkinoimaan omaansa luomalla joko hyvän tai lähes täydellisen yhteensopivuuden PhoneGapin kanssa, jolloin vanha PhoneGap-osaaminen voidaan siirtää uuteen alustaan ja vähentää käyttöönoton monimutkaisuutta.

3.2.2 Appcelerator Titanium -sovelluskehys

Appcelerator Titanium on käyttäjämäärältään ja suosioltaan PhoneGapin suurimpia haastajia. Appcelerator itse ilmoittaa Titaniumin käyttäjämääräksi 660 000 sovelluskehittäjää, mutta on epäselvää, perustuuko luku sovellusympäristön latauskertoihin vai muuhun arvioon. Appcelerator pyrkii tarjoamaan ympäristön, jossa JavaScriptilla toteutetut sovellukset kyetään kääntämään kaikille suurimmille mobiilialustoille tekemällä vain pieniä muutoksia koodiin. Appceleratorin virallinen arvio on, että siirryttäessä alustalta toiselle sovelluskoodista kyetään uudelleenkäyttämään 60–90 %. PhoneGapin tavoin myös Appcelerator Titanium pohjaa avoimeen lähdekoodiin. [25.]

Käyttötarkoitukseltaan Titanium ja PhoneGap kuulostavat identtisiltä; molempia hyödynnetään luotaessa mobiilisovelluksia, joiden tulisi olla helppoja kääntää tarpeen tullen myös muille alustoille. Metodit, joilla tämä lopputulos saavutetaan, ovat kuitenkin pohjimmiltaan erilaiset. PhoneGap pyrkii ”*write once, run everywhere*” -ajattelutapaan, jossa sama koodi voidaan hyödyntää sellaisenaan eri alustoille tekemättä muutoksia. Tämä toteutuu siten, että alustasta riippumatta WebView-komponentit tulkitsevat samankaltaisesti HTML:n, CSS:n ja JavaScriptin yhdistelmää, josta PhoneGap-sovellukset muodostuvat. Titanium pyrkii menemään lähemmäs natiivisovellusta ja kääntääkin JavaScriptilla kirjoitetut sovelluksensa natiivikielille. Tämä tarkoittaa, ettei identtisellä koodilla saadakaan sovellusta, joka on täysin alustariippumaton, mutta Titaniumin kehitysympäristö on optimoitu juuri niiden pienien muutosten tekemiseen, joita eri päätelaitteet vaativat. Titaniumin JavaScript-API pyrkii tarjoamaan mahdollisimman helpon pääsyn kaikkiin elementteihin, joita natiivisovelluskehittäjä kykenisi käyttämään. [26.]

Ylimääräisen vaivan vastapainona on, että Titaniumin sovellukset ovat suorituskyvyltään PhoneGapin sovelluksia nopeampia. Titaniumin lähestymistapa luo myös rajoituksia, joita PhoneGapissa ei ole, kuten erilaisten kirjastojen käytön. PhoneGap pyrkii pi-

tämään oman hierarkiansa kohtuullisen rajoittamattomana, jolloin kehittäjä saa valita työkalunsa melko vapaasti yleisimpien web-teknologioiden joukosta, mutta Titaniumin kanssa työskennellessä kehittäjä joutuu nojaamaan enemmän Titaniumin omaan kehitysympäristöön. Käytännössä Titanium on sovelluskehys, joka pakottaa oppimaan ja kehittämään asiat omalla tavallaan siinä missä PhoneGap sallii vapaammin erilaisten web-tekniikoiden käärimisen sovellukseksi. PhoneGapin ja Titaniumin lähestymistapojen erot ovat hyvä esimerkki siitä, miten hybridisovellusten kehitys- ja työnkulkua ei ole vielä standardoitu erityisen pitkälle tai parasta tapaa löydetty. [27.]

3.2.3 Supersonic-sovelluskehys

Supersonic on suomalaisen AppGyverin kehittämä PhoneGapiin pohjaava kehitysalusta. AppGyver koki olemassa olevien kääntäjien ja kehitysympäristöjen olevan liian puutteellisia sen omiin tarpeisiin, joten monien muiden tavoin se pyrkii ”korjaamaan PhoneGapin virheet”. Tärkeimmäksi tavoitteekseen se listaa hybridisovellusten kehittämisen sille tasolle, että niitä ei ole mahdollista erottaa natiivisovelluksista. Toisena tavoitteena on virtaviivaistaa ja etenkin nopeuttaa hybridisovellusten kehittämistä.

AppGyverin alkuperäinen kehitysympäristö tunnettiin nimellä Steroids, mutta se on sittemmin kehittynyt muotoon Supersonic. Steroidsin perintö on yhä mukana Steroids CLI:nä, joka tukee Supersonic-kehitystä tarjoamalla mahdollisuuden luoda, viedä ja rakentaa projekteja komentorivin kautta. Käytännössä Steroids vastaa projektin kommunikoinnista joko pilven tai simulaattorin kanssa. [28.]

AppGyver itse luonnehtii Supersonicin koostuvan kolmesta komponentista: natiiviosista, Ionic HTML5-sovelluskehyksestä ja ”taikuudesta”, joka käytännössä viittaa helpotettuun dataintegrointiin. Supersonic kykenee käyttämään kaikkia PhoneGapin ominaisuuksia, mutta tarjoaa myös paljon valmiita ratkaisuja ja komponentteja, jolloin etenkin ulkoasun rakentaminen helpottuu huomattavasti. Supersonic kuvailee itseään ”rajapinta-agnostiseksi”, eli se ei pakota kehittäjää käyttämään tiettyä JavaScript-sovelluskehystä. Tämä ei kuitenkaan toteudu täydellisesti ilman vaivannäköä, sillä Supersonic on kehitetty vahvasti AngularJS mielessä. Mikäli Angularin haluaa korvata kilpailevalla kehyksellä, joutuu tekemään jonkin verran omaa optimointia tai yhä käyttämään Angularia rajatusti. AngularJS:n vahvassa tukemisessa on hyvätkin puolensa, kuten Angularin hyvin laaja käyttäjäkunta ja AngularJS kehitystiimin Supersonicille an-

tama tuki. Muun muassa ngAnimaten kehittäjä Matias Niemelä on avoimesti tukenut ja ylistänyt Supersonicin toimivuutta. [28.]

Kehitystyötä helpottavista ominaisuuksista kenties oleellisin on AppGyver Scanner, jolla kehittäjä pääsee testaamaan sovellustaan oikealla laitteella kesken kehitysprosessin kääntämättä projektia valmiiksi sovellukseksi. Yhdistämällä projektin AppGyverin pilveen saa QR-koodin, jonka voi skannata AppGyver Scanner -mobiilisovelluksella. Scanner päivittää sovellusversionsa muutamien sekuntien välein, jolloin lopetettuaan kirjoittamisen Supersonicin puolella näkee tekemänsä muutokset käytännössä välittömästi.

AppGyverin kokonaisuus on käytännössä ilmainen; perusratkaisu ei maksa mitään ja yritys myös lupaa asianlaidan pysyvän nykytilassa. Maksullisia lisäpalveluita on ja varmasti tulee lisää. Nykyisen mallin suurin heikkous on se, että Supersonic-sovellusta ei kykene rakentamaan ilman Steroidsia ja AppGyverin omaa pilvipalvelua. Uhkakuva ei lupauksista takia ole palvelun muuttuminen maksulliseksi, vaan yrityksen mahdollinen kaatuminen. Tällöin pilvipalvelun jatkuminen olisi vähintäänkin epätodennäköistä, ellei AppGyver tarjoa kyseisessä tilanteessa avoimesti lähdekoodiaan.

Toinen huomattava heikkous on kehittäjien määrä. AppGyver ei ole julkaissut virallisia lukuja käyttäjäkunnastaan, mutta yksi tapa vertailla eri kehitysympäristöjen suosiota on tarkkailla, kuinka paljon niistä keskustellaan alan foorumeilla. Uusien kehitysympäristöjen haaste on aina se, ettei välttämättä löydä vertaisilta vastauksia ongelmiinsa. PhoneGapin laaja käyttäjäkunta kykenee auttamaan PhoneGapin ominaisuuksissa, mutta Supersonicin omien ominaisuuksien suhteen ei välttämättä heti löydä ratkaisua ongelmia kohdatessa. AppGyver toki pyrkii panostamaan omaan yhteisöönsä kehittämällä kattavat ohjeet ja tarjoamalla keskustelualustan, mutta lopulta silkka käyttäjämäärä auttaisi paljon enemmän.

Kolmantena heikkoutena on kattavuus; tällä hetkellä Supersonic kääntää sovelluksia vain Androidille ja iOS:lle. Nämä ovat suosituimmat alustat, mutta monella yrityksellä on myös tarvetta tarjota sovelluksensa samalla Windows Phoneillekin. Hybridisovelluksista puhuttaessa usein suureksi valtiksi nostetaan helppo kääntäminen kaikille alustoille, joten Windows Phonen (toistaiseksi) väliin jättäminen on omalla tavallaan alan omia periaatteita vastaan sotivaa.

3.3 Hybridi verrattuna natiiviin

Sovelluksen kehitysprosessia on haastavaa vertailla objektiivisesti, sillä eri ihmiset työskentelevät hyvin eri vauhdilla ja saman ihmisen tuskin voidaan olettaa olevan yhtä kokenut kehittäjä sekä web- että natiivipuolella. Näin ollen on vaikeaa todeta hybridisovelluskehityksen olevan yksiselitteisesti nopeampaa kuin natiivisovelluskehityksen. Tästä on kuitenkin viitteitä: Java on vahvasti tyypitetty kieli ja vaatii paljon tekstiä ollakseen ilmaisukykyinen. HTML5 on anteeksiantavampi virheiden suhteen ja ilmaisukykyinen pienelläkin määrällä koodirivejä.

Kuvassa 1 on esimerkki HTML5:n ja Javan luontaisesta erosta syntaksissa ja kielen byrokraattisuudessa. Perinteisesti ensimmäinen ohjelma, joka kirjoitetaan kieltä opetellessa, on ”Hello World”, eli ohjelma, joka vain tulostaa näytölle kyseisen tekstin. Kuva 1 esittelee ohjelman kummallakin kielellä kirjoitettuna.

```html
<!-- HTML5 HELLO WORLD ESIMERKKI -->

<!DOCTYPE html>

  <head>
    <title>Hello World</title>
  <head>

  <body>
    <p>Hello World!</p>
  </body>

</html>
```

```java
// ANDROID/JAVA HELLO WORLD ESIMERKKI

package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.widget.TextView;

public class MainActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      TextView text = new TextView(this);
      text.setText("Hello World!");
      setContentView(text);

   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.activity_main, menu);
      return true;
   }
}
```

Kuva 1. ”Hello World” esimerkki kirjoitettuna vasemmalla HTML5:llä ja oikealla Javalla.

Näin ollen todetaan HTML5:n olevan useimmissa tapauksissa nopeampi kieli kirjoittaa. Ainoa viittaus tilanteisiin, joissa natiivisovellus olisi nopeampi kehittää, ovat tapaukset, joissa sovellus sisältää ensisijaisesti vain laajan joukon toimintoja mitkä käyttävät hy-

väkseen laitteen fyysisiä ominaisuuksia. Nämä samat ominaisuudet voidaan kehittää myös hybridinä, mutta tämä tarkkaan määritelty skenaario antaa edun natiivikehittäjälle.

Toinen yritystä varmasti kiinnostava tekijä on kuitenkin projektin kulut. Koska ei ole suoraan mahdollista sanoa hybridikehityksen olevan aina nopeampaa, oletetaan projektin vaativan yhtä paljon työtunteja ohjelmoijalta riippumatta siitä, onko kyseessä natiivi- vai hybridisovellus. Näissä työtunneissa kuitenkin on ero, sillä keskimäärin Android-sovelluskehittäjä saa suurempaa palkkaa kuin web-sovelluskehittäjä. Kuukausitasolla ero saattaa olla jopa tuhansia euroja työpaikasta riippuen. Palkkaero pohjaa todennäköisimmin eroon käytettyjen kielten oppimiskäyrissä ja vaadituissa taustaopinnoissa. Edellä läpi käytyjen kehitysaikaan liittyvien seikkojen nojalla on kuitenkin epätodennäköistä, että sekä hybridi- että natiivisovellus veisivät identtisen ajan – etu olisi hybridillä.

Projektin tasolla ero saattaa tarkoittaa hybridin olevan melkein puolet halvempi. Aina vain enemmän tehokkuutta ja tuottoa vaativassa yritysmaailmassa tämä saattaa kuulostaa houkuttelevalta vaihtoehdolta yritykselle. Myös sovelluksen elinkäyrää ajatellen hybridisovellus tarjoaa etuja tuotteen päivitettävyyden nopeuteen. Tilanne palaakin kysymykseen siitä, missä tilanteissa kompromissi on eduksi. [29.]

Hybridisovellusten suorituskyky on yleensä kuitenkin tärkein tekijä loppukäyttäjän kannalta. WebView-komponentit ovat todetusti hitaampia kuin natiivikielestä käännetyt sovelluskomponentit [30]. Ongelma voidaan kääntäjästä riippuen osittain ohittaa erilaisin oikotein, kuten esilataamalla eri näkymien sisällöt esimerkiksi toisen näkymän taustalla. Tämä ei kuitenkaan vielä takaa, että eri päätelaitteiden WebView’n versiot tuottavat identtisen lopputuloksen. Suoritusero vanhempien WebView-versioiden ja tuoreempien JavaScript-renderöintimoottorien välillä voi olla huomattava. Yksi ratkaisu tämän eroavaisuuden poistamiseen on pakata sovellukseen mukaan Crosswalk Project, joka yhdenmukaistaa WebView’n versiot tuoreeseen Chromiumiin [31].

Yhtenäisen standardin puute tai selvästi suosituimman ratkaisun puute itsessään kertoo, että hybridin suorituskyvyssä riittää yhä parantamisen varaa. Ongelmat pitää siis yhä ratkoa tapauskohtaisesti, mikäli niihin edes on sopivaa ratkaisua.

Heikoimmassa tapauksessa WebView'n ongelmia ei ratkota, vaan lopputulos on suorituskyvyltään heikko sovellus. Esimerkiksi Apple on omassa sovelluskaupassaan ryhtynyt jopa laadunvalvonnassa hylkäämään hybridisovelluksia, joiden suorituskyky ei ole "tarpeeksi natiivin kaltainen". Androidin Play Store ei ole ryhtynyt vastaaviin toimenpiteisiin, mutta heikkojen sovellusten osuuden kasvu valikoimasta ei myöskään tee hyvää hybridisovellusten maineelle. [32.]

Taulukko 1. Android-sovellusten vertailu jaettuna natiiveihin ja hybrideihin.

Sovellus	**Natiivi**	**Hybridi**
Facebook	**X**	
Linkedin	**X**	
Imgur	**X**	
Wikipedia		**X**
Netflix		**X**
Evernote		**X**

Suurten yritysten valinta sovellusteknologiakseen antaa myös vertailupohjaa natiiville ja hybridille. Kenties tärkein puheenvuoro hybridin suorituskykyä vastaan oli Facebookin perustajan Mark Zuckerbergin ilmoitus Facebookin hybridisovelluksen ongelmista. Zuckerberg kuvaili natiivin korvaamista hybridillä suurimmaksi virheeksi, mitä Facebook on yrityksenä tehnyt. Puheenvuoro on usein mainittu esimerkkinä siitä, ettei hybridi soveltuisikaan laajoille sovelluksille. Taulukon 1 esittämä vertailu eri mobiilisovellusten rakennustavasta osoittaa, etteivät hybridisovellukset kuitenkaan ole vain pienille sovelluksille, joiden suorituskykyä on helpompi optimoida. Nyrkkisääntönä natiivi on kuitenkin ylivoimainen vaihtoehto paljon grafiikkaa tai raskaita elementtejä sisältävälle sovellukselle. [33.]

Hybridisovelluksen edut:

- edullisempi
- nopeampi kehittää
- olemassa olevan osaamisen siirtäminen sovelluskehitykseen
- sama ohjelmointikieli laitteesta riippumatta, pienempi riippuvaisuus päätelaitteen tyypistä
- sisällön päivitettävyys ilman sovelluksen päivittämistä

Natiivisovelluksen edut:

- parempi suorituskyky, korostuu raskaammissa sovelluksissa
- parempi pääsy laitteen ominaisuuksiin
- ei rajoituksia

[34]

4 Sovelluskehitysprojekti

4.1 Vaatimukset

Insinöörityössä esimerkisovelluksen kehitykselle asetettiin seuraavat vaatimukset:

- Mediastandissa toimiva esimerkkisovellus
- mahdollisuus offline-tilaan
- päivitettävyys asiakkaan toimesta.

Kaikkiin osa-alueisiin kuuluu myös tärkeänä tekijänä arvio tekniikan kannattavuudesta verrattuna vanhaan työskentelytapaan. Aiempi metodi oli puhdas web-sovellus, joka lukitaan Android-tablettiin kolmannen osapuolen kioskiohjelman kautta. Kioskiohjelmat tarjoavat hyödyllisiä ominaisuuksia tilanteisiin, joissa halutaan rajoittaa loppukäyttäjän kykyä päästä käsiksi tiettyihin laitteen ominaisuuksiin tai sisältöihin, mutta toisaalta kioskit ovat luonteeltaan myös hyvin rajoittavia kehittäjän itsensäkin suuntaan. Laitteen omia ominaisuuksia hyödyntävät palvelukutsut saatetaan kieltää vääränä liikenteenä siinä missä oikeakin kielletty liikenne, kuten muiden sovellusten avaaminen tai kioskin sulkeminen. Kun hybridisovellus on itsenäinen sovellus, ei vastaavia rajoitteita oletusarvoisesti pitäisi tulla vastaan.

4.2 Sovellukseen valitut tekniikat

Yrityksen koon ja sovelluskehityksen uudistamiseen tarkoitettujen resurssien vähäisyyden huomioiden esimerkkisovellukseen valitut teknologiat edustavat ilmaisia ja usein vapaan lähdekoodin ratkaisuja. Yksi painopiste on web-sovellusten tekemiseen vaadittavien taitojen siirrettävyys hybridipuolelle, joten valitut tekniikat edustavat laajasti käytössä olevia ratkaisuja. Pienen yrityksen tuskin kannattaa panostaa tekniikkaan, joka joko monimutkaisuutensa tai pienen käyttäjämääränsä takia saattaa aiheuttaa ongelmia ja hidasteita kehitysvaiheessa. Laajempi käyttäjämäärä tarkoittaa myös laajempaa joukkoa kehittäjiä, jotka keskustelevat ongelmistaan ja ratkaisuistaan vapaasti nähtävillä keskustelualueilla.

Pienen yrityksen myös täytyy kyetä toimimaan nopeasti kilpaillakseen suurempien kanssa, joten teknologioita valitessa muuten samankaltaisten kilpailijoiden kesken on valittu yksilöt, joiden oppimiskäyrä on matalampi.

4.2.1 Hybridisovelluskehys

Hybridisovelluskehykseksi ja kääntäjäksi valittiin suomalaisen AppGyverin Supersonic/Steroids. Tarkoituksenmukaista projektissa on valita alusta, joka on sekä ilmainen että pohjaa teknologiaan, jolla on jo paljon osaajia. Supersonic on rakennettu PhoneGapin päälle ja tukee kaikkia Cordovan vanhoja ominaisuuksia, joten useimpiin kohdattaviin ongelmiin löytyy todennäköisesti apua tai valmiita ratkaisuja. Supersonicin laaja Supersonic UI tarjoa myös ratkaisuja ”natiivin kaltaisen” ulkoasun luomiseen vailla

CSS- ja JavaScript -kiertoteitä, jotka perinteisesti ovat hävinneet suorituskyvyssä oikealle. Mainospuheiden suhteen tulee aina olla kriittinen, mutta Supersonic ainakin asettaa korkealle tavoitteissaan suorituskyvyn tuomisen natiivisovellusten tasolle – eli tarkoituksena on ratkaista yksi hybridisovellusten suurimmista heikkouksista.

Supersonic valittiin seuraavista syistä:

- taaksepäin yhteensopivuus kaikkien PhoneGapin ominaisuuksien kanssa
- phoneGapia väitetysti parempi suorituskyky
- painopiste sovelluskehittäjän työn helpottamisessa
- valmiit kirjastot ja elementit
- suomalaisen yrityksen tukeminen mahdollisuuksien mukaan.

4.2.2 JavaScript-ohjelmistokehys

HTML5-kehitystä tukemaan valittiin AngularJS-ohjelmistokehys. Se on Googlen kehittämä kehys, joka erikoistuu etenkin SPA-sovelluksiin, mutta soveltuu myös muihin ratkaisuihin. Angular pyrkii myös tuomaan JavaScriptia lähemmäs natiivisovelluskielien sovellusrakenteita käyttämällä MVVC-arkkitehtuuria. Monimutkaisemmissa ja laajemmissa sovelluksissa on hyvien käytäntöjen mukaista pyrkiä soveltamaan suunnittelumallia, joka takaa ohjelmistologiikalle paremman alustan. MVVC on yksi suosittu ratkaisu etenkin mobiilikehityksen puolella [35, s. 278–279]. AngularJS:n pohjimmainen tavoite on täydentää HTML:ää itseään ja muuttaa sen luonne staattisesta dynaamiseksi. AngularJS ei vain lisää kirjastoa täynnä JS-funktioita, vaan käytännössä lisää HTML:n ”sanastoa” uusilla ilmaisuilla. Näistä tyypillisempiä ovat kaksisuuntaiset datan sitomiset, eli käytännössä muuttujat, jotka kommunikoivat reaaliajassa mallin ja näkymän välillä. Tämä tekee AngularJS:stä erityisen hyvän muun muassa datan näyttämiseen.

Googlen kehittämänä se myös nauttii suurta tukea tekijöiden taholta. Verrattaessa esimerkiksi AngularJS-keskustelua StackOverFlow’ssa tulee vastaan moninkertainen määrä tuloksia muihin suuriin JavaScript-ohjelmistokehyksiin verrattuna. Taulukko 2

esittelee tilastotietoa sovelluskehysten suosiosta perustuen juuri tämänkaltaisiin epäortodoksisiin mittareihin. W3Schools on myös valinnut AngularJS:n yhdeksi opetuspakettinsa aiheeksi, mikä puhuu paljon Angularin puolesta, kun ottaa huomioon W3Schoolsin aiemmin ottaneen vain yhden JS-kirjaston valikoimaansa, eli jQueryn.

Taulukko 2. Suosituimpien sovelluskehysten tilastotietoa [36].

	AngularJS	BackBone.js	Ember.js
Tähtiä Githubissa	40 200	18 800	14 100
Kolmannen osapuolen moduuleja	1488	256	1155
Kysymyksiä StackOverFlow'ssa	104 000	18 200	15 700
Youtube-hakutuloksia	93 000	10 600	9 100
GitHub-osallistujia tai -projekteja	96	265	501
Avoimia ongelmia	922	13	413

AngularJS valittiin kolmesta syystä:

- laaja tuki suuren yhteisön ansiosta, helppo löytää ratkaisut ongelmiin
- valmis laaja yhteensopivuus Supersonicin kanssa
- valmis kokemuspohja AngularJS-web-sovellusten parista.

4.2.3 Bootstrap-sovelluskehys

CSS3-sovelluskehykseksi valittiin Twitterin kehittämä Bootstrap. Se on tällä hetkellä suosituin CSS-sovelluskehys. Bootstrapin perimmäinen ajatus on helpottaa responsiivisten ja mobiiliystävällisten sivujen ja web-sovellusten kehittämistä. Käytännössä Bootstrap tarjoaa paljon valmiita komponentteja ja luokkia, jotka piti ennen kehittää kömpelömmin erilaisin kiertotein. Bootstrapin tarkoituksena on myös CSS3:n periaatteiden mukaisesti tarjota laaja yhteensopivuus eri selaimille ja alustoille.

Yksi responsiivista kehitystä nopeuttava tekijä Bootstrapissa on sen oma ruudukkojärjestelmä, jossa sivun elementit voidaan jakaa painotettuihin riveihin. Sivu tai sovellus voidaan niiden avulla jakaa riveihin, jotka taas on jaettu eri painotuksille jaettuihin sarakkeisiin. Ajatuksena tämä kuulostaa ennen HTML5:n aikaa käytössä olleilta taulukkotyylin sivustoilta, mutta taulukkotyylin sivupohjat olivat pohjimmiltaan erittäin staattisia

ja mukautumattomia verrattuna Bootstrapin ruudukkoihin. Käytännössä ruudukot kykenevät reagoimaan näytön kokoon, jolloin esimerkiksi elementit asemoituvat käsketysti vierekkäin suurilla näytöillä, mutta pienellä näytöllä taas asettuvat vertikaalisesti.

Bootstrap valittiin seuraavista syistä:

- suosituin CSS-sovelluskehys
- erinomainen soveltuvuus mobiililaitteisiin eli myös hybridisovelluksiin
- todettu yhteensopivuus AngularJS:n kanssa
- valmis kokemuspohja Bootstrapin edellisistä versioista.

4.2.4 PHP-palvelinpää

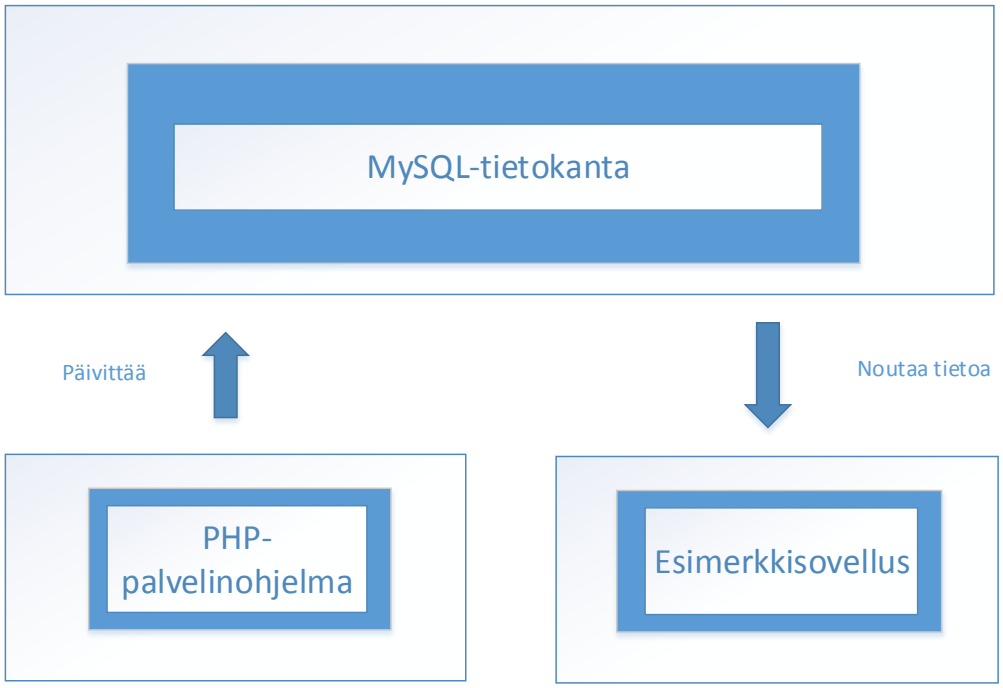
Palvelinpuolelle valittiin PHP:n ja MySQL:n yhdistelmä, jossa PHP toimii palvelinpuolen kielenä ja MySQL tietokantakielenä. Syy teknologioiden valintaan oli niiden suosio ja kehittäjän vanha kokemus kielten kanssa. Web-sovelluskehittäjät, jotka eivät tunne vähintään perusteita PHP:n ja MySQL:n kanssa kehittämisestä ovat harvassa, vaikka uudemmat enemmän JavaScriptiin pohjaavat ratkaisut ovat viime vuosina kasvattaneet suosiotaan.

PHP on vapaan lähdekoodin palvelinpäässä ajettava skriptikieli. Erona JavaScriptiin ja HTML:ään on koodin suoritusvaihe: loppukäyttäjä ei näe PHP:ta, vaan näkyvä lopputulos on jo käännetty PHP:sta asiakaspään kieleksi, kuten HTML:ksi. MySQL on vapaan lähdekoodin relaatiotietokantaratkaisu, jossa tieto tallennetaan toisiinsa suhteessa oleviin tauluihin. Kummankin laaja ja pitkäikäinen käyttäjäkunta tarjoaa paljon apua ongelmatilanteissa. [37, s. 3–7.]

Käytännössä kaikki laajat sovellukset käyttävät tietokantaa johonkin. Tietokannassa voidaan säilyttää joko staattista tai muuttuvaa dataa, jonka käsittely on helpompaa palvelimen päässä. Tämä data voi olla muun muassa käyttäjätietoja tai jotain, mitä käyttäjä kykenee hakemaan, esimerkiksi verkkokaupan kaikki tuotteet. PHP tarjoaa poikkeavan hyvän valmiuden eri tietokantoihin yhdistämiseen, joten PHP:n ja MySQL:n yhteistyö on jo oletusarvoisesti hyvin tuettu [37, s. 233–236]. Tietokantaoperaatioiden siirtä-

minen asiakkaan päästä palvelinpäähän luo myös tietoturvaa, sillä itse ohjelma voi vain rajatusti vaikuttaa siihen, mitä asiakaspäässä tapahtuu. Tietokantojen muokkauksen ja hakutoimintojen tapahtuminen etäpalvelimella vaikeuttaa mahdollisesti pahaa tahtovien loppukäyttäjien mahdollisuuksia päästä käsiksi tai muokkaamaan tietoa, joka ei kuulu heille [37, s. 531–534].

Esimerkkisovelluksessa palvelinpään mukaan tuomisella pyritään ratkomaan päivitettävyys asiakkaan toimesta. Käyttäjäystävällisempi ratkaisu mobiilisovelluksen sisältöön vaikuttamiseen on muokkaustoimenpiteiden siirtäminen itse sovelluksen ulkopuolelle. Käytännössä sisältö muokattaisiin PHP-käyttöliittymästä, joka vaihtaa MySQL-tietokannassa olevia tietoja. Itse esimerkkisovellus pääsee tietoihin käsiksi rajapinnan kautta, joka tarjoaa sovelluksen kaipaamat tiedot. Kuva 2 esittää yksinkertaisen version tästä toimintamallista.



Kuva 2. Tietokannan, palvelinsovelluksen ja esimerkkisovelluksen toiminta.

Tietokannan suhteen toinen harkinnassa oleva vaihtoehto oli perinteisestä MySQL-relaatiotietokannasta siirtyminen NoSQL-tietokantaan. Web-kehityksen trendejä seuratessa törmää aina vain useammin MongoDB:n kaltaisiin NoSQL-tietokantoihin vaihtoehtona perinteisille relaatiotietokannoille. Suurin ero MySQL:n ja MongoDB:n välillä on juuri tapa, millä tietoa käsitellään ja missä muodossa se tallennetaan. MySQL edustaa relaatiotietokantoja, joten tieto tallennetaan tauluihin, joilla on tietty suhde muuhun tie-

toon. Käytännössä tiedolle tehdään valmiit lokerot, joihin vain kyseinen tieto sopii. MongoDB:n NoSQL edustaa mallia, jossa tiedolla ei tarvitse olla valmista pohjaa tai lokeroa, vaan tietoja voidaan tallentaa toisistaan poikkeavina. Käytännössä lopputulos muistuttaa hyvin paljon joukkoa JSON-olioita, joita MonboDB itse kutsuu nimellä BSON. [38.]

Syy perinteisen MySQL:n valitsemiseen MongoDB:n tai muun kasvavan, uuden tietokantajärjestelmän sijasta mainittiin jo luvun 4 alussa: yksi esimerkkisovelluksen tarkoituksista on arvioida, kuinka hyvin vanhaa osaamista voidaan siirtää hybridikehitykseen. Asiakasyrityksellä ei ole käytössä MongoDB:ta vanhastaan, mutta sillä on MySQL-tietokannat. Toinen syy on sovelluksen mittakaava: MongoDB on vahvimmillaan tapauksissa, joissa tietoa syötetään tai muutetaan usein ja myös pohjimmaista skeemaa pitää kyetä muokkaamaan. Esimerkkisovellus ei suoranaisesti sovi tähän kuvaukseen, sillä tietorakenne ei muutu kaiken aikaa eikä oletuksena kaipaa muutoksia skeemaan. [39.]

4.2.5 Offline-elementit

Yksi esimerkkisovelluksen suunnittelussa vaadituista tekijöistä oli mahdollisuus toimia offline-tilassa. Käytännössä tämä tarkoittaa, että sovelluksen toiminta ei ole kaiken aikaa riippuvainen internetyhteydestä. Yksinkertainen ratkaisu olisi pakata kaikki sovelluksen tarvitsema tieto laitteeseen itseensä, jolloin ulkoisia yhteyksiä ei tarvita. Tämä kuitenkaan ei ole erityisen käyttäjäystävällinen ratkaisu sovelluksen päivittämisen tai muokkaamisen kannalta, sillä lopputulos on auttamatta staattinen ja päivittäessä pitäisi päästä aina käsiksi päätelaitteeseen.

Hybridisovellusten yksi tavoite on vähentää itse sovelluspakettien tarvetta päivittää itseään. Käyttäjän on helppoa olla kriittinen sovellusta kohtaan, joka varaa ajan kuluessa aina vain enemmän tilaa laitteen muistista. Osasyy tähän on natiivisovellusten taipumus päivittää itseään päästyään kommunikoimaan sovelluskaupan kanssa, joka ilmoittaa uuden version olevan saatavilla. On jopa melko tavallista, että käyttäjät estävät sovelluskauppaa toimimasta ollenkaan ilman käyttäjän lupaa, ettei laitteen muisti ala hitaasti täyttyä sovelluspäivityksistä, mitä käyttäjä ei kaipaa. Hybridisovelluksia on pyritty markkinoimaan ajatuksella, että itse sovellusta ei tarvitse päivittää, vaan vain ulkoista sisältöä, jonka perusteella sovellus vaihtaa ulkoasuaan. Tyypillinen esimerkki on hybridisovellus, joka päästyään internetyhteyteen lataa XML- tai JSON-tiedoston,

joka sisältää sen tarvitsemat tekstikappaleet tai ulkoasumääritelmät. Kysymykseksi tietenkin muodostuu, miten kuvatulla tavalla internetyhteydestä riippuvainen sovellus saadaan offline-sopivaksi.

Ratkaisua lähdettiin etsimään hybridisovellusten periaatteiden mukaisesti web-teknologioiden puolelta mobiililaitteiden omien ominaisuuksien sijasta. Esimerkkisovellukseen valittiin HTML5:n tuoma Cache manifest ratkaisemaan tarve offline-elementeille. Cache manifest toimii sivun tai sovelluksen asiakirjana, jonne listataan elementit, joiden tulee olla saatavilla offline-tilassa. Kuvassa 3 on yksinkertainen esimerkki manifestista. Tällöin listatut tiedostot latautuvat laitteen muistiin ja sovellus käyttää jatkossa niitä, eikä pyri joka kerta lataamaan tiedostoja uudestaan. Ainoa tiedosto, jonka sovellus pyrkii aina käynnistyessään lataamaan, on itse manifesti. Tämän tarkoituksena on tarkistaa, onko manifesti muuttunut edellisestä kerrasta. Mikäli muutoksia on tapahtunut, sovellus pyrkii lataamaan listatut tiedostot uudestaan. Mikäli muutoksia ei ole tapahtunut, sovellus jatkaa vanhoilla tiedostoilla. Vastaavasti mikäli sovellus ei pääse käsiksi manifestiin esimerkiksi internetyhteyden puutteen vuoksi, otetaan käyttöön manifestin edellinen tallennettu versio. [40, s. 131–136.]

```
1  CACHE MANIFEST
2  # Päivitetty 29.10.2015
3
4  CACHE:
5  index.html
6  css/tyyli.css
7  img/kuva.png
8  js/main.js
9
10
11 NETWORK:
12 *
13
14 FALLBACK:
15 offline.html
```

Kuva 3. Yksinkertainen cache manifest, joka kertoo offlineen tallennettavat tiedostot.

Cache manifestissa on paljon huomioon otettavia pieniä asioita, mutta suurin riskitekijä ovat mahdolliset toimivuusongelmat hybridisovellusten WebView-komponenttien kanssa. Tarpeen tullen offline-elementtien osuutta esimerkkisovelluksessa pyritään asettamaan pienempään rooliin tai korvaamaan muilla HTML5-teknologioilla, kuten LocalStoragella, joka sallii noin 10 megatavun tallentamisen laitteen omaan muistiin JavaScriptin kautta. Ideaalinen tilanne olisi, että hybridisovellus rakentaa itsensä ladatuista elementeistä, mutta heikommassa tapauksessa offline-toimivuus voidaan myös rajata vain tiettyyn osaan sovellusta, joka esimerkiksi täyttää WebView'n laitteen ulkopuolisesta, valmiista www-sivusta, jossa cache manifestin toimivuus on vielä todennäköisempää.

Cache manifest on kuitenkin pohjimmiltaan abstraktio jo olemassa oleville toimenpiteille, jotka vaatisivat skriptauskielien käyttöä HTML:n sijasta. Näin ollen on myös todennäköistä, että cache manifestin paljastuessa toimimattomaksi vastaava teknologia kyetään toistamaan AngularJS:n ja LocalStoragen avulla. Tämä tarkoittaisi enemmän kehitystä ja omaa kirjoittamista, sillä pienistä erikoisuuksistaan huolimatta cache manifest on loppujen lopuksi hyvin pitkälle automatisoitu teknologia, jonka käyttäminen vaatii vain suunnittelua ja huolellisen manifestin laatimisen. Tämän korvaaminen JavaScriptilla edellyttäisi etenkin laajempaa lisätestausta.

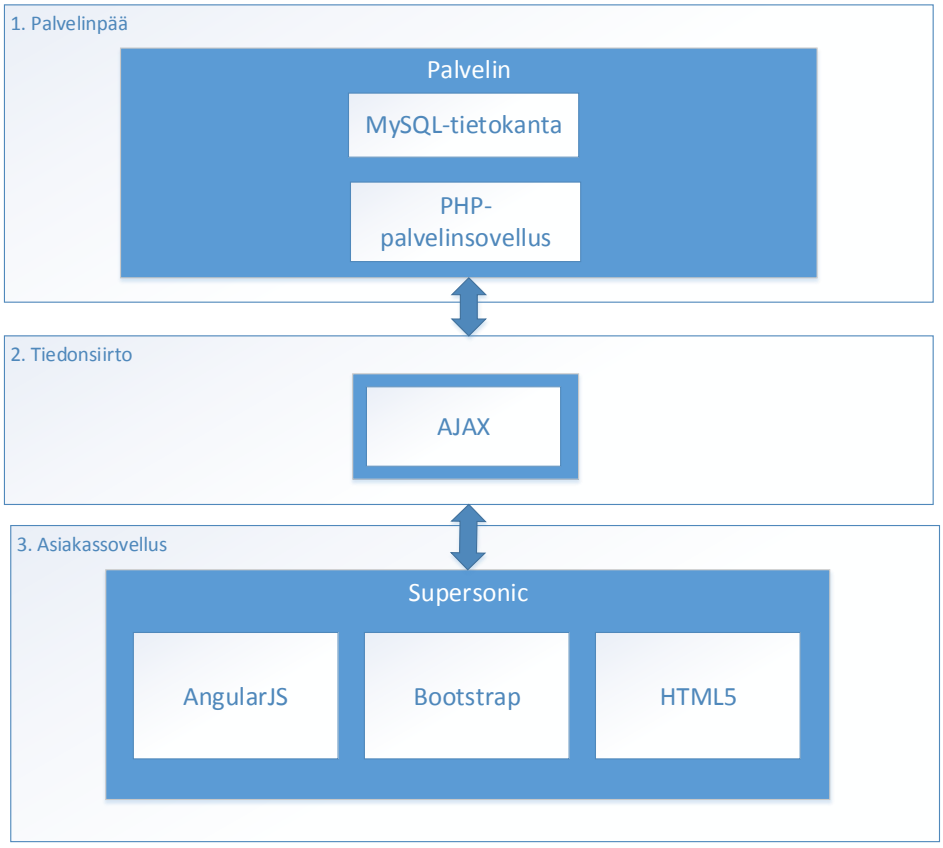
4.3 Sovelluksen rakenne

Sovellukseen suunniteltiin toteutettavaksi kaksi päänäkymää, jotka edustavat asiakasyrityksen Mediastand-konseptin yleisimpiä käyttötapoja. Ensimmäinen on niin sanottu ”Digital Signage” -osio, joka esittää pääosin staattista tietoa inforuudun tavoin. Kyseessä saattaisi olla vaikka tapahtuman vaihtuva ohjelma tai ravintolan päivittäin vaihtuva lounaslista. Osion on tarkoitus toimia tarpeen tullen offline-tilassa, ja sisällön tulee olla omistajan muokattavissa ilman pääsyä itse laitteelle.

Toinen osio edustaa palautekyselyä, jota voi soveltaa esimerkiksi rekisterin keräämiseen. Yrityksen käytäntöjen mukaisesti kyselyjen vastaukset tulee tallentaa palvelimelle, jolloin asiakas voi seurata rekisterin kasvua ilman pääsyä päätelaitteelle. Myös tämän osion tulee toimia ympäristössä, jossa internetyhteys saattaa olla ajoittain poikki,

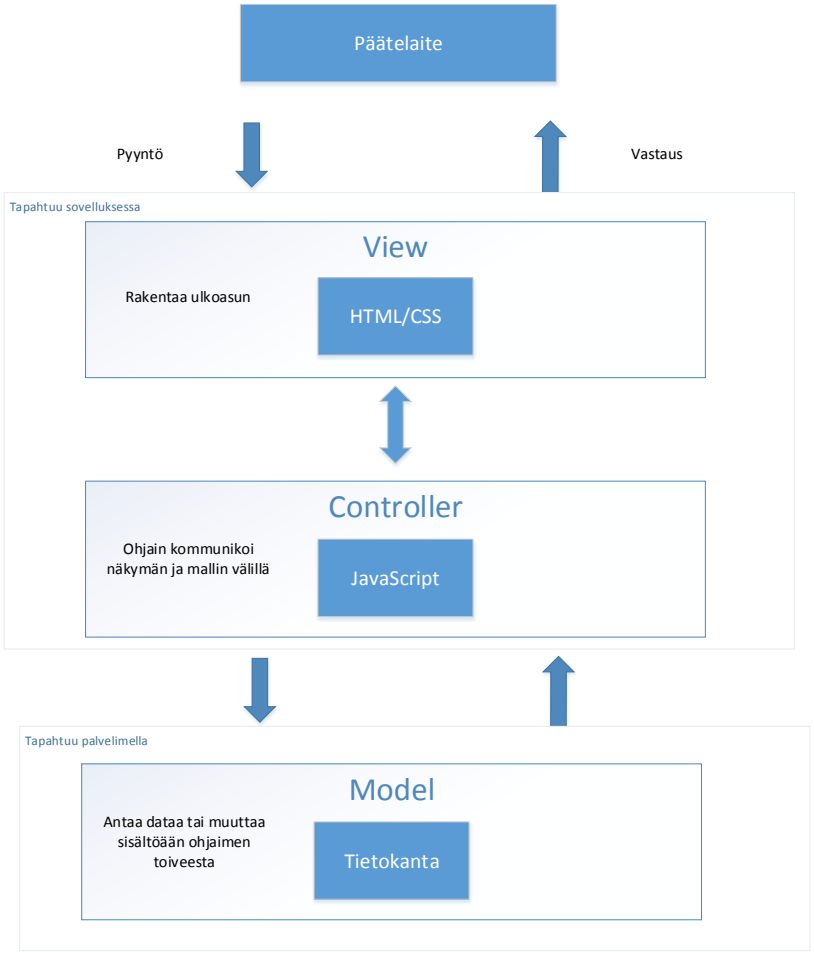
joten suunnittelun pitää huomioida mahdollisuus tallentaa tieto väliaikaissijaintiin, ennen kuin se voidaan lähettää loppusijoituspaikkaansa palvelimelle.

Kuva 4 esittää sovelluksen suunnitellun rakenteen korkealla abstraktiotasolla. Ensimmäinen taso kuvaa palvelinpuolen toimintaa, jossa tietoa noudetaan ja varastoidaan MySQL-tietokantaan PHP:llä toteutetun sovelluksen kautta. Toinen taso kuvaa ensimmäisen ja kolmannen tason välistä asynkronista Ajaxilla tapahtuvaa tiedonsiirtoa. Kolmas taso on asiakassovellus, joka koostuu Supersoniciin käärityistä HTML5-sovelluskehyksistä ja kirjastoista.



Kuva 4. Esimerkkisovelluksen rakenne.

Kuva 5 taas kuvaa sovelluksen toimintalogiikkaa MVC-periaatteiden mukaisesti.



Kuva 5. Esimerkkisovelluksen toimintalogiikka MVC-tyylillä.

4.4 Sovelluskehitys

Sovelluskehitysprojekti aloitettiin laajalla Supersoniciin tutustumisella. AppGyver pyrkii tarjoamaan paljon apua ja ohjeita Supersonicin ja Steroidsin käyttöönottoon. Käytännössä yksinkertaiset toiminnot on hyvin dokumentoitu sekä tekstinä että opetusvideoina, mutta tarkempi läpikäynti nopeasti paljastaa Supersonicin vaativan kohtuullisen paljon esitietoja teknologioista, joita ei mainospuheissa mainita. Periaatteen tasolla Supersonic vaatii vain Steroidsin ja vapaavalintaisen tekstieditorin, mutta Steroids puolestaan vaatii muun muassa Node.js:n, Bowerin, Gitin ja tietyn version Windowsista. Omituista kyllä AppGyver on valinnut tukea versiota Windows 8, mutta sovelluskehitysprojektissa käyttöön saatiin ensin Windows 7 ja myöhemmin Windows 10. AppGyver ilmoittaa tuotteidensa todennäköisesti toimivan muissakin versioissa, mutta ei itse lupaa tuotetukea niihin. Käytännössä ratkaisu on ulkoistettu yhteisöllem ja esimerkiksi

Windows 7:n käyttäjiä kehotetaan ongelmissa kääntymään AppGyverin yhteisön puoleen virallisen tahon sijasta. Periaatteessa sovelluskehitys ei vaadi huomattavaa CLI:n tai Node.js:n käyttöä, mutta Steroids on selvästi rakenneltu tätä yhteyttä ajatellen.

Pohjimmiltaan Supersoniciin tutustuminen on kuitenkin hyvin samanlaista kuin mihin tahansa uuteen teknologiaan tutustuminen. Dokumentaatio on tarpeeksi kattava vastatakseen useimpiin kysymyksiin, mutta aktiivisten kehittäjien määrä ei ole erityisen näkyvä. AppGyverin omassa yhteisössä käydään keskustelua ongelmista ja ratkaisuista, mutta vastausten määrä on usein vähäinen. Laaja yhteisö helpottaa aina oppimista, mutta Supersoniciin liittyvissä ongelmissa huomaa avun löytämisen olevan haastavampaa kuin esimerkiksi AngularJS:n ongelmissa.

AngularJS:n ja Supersonicin yhteistyö havaittiin toimivaksi. AngularJS tarjosi lisäosineen ratkaisut useimpiin kehitysprosessin ongelmiin. Kosketuseleisiin liittyvät toiminnot saatiin kehitettyä ilman HammerJS:n kaltaisia erillisiä kehyksiä lisäämällä Angularin oma lisämoduuli ngTouch. Cache manifest havaittiin lopulta huonosti toimivaksi vaihtoehdoksi Supersonicin WebView-rakenteen kanssa, joten ratkaisua lähdettiin hakemaan LocalStoragen kautta. Angularin lisäosista löytyi myös localStoragen käyttöä helpottamaan ngStorage, joka tarjoaa abstraktion HTML5:n localStoragen käyttöön ja vähentää varastointiin liittyvän koodin määrää melkein puolella. Käytännössä ngStorage hoitaa sovelluslogiikassa sellaisen tiedon tallentamisen laitteen muistiin, jota tarvitaan joko internetyhteyden katketessa tai myöhemmin palatessa. Juuri tämänkaltaisia etuja hybridisovelluskehityksessä haetaan natiiviin verrattuna. Javan ja Android SDK:n byrokraattinen luonne usein hidastaa pienienkin lisäominaisuuksien lisäämistä. Kuva 6 havainnollistaa eron tiedon tallentamisen eroista Javalla ja AngularJS:llä.

```
//MUISTIIN TALLENTAMINEN JAVALLA

String filename = "myfile";
String string = "Tallennettava tieto";
FileOutputStream outputStream;

try {
  outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
  outputStream.write(string.getBytes());
  outputStream.close();
} catch (Exception e) {
  e.printStackTrace();
}
```

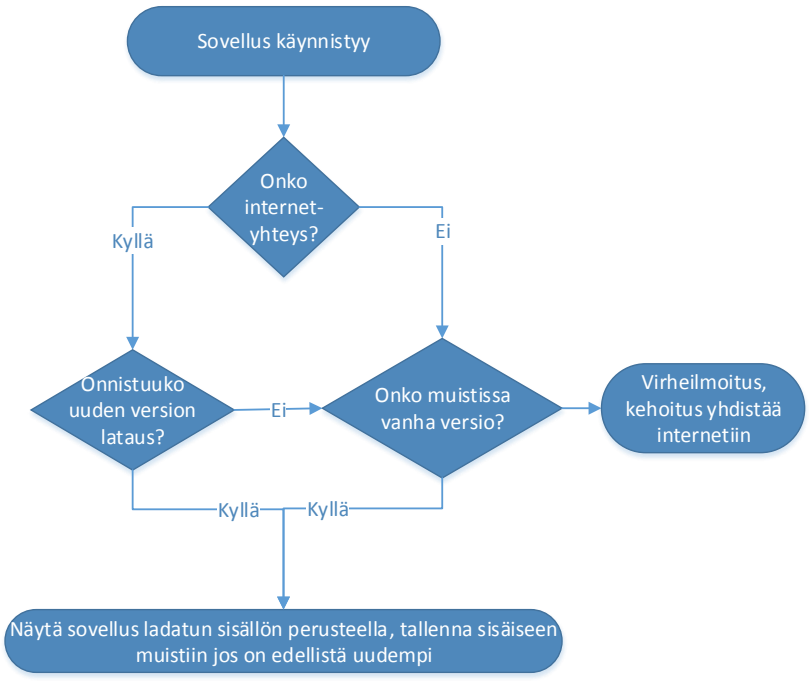
```
//TALLENNUS NGSTORAGELLA


var data = "tallennettava tieto";
$localStorage.data = data;
```

Kuva 6. Tiedon tallentaminen ensin Javalla ja sitten JavaScriptilla ngStoragea hyödyntäen.

Yksi osasyy Supersonicin valitsemiseen oli sen pohjalla lepäävä Cordova-teknologia. PhoneGapin suuri suosio takaa paljon valmista materiaalia ja ratkaisuja ongelmatilanteisiin, joten sovelluskehitysprosessin kannalta on myös loogista pyrkiä hyödyntämään PhoneGapin ominaisuuksia. Yksi esimerkki hyödynnetyistä PhoneGapin ominaisuuksista oli Cordovan connection-rajapinta, joka tarjoaa pääsyn päätelaitteen mobiilidata- ja WI-FI-yhteyksiin. Connectionin alla on myös muutamia hyödyllisiä tapahtumia, joita voi koodillisesti seurata. Androidin natiivisovelluksissa internetyhteyden muuttumisen tai menettämisen seuraaminen vaatii kohtuullisen suuren määrän vaikeaselkoista koodia tehtävän laadun huomioiden, mutta saman seurannan toistaminen hybridipuolella on verrattain yksinkertainen tehtävä. Cordovan ”offline-event” ja ”online-event” tarjoavat jo muutamalla koodirivillä yksinkertaisen tavan kuunnella muutoksia sovelluksen päätelaitteen yhteystilassa. Tietoa laitteen yhteyden tilasta käytettiin offline-ominaisuuksia kehitettäessä, sillä yhteyden tila sanelee suuren osan toimintalogiikasta yhteystilanteissa.

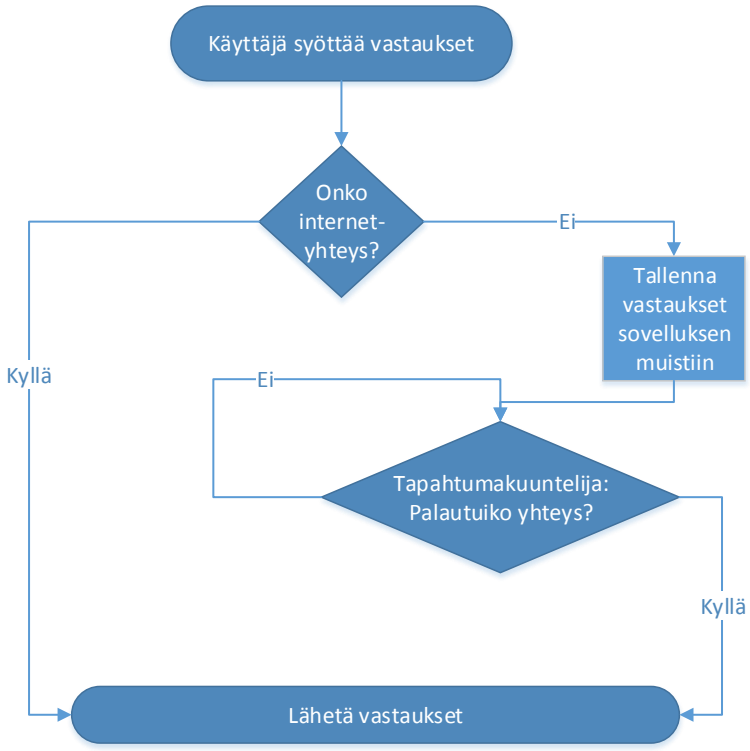
Kuvan 7 vuokaavio esittää algoritmista yksinkertaisen version, joka rakentaa pääosin ulkoisesta sisällöstä koostuvan näkymän. Esimerkkisovelluksessa tätä käytetään niin sanotun ”digitaalisen kyltityksen” (digital signage) näkymän toteuttamiseen.



Kuva 7. Vuokaavio näkymän toiminnasta offline-tilanteissa.

Sovellus lataa ylläpitäjän tietokantaan syöttämät otsikot, tekstin ja tarpeelliset infotiedot muodostaen niistä näkymän loppukäyttäjälle. Ulkoiseen, päivitettävään tietokantaan pohjaava ratkaisu on kuitenkin riippuvainen internetyhteydestä, joten sovellus pyrkii tarjoamaan myös offline-mahdollisuuden tallentamalla tarpeelliset tiedot omaan muistiinsa aina saadessaan tuoreemman version sisällöstä.

Kuvan 8 vuokaavio esittää toisen version algoritmista. Tämä näkymä ei tarvitse palvelimelta ladattua tietoa esittääkseen omaa sisältöään, mutta se välittää käyttäjän syöttämää tietoa palvelimelle. Ei ole siis tarpeen säilyttää vanhoja versioita näkymän ulkoasusta, vaan tarvitaan ratkaisu käyttäjän syöttämän datan tallentamiseen tilanteita varten, jolloin ei kyetäkään muodostamaan yhteyttä palvelimeen. Käytännön toteutus tallentaa loppukäyttäjän syöttämät vastaukset JavaScript-olioksi, jotka tallennetaan taulukkomaisena tietorakenteena ngStoragen kautta localStorageen. Sovellus seuraa oletusarvoisesti internetyhteyden tilanmuutoksia tapahtumakuuntelijalla, joten sovelluksen palaaminen internetiin on helppo havaita. Tämän tapahtuessa algoritmi syöttää muistiin tallennetut vastaukset järjestyksessä palvelimelle. Lomakkeita käsitellessä on myös useimmiten olennaista tarkistaa tiedon oikeellisuus, eli minkätyyppistä dataa kenttiin on syötetty. Tyypillisesti tämä tarkastetaan vertailemalla syötettyä tekstiä tiettyjä parametrejä vastaan, esimerkiksi sähköpostin tapauksessa tarkistetaan, löytyykö syötteesta ensin tekstiä, sitten @-merkki, jonka jälkeen taas tekstiä ja lopulta piste ennen muutamaa viimeistä merkkiä. Prosessin helpottamiseksi etsittiin apua AngularJS:n valmiista lomakkeiden validointiin liittyvistä ominaisuuksista.



Kuva 8. Vuokaavio vastausten lähettämisestä palvelimelle.

PhoneGap on pohjimmiltaan jo tarpeeksi laaja kattamaan lähes kaikki mobiilikohtaiset ominaisuudet, joihin käsiksi pääsy luetaan usein natiivisovellusten eduksi. PhoneGapia vaivanneet suorituskykyongelmat ovat kuitenkin tekijä, joka on suurimpia huolenaiheita hybridisovelluskehittäjille. Supersonic pyrkii optimoimaan Cordovan kykyjä entisestään, mutta käytännössä AppGyver ei tarjoa tarkkaa dataa suorituskyvyn paremmuudesta. Ennen esimerkkisovelluksen tekemistä pyrin testaamaan Supersonicin WebView-komponenttien suorituskykyä erilaisella sovelluksella, joka käytännössä toimi tuoteluettelona yritykselle. Tuoteluettelo toimi perinteisellä web-periaatteella, jossa on vain yksi tuotesivu, jonka ylimääräisellä id-parametrillä täytetään oikean tuotteen tiedot valmiiseen sivupohjaan. Tämä kokeilu opetti paljon Supersonicin navigaation toiminnasta ja siitä, miten sen sisäiset WebView't toimivat. Supersonic käyttää "layer stackia" (tasopino), eli uuden WebView'n ilmaantuessa se pinotaan vanhan päälle, jolloin menneet näkymät muodostavat korttipakkamaisen rakenteen. Tämä poikkeaa itsessään perinteisestä web-selaimen toiminnasta, joten tilanne voi helposti aiheuttaa hämmennystä kehittäjälle, joka ei ole päätynyt lukemaan oikeaa osiota dokumentaatiosta. Käyttäjä saattaa esimerkiksi huomaamattaan aina luoda uuden näkymän sen sijaan, että navi-

goisi jo aiemmin avattuun näkymään. Tällaisessa tapauksessa näkymä ei olisi siis ainutlaatuinen, vaan enemmänkin luokkamainen ohjeistus, jonka pohjalta voi tuottaa monia instansseja samasta näkymästä.

Projektin kannalta harmilliseksi havaittiin, että uuden WebView'n lataaminen oli selvästi hitaampaa kuin uuden näkymän avaaminen perinteisessä natiivisovelluksessa. Riippuu sovelluksen rakenteesta, onko tämä ongelma. Jos sovelluksessa navigoidaan jatkuvasti näkymästä toiseen, kumuloituvat käyttäjän silmissä pienet hidastukset suuremmaksi suorituskykyongelmaksi. Jos näkymä vaihtuu hyvin harvoin, ei ongelmaa huomaa niin hyvin. Yksi Supersonicin ratkaisu tähän on esiladata halutut WebView't käsketyssä vaiheessa – esimerkiksi sovelluksen käynnistyessä. Tämä luo halutut WebView't valmiiksi tasopinoon, jolloin navigointi näkymien välillä on välittömän nopeaa. Haasteeksi muodostuvat sovellukset, joissa on hyvin paljon näkymiä – tulisiko kymmeniä tai satoja näkymiä ladata valmiiksi?

Kuvassa 9 on esimerkki WebView'n esilatauksesta sekä koodillisesti että sovelluksen oman rakennetiedoston kautta. Koodillisesti esilataus on käytännössä näkymän käynnistämistä etukäteen JavaScriptilla tai Coffeescriptilla, rakennetiedoston kautta taas näkymien listausta Coffeescriptistä koostuvaan rakennetiedostoon.

```
//SUPERSONIC ESILATAUS KOODILLISESTI

var showView1 = new supersonic.ui.View(
  location: "esimerkki#ladattavsivu",
  id: "esilataus1"});
showView1.start();
```

```
# ESILATAUS SOVELLUKSEN ASETUKSISTA

module.exports =


  preloads: [
    {
      id: "ladattavasivu1"
      location: "esimerkki#ladattavsivu"
    }
    {
      id: "ladattavasivu2"
      location: "esimerkki#ladattavsivu2"
    }

  ]
```

Kuva 9. Näkymien esilataus Supersonicissa.

Sovelluskehityksen suurimpia hidasteita oli sovelluksen osien testaaminen AppGyver Scannerin kautta. Noin puolet projektiin upotetusta ajasta Scanner toimi moitteetta ja

muutosten tarkistaminen oli nopeaa, mutta AppGyver tuntuu kärsivän selittämättömistä yhteysongelmista valitettavan usein. Aina ajoittain laite, joka sai moitteetta yhteyden AppGyverin kautta, ei saanutkaan yhteyttä ollenkaan. Supersonicin rajatun käyttäjämäärän vuoksi avun hakeminen keskustelupalstoilta ei myöskään johda erikoisen pitkälle. Mikäli yhteyttä ei saada, testaus tulee suorittaa hitaamman kaavan kautta lataamalla sovellus AppGyverin pilveen ja lataamalla se sieltä ”jaa ystävälle” -toiminnon kautta. Tämä askel hidastaa sovelluksen pienten muutosten testaamisen sekunneista jopa noin minuuttiin.

4.5 Lopputulos

Esimerkkisovellus toteutettiin pääosin suunnitelman mukaisesti. Kaikkia esisuunniteltuja teknologioita, kuten cache manifestia, ei kyetty käyttämään täysin oletetulla tavalla, mutta toiminnallinen lopputulos on vastaava. Merkittävin muutos suunnitelmassa oli offline-ominaisuuksien rakennemuutos Cache manifestista ngStorageen ja omaan koodiin. Käytännössä tämä ei ollut ongelma, sillä Cache manifest pohjimmiltaan käyttää LocalStoragea vastaavalla tavalla, mutta tarjoaa nopeat oikotiet samaan lopputulokseen. Cache manifest ei kuitenkaan ollut lopulta täysin yhteensopiva Supersonicin rakenteen kanssa esimerkkisovelluksen tapauksessa.

Sovellus ei itsessään ole käyttövalmis tuote, vaan enemmänkin tutkimus teknologioiden soveltuvuudesta asiakasyrityksen käyttöön.

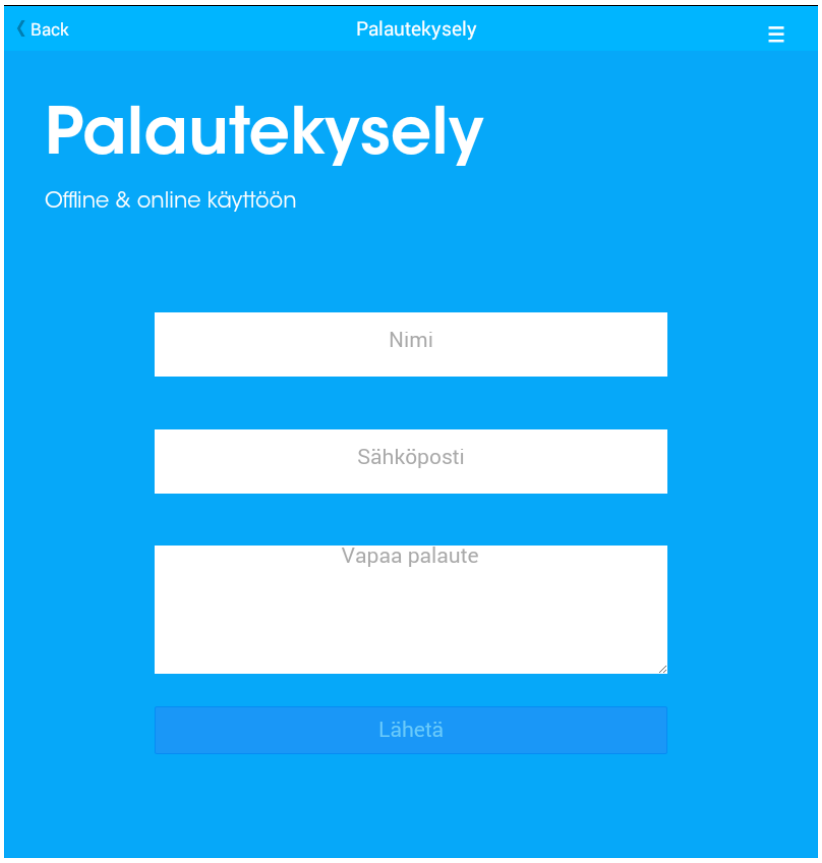
Kuvassa 10 on sovelluksen aloitusnäkymä, jonka pääasiallinen tarkoitus on toimia aloitussivuna ja tarjota tiivistelmä sovelluksen rakenteesta ja toiminnasta. Samalla se tarjoaa katsauksen sovelluksessa käytettyihin web-teknologioihin, sovelluskehyksiin ja kirjastoihin.

Kuva 10. Esimerkkisovelluksen aloitusnäkymä.

Digital Signage -osio muodostaa itsensä palvelimelta noudetun datan perusteella. Kuvassa 11 oleva tieto on luonnollisesti vain satunnaista lorem ipsum -tekstiä, jolla testattiin sisällön päivittymistä.

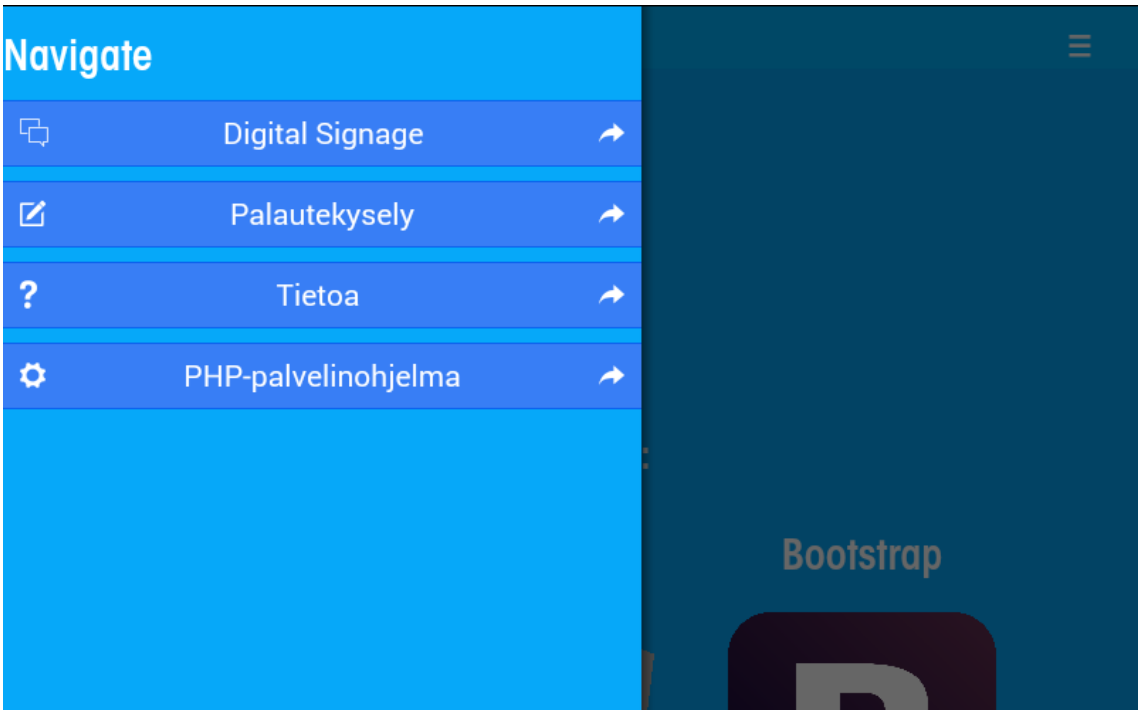
Kuva 11. Esimerkkisovelluksen Digital Signage –näkymä.

Palautekysely kerää yksinkertaisen palautteen ja joko lähettää sen välittömästi palvelimelle tai tallentaa sen omaan välimuistiinsa yhteyden puutteen vuoksi. Syötetyn tiedon tyypin oikeellisuus tarkistetaan AngularJS:n valmiilla form-työkaluilla. Tämä ilmenee kuvassa 12 olevan ”Lähetä”-painikkeen harmautena, kun tekstikenttiä ei ole vielä täytetty oikein.



Kuva 12. Esimerkkisovelluksen palautekysely.

Kuvassa 13 on esimerkki näkymien välisestä navigoinnista. Vetovalikko aukeaa joko raahaamalla sovelluksen vasemmasta reunasta kosketuseleellä tai klikkaamalla yläpalkin ☰-kuvaketta. Valikkoon on testaussyistä upotettu myös linkki PHP-palvelinohjelmaan, jonka kautta on mahdollista muokata tietokannan tietoja. Sivu ei ole rakennettu osaksi sovellusta, vaan yksinkertaisesti upotettu ulkoiseen sivuun viittaavana WebView-komponenttina linkin takaa. Supersonicin moninäkymäisen navigoinnin mukaisesti navigointipainikkeet vain muuttavat tasopinon järjestystä ja tuovat halutun näkymän päällimmäiseksi.



Kuva 13. Esimerkkisovelluksen navigointivalikko.

4.6 Havainnot

Tärkeimpiä esimerkkisovellusta kehitettäessä tutkittuja asioita olivat vanhan web-osaamisen siirtäminen hybridisovelluskehyksiin, hybridisovellusten edut asiakasyritykselle ja offline-ominaisuuksien kehittäminen.

Web-osaamisen siirtäminen hybridisovelluskehitykseen kuului myönteisiin havaintoihin. Kohtuullisella perehtymisellä web-kehittäjän on mahdollista ryhtyä kehittämään sovelluksia, joilla on mahdollista kilpailla natiivisovellusten kanssa. Suurin perehtyminen ei välttämättä olekaan koodillisesti uuden opetteleminen, vaan sovelluksen rakenteen tarkka suunnittelu suorituskyvyn kannalta. Oletusarvoisesti Java-kehittäjän tai web-kehittäjän ei tarvitse kiinnittää paljoa huomiota sellaisiin asioihin kuin näkymien määrään navigointia suunnitellessa, mutta WebView-komponenttien kanssa tämäkin tulee ottaa huomioon. Suorituskykyä ratkaisevat lisäykset, kuten Crosswalk, voivat myös tuoda tullessaan muita ongelmia, kuten sovelluksen suuri koko. Crosswalk yhtenäistää WebView'n toiminnan Chromiumiksi, mutta se myös lisää sovelluksen kokoa 22 megatavulla [41]. Tämä ei ole kaikissa tapauksissa ongelma, mutta yksinkertaisen sovelluksen koon kasvaminen jopa kymmenkertaiseksi ei myöskään ole toivottava lopputulos.

Kuluttajakäyttöön tulevassa sovelluksessa merkitys on suurempi, sillä käyttäjät ovat aina vain kriittisempiä sen suhteen, minkä sallivat viedä tilaa laitteestaan.

Web-teknologiat kurovat kuitenkin umpeen suorituskyvyllistä välimatkaansa Javaan, joten uskon WebView'n myös kehittyvän paljon tulevina vuosina. Androidin tapauksessa jo vanhempien versioiden hidas poistuminen käytöstä itsessään helpottaa tilannetta, kun vanhempia laitteita ei tarvitse tukea. Android 4.4 KitKat -versiosta eteenpäin WebView't ovat perustuneet Chromiumiin, ja nykytilassa 62,4 % Androidin käyttäjistä käyttää joko 4.4-versiota tai uudempaa [42].

Asiakasyrityksen hyötyä arvioidessa havainnot eivät välttämättä ole yhtä myönteisiä. Optimaalinen lopputulos varmastikin olisi ollut, että hybridisovelluskehitys on sopiva vaihtoehto korvaamaan sekä vanhat web-sovellukset että ajoittain vaadittavat natiivisovellukset, mutta käytännössä en lähtisi korvaamaan kumpaakin. Asiakasyrityksen koon ja useimpien projektien luonteen huomioiden web-sovelluksista ei kannata luopua, sillä hybridisovellusten hyödyt niihin nähden eivät olleet tarpeeksi suuria. Tämä johtuu osaltaan siitä, että tuoreimmat JavaScript-sovelluskehykset tuntuvat jo web-sovelluksissa tarjoavan hyvin paljon ratkaisuja mobiililaitteille. Sen sijaan hybridisovellus voi asiakasyrityksen sovelluskehitystä tarkastellessa olla hyvä vaihtoehto korvaamaan harvinaisemmat tarpeet kehittää natiivisovellus Androidille. Asiakasyritys on silloin tällöin päätynyt kehittämään natiivisovelluksen joko omiin tai asiakkaan tarpeisiinsa, mutta tämä on edellyttänyt lisätyövoimaa ja -resursseja. Hybridisovellus olisi useimmissa näistä tapauksista riittänyt vastaamaan tarpeisiin, mutta kuluissa ja ajassa olisi säästetty.

Offline-ominaisuuksien kehittäminen onnistui, mutta ratkaisussa on kritisoitavaa. Android-sovellusten offline-käytettävyyteen ei ole valmista patenttiratkaisua, vaan toimivat metodit riippuvat paljon sovelluksen luonteesta. Asiakasyrityksen sovellukset ovat usein pieniä, joten LocalStorage palvelee tarpeeksi hyvin sen tarpeita. Samaa ei kuitenkaan voisi sanoa suuresta sovelluksesta, tai vaikkapa sovelluksesta, jonka tulisi tallentaa paljon kuvia muistiinsa. LocalStorage sallii loppujen lopuksi kohtuullisen vähän tilaa sovellusta kohden, joten muun kuin tekstin tallentaminen saattaisi olla ongelmallista. Tutkimuksen havainnot eivät siis välttämättä ole sovellettavissa suuremmille yrityksille, mutta positiivista on, että offline-ominaisuudet alkavat hitaasti olla HTML5:n myötä osa tavallisempaa sovelluskehitystä eikä vain harvinaisuus tai erikoisvaatimus.

5 Yhteenveto

Insinöörityön tarkoituksena oli tutkia hybridisovelluskehitystä Android-laitteille ja teknologian soveltuvuutta asiakasyrityksen Mediastand-konseptiin. Työssä perehdyttiin erilaisiin tapoihin tuottaa hybridisovellus ja valittiin asiakasyritykselle sopiva teknologia, jonka pohjalta tehtiin esimerkkisovellus.

Android-sovellukset jaetaan perinteisesti natiivisovelluksiin ja web-sovelluksiin. Natiivisovellukset ovat järjestelmän omalla kielellä kehitettyjä laitteen muistiin asennettavia sovelluksia, web-sovellukset taas web-kielillä kirjoitettuja selainmoottorissa suoritettavia ohjelmia. Hybridisovellus on näiden kahden välinen silta, jossa pyritään saavuttamaan etua sovelluksen kehityskaareen kirjoittamalla sovellus web-kielillä, mutta kääntämällä se natiivisovelluksen kaltaiseksi asennettavaksi sovellukseksi. Web-tekniikoiden hyödyntäminen ja JavaScriptin käyttäminen Javan sijasta luo uusia mahdollisuuksia sovelluksen tekemiseen eri sovelluskehysten tyylillä.

Hybridisovellukset havaittiin potentiaaliseksi kilpailijaksi natiivisovelluskehitykselle sekä ajankäytön että kustannusten suhteen. Vastaavaa kilpailuetua ei kuitenkaan havaittu verratessa tilanteisiin, joissa web-sovellus on riittävä ratkaisu. Monet hybriditeknologiat perustuvat avoimeen lähdekoodiin ja web-kehittäjän taidot olivat kohtuullisella vaivalla siirrettävissä hybridipuolelle, joten yrityksen on mahdollista kokeilla teknologiaa sijoittamatta projektiin huomattavia resursseja. Suurin säästö koituisi tilanteissa, joissa sovellus on tarpeen julkaista usealle alustalle hyödyntäen samaa lähdekoodia, mutta tämän työn asiakasyritys kehittää nykytilassa vain Android-pohjaisia ratkaisuja.

Lähteet

1 Oehlman, Damon, Blanc, Sebastian. 2011. Pro Android Web Apps - Develop for Android Using HTML5, CSS3 & JavaScript. New York: Springer Science+Business Media.

2 Apple developer - Swift. Verkkodokumentti. Apple. <https://developer.apple.com/swift/ > Luettu 29.10.2015

3 About Android. Verkkodokumentti. Android. <http://developer.android.com/about/ >. Luettu 20.7.2015.

4 Getting started with developing for Windows Phone 8. 2014. Verkkodokumentti. Windows. <https://msdn.microsoft.com/en-us/library/windows/apps/ff402529> Luettu 29.10.2015.

5 Ghatol, Rohit, Patel, Yogesh. 2012. Beginning PhoneGap - Mobile Web Framework for JavaScript and HTML5. New York: Springer Science+Business Media.

6 Rogers, Rick, Lombardo, John, Mednieks, Ziguard & Meike, Blake. 2009. Android Application Development. USA: O'Reilly.

7 Learn about Java technology. Verkkodokumentti. Java. <https://java.com/en/about/> Luettu 20.7.2015.

8 Web Application Definition. Verkkodokumentti. TechTerms. <http://techterms.com/definition/web_application> Luettu 20.7.2015.

9 What is a hybrid mobile app. 2015. Verkkodokumentti. Telerik Developer Network. <http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/> Luettu 31.7.2015.

10 WebView Class Overview. Verkkodokumentti. Android. <<http://developer.android.com/reference/android/webkit/WebView.html> Luettu 6.11.2015.

11 Rudolph, Patrick. 2014. Hybrid mobile apps - providing a native experience with web technologies. Verkkodokumentti. <http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/> Luettu 1.9.2015.

12 HTML5 Introduction. Verkkodokumentti. W3Schools. <http://www.w3schools.com/html/html5_intro.asp/> Luettu 3.9.2015.

13 Gardner, Lyza Danger. 2012. Head first mobile web. Sebastopol, CA : O'Reilly

14 Gerchev, Ivalo. 2014. The 5 most popular frontend frameworks compared. Verkkodokumentti. <http://www.sitepoint.com/5-most-popular-frontend-frameworks-compared/> Luettu 10.9.2015.

15 Get Bootstrap. Verkkodokumentti. Twitter Bootstrap. <http://getbootstrap.com/> Luettu 24.9.2015.

16 Foundation - The most advanced responsive front-end framework in the world. Verkkodokumentti. Foundation. <http://foundation.zurb.com/> Luettu 24.9.2015.

17 Semantic UI - User interface is the language of the web. Verkkodokumentti. Semantic UI. <http://semantic-ui.com/> Luettu 24.9.2015.

18 Skeleton - A dead simple, responsive boilerplate. Verkkodokumentti. Skeleton. < http://getskeleton.com/> Luettu 24.9.2015.

19 McPeak, Jeremy, Wilton, Paul. 2015. Beginning JavaScript. Indianapolis: John Wiley & Sons.

20 HammerJS general. Verkkodokumentti. HammerJS. <http://hammerjs.github.io/api/> Luettu 16.9.2015.

21 A Framework for creating ambitious web applications. Verkkodokumentti. Ember. <http://emberjs.com> Luettu 16.9.2015.

22 JavaScript Frameworks: The Best 10 for Modern Web Apps. 2015. Verkkodokumentti. Noeticforce. <http://noeticforce.com/best-Javascript-frameworks-for-single-page-modern-web-applications> Luettu 20.9.2015.

23 About Phonegap. Verkkodokumentti. PhoneGap. <http://phonegap.com/about/> Luettu 16.9.2015.

24 The Future of PhoneGap – Will it ever reach native app quality. 2013 Verkkodokumentti. PhoneGap. http://blog.appgyver.com/heartbeat/steroids/the-future-of-phonegap> Luettu 16.9.2015.

25 Titanium. Verkkodokumentti. Appcelerator. <http://www.appcelerator.org/#titanium> Luettu 16.9.2015.

26 Whinnery, Kevin. 2012. Comparing Titanium and Phonegap. Verkkodokumentti. <http://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/> Luettu 16.9.2015.

27 Elkharashy, Muhammad. 2013. PhoneGap vs Titanium in few points. Verkkodokumentti. <http://badrit.com/blog/2013/4/10/phonegap-vs-titanium-in-few-points> Luettu 16.9.2015.

28 Supersonic. Verkkodokumentti. AppGyver. <http://www.appgyver.com/supersonic/> Luettu 16.9.2015.

29 Native vs Hybrid / PhoneGap App Development Comparison. Verkkodokumentti. Comentum. http://www.comentum.com/phonegap-vs-native-app-development.html> Luettu 7.10.2015.

30 Titanium vs Phonegap vs Native application development. 2012. Verkkodokumentti. Sapan Diwakar. http://www.sapandiwakar.in/api-research-study-iphone-and-android-applications/ > Luettu 7.10.2015.

31 Crosswalk project – build world class hybrid apps. Verkkodokumentti. Crosswalk Project. <https://crosswalk-project.org/ > Luettu 2.11.2015

32 Trice, Andrew. 2012. PhoneGap advice on dealing with Apple application rejections. Verkkodokumentti. <http://www.adobe.com/devnet/phonegap/articles/apple-application-rejections-and-phonegap-advice.html > Luettu 7.10.2015.

33 Facebook's Zuckerberg: 'The biggest mistake we've made as a company is betting on HTML5 over native. 2012. Verkkodokumentti. Venturebeat. <http://venturebeat.com/2012/09/11/facebooks-zuckerberg-the-biggest-mistake-weve-made-as-a-company-is-betting-on-html5-over-native/ > Luettu 7.10.2015.

34 Viswanathan, Priya. The Pros and Cons of Native Apps and Mobile Web Apps. Verkkodokumentti. <http://mobiledevices.about.com/od/additionalresources/qt/The-Pros-And-Cons-Of-Native-Apps-And-Mobile-Web-Apps.htm > Luettu 7.10.2015.

35 Holzner, Steve. 2006. Design Patterns for Dummies. USA: Wiley Publishing.

36 Shaked, Uri. AngularJS vs. Backbone.js vs. Ember.js. Verkkodokumentti. Airpair. <https://www.airpair.com/js/javascript-framework-comparison > Luettu 6.11.2015.

37 Converse, Tim, Park, Joyce, Morgan, Clarke. 2004. PHP5 and MySQL Bible. Indianapolis: Wiley Publishing.

38 The MongoDB 3.0 Manual. Verkkodokumentti. MongoDB. <https://docs.mongodb.org/manual/ > Luettu 29.10.2015.

39 Kaplan, Moshe. 2014. When to use MongoDB rather than MySQL. Verkkodokumentti. < https://dzone.com/articles/when-use-mongodb-rather-mysql> Luettu 29.10.2015.

40 Frain, Ben. 2012. Responsive web design with HTML5 and CSS3. Birmingham, UK: Packt Publishing.

41 AppGyver Tooling Guides – Android Crosswalk. Verkkodokumentti. AppGyver. <http://docs.appgyver.com/tooling/wrapper/android/android-crosswalk/> Luettu 2.11.2015.

42 Platform versions. 2015. Verkkodokumentti. Android Developers. <http://developer.android.com/about/dashboards/index.html/> Luettu 2.11.2015.