MOBIILISOVELLUKSEN TOTEUTUS WEB-TEKNIIKOIN

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietotekniikka

Riihimäki, kevät 2015

Taru Pyykkönen

HAMK
HÄMEEN AMMATTIKORKEAKOULU

RIIHIMÄKI
Tietotekniikka
Ohjelmistotekniikka

Tekijä Taru Pyykkönen **Vuosi** 2015

Työn nimi Mobiilisovelluksen toteutus web-tekniikoin

TIIVISTELMÄ

Opinnäyte perustuu työharjoittelun aikana tehtyyn projektityöhön. Projektin tarkoituksena oli kehittää mobiiliversio asiakasyrityksen pöytäkoneella toimivasta tuotekatalogiohjelmasta. Työ toteutettiin web-tekniikoita käyttämällä, ja ensisijainen kohdealusta oli Android-käyttöjärjestelmä.

Sovellus toteutettiin käyttämällä yleisimpiä web-tekniikoita, joihin luetaan merkkauskieli HTML5, ohjelmointikieli JavaScript ja sen ohjelmakirjasto jQuery sekä kuvauskieli CSS. Näiden käytössä on omat vahvuutensa ja heikkoutensa mobiiliympäristössä. Lisäksi käydään läpi yleisempien web-selainten välisiä eroja, sillä tällaiset sovellukset tarvitsevat ajoympäristökseen tekniikoita tukevan web-selaimen.

Alkuperäinen ohjelma vaikutti vahvasti tehtävän työn taustalla. Sieltä poimittiin oleellisimmat toiminnot ja ominaisuudet. Lisäksi alkuperäisen alustan ja uuden alustan välillä oli sovelluksen ominaisuuksien kannalta oleellisia eroja. Näitä olivat Flash-tuen puute, laitteiden näyttörajoitukset, sormen käyttö osoittimena hiiren sijaan sekä mobiilikäyttöjärjestelmien tiedostojärjestelmät.

Sovellusta räätälöitiin sekä toiminnoiltaan että ulkonäöltään kohdealustalle ja sovelluksen käyttäjille lähinnä oman koulutuksen myötä kerättyjen tietojen ja taitojen avulla. Yhdeksi keskeiseksi ongelmaksi kehittyi myös alustojen erilainen tiedostojärjestelmä ja se, millaisin keinoin se oli mahdollista ratkaista. Tehdyn työn tulokset olivat kuitenkin enimmäkseen positiivisia. Ohjelma toimi niin kuin suunniteltiin ja lisäksi toiminnot ja ulkonäkö palvelivat käyttötarkoitustaan. Kuitenkin sovellukseen jäi vielä jonkin verran jatkokehityksen varaa.

Lopuksi oli syytä arvioida myös vaihtoehtoisia ratkaisuja kuin käytetyt web-tekniikat. Tällaisia ratkaisuja ovat esimerkiksi alustan nativiisovellus, natiivisovellukseen käärittyWebView-näkymä, maksuttoman sovellusympäristö PhoneGapin käyttö sekä maksullinen Xamarin. Näiden pohtimiseen käytettiin ohjelmistojen virallisia dokumentaatioita.

Avainsanat Mobiili, HTML5, JavaScript, CSS

Sivut 34 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Degree Programme in Information Technology
Programming Technology

Author Taru Pyykkönen **Year** 2015

Subject of Bachelor's thesis Developing a mobile application on Web Technologies

ABSTRACT

This thesis is based on a project conducted during the internship of the author. The goal of this project was to develop a mobile version of the client company's already existing application on a desktop computer. The development work was performed on web technologies and the primary target platform was the Android operating system.

The basic web technologies include the markup language HTML5, the programming language JavaScript and its library jQuery and the style sheet language CSS. Using these technologies has some advantages and some disadvantages on the mobile platform. Since these applications need a web browser in order to work, the differences between the most popular browsers are also discussed in this thesis.

The original application provided the basic guidelines to the project. All the necessary functions and features came from there. In addition, there were some essential differences between the original and the new platform that needed to be taken into consideration. These differences were the lack for Flash support, the limited screen space, using a finger as the pointer and the file systems of the mobile operating systems.

The functions and the look were customized for the targeted platform and the intended users. This was achieved mostly by the skills and knowledge gathered during the education at HAMK. One of the most essential problems was the differing file system between the platforms and handling it. The results of the work were mostly positive. The application worked as planned and both the functions and the look applied well to the intended use. Yet, there is still some room left for possible future improvements.

It is reasonable to take a look at some alternative methods for developing the application. These include the native application, the WebView and using development environments such as PhoneGap and Xamarin. The discussion on these solutions is based on their official documentation.

Keywords Mobile, HTML5, JavaScript, CSS.

Pages 34 p.

SISÄLLYS

1 JOHDANTO

Osallistuin keväällä 2014 osana työharjoittelua projektiin, jonka tarkoituksena oli tehdä asiakasyrityksen pöytäkoneella toimivasta tuotekatalogista mobiiliversio. Tarve tälle ohjelmalle oli selkeä: Koska nykyisin mobiililaitteiden käyttö on kasvanut räjähdysmäisesti, on hyvinkin järkeenkäypää hyödyntää niitä myös työelämässä. Kevyemmän rakenteensa ansiosta mobiililaitetta on kannettavaa konetta helpompi kantaa sinne, missä sen käyttöä tarvitaan ja se on aina saatavilla.

Asiakasyrityksen edustaja halusi, että sovellus tehtäisiin web-tekniikoin ja ensisijaisesti Android-alustalle. Käyttäjäryhmänä sovelluksella olisi yrityksen myyjät, jotka käyttäisivät ohjelmaa tietyn tuotteen tietojen etsimiseen. Alkuperäisen ohjelman pohjalta mobiiliversiolle määritettiin vaaditut toiminnot ja mahdolliset lisäominaisuudet. Projektin alussa oli tärkeää selvittää, miten mobiilialusta eroaa pöytäkoneympäristöstä ja suunnitella, miten nuo eroavaisuudet otetaan huomioon toteutusvaiheessa. Opinnäytteeni pohjaa projektin aikana tehtyyn työhön lähinnä mobiilialustan näkökulmasta. Lisäksi se sisältää pohdintaa sovelluksen toimivuudesta ja vaihtoehtoisista toteutusmenetelmistä.

Opinnäytteeni sisältää myös havainnollistavia ruutukaappauksia toteutetusta sovelluksesta. On kuitenkin huomautettava, etteivät ne sisällä oikeaa tietoa varsinaisista tuotteista vaan kaikki kuvissa esiintyvät tiedot ovat tekaistuja.

2 TEKNIIKAT

Web-tekniikoilla tarkoitetaan niitä kooditekniikoita, joita hyödynnetään verkkosivuja ja -sovelluksia tehtäessä. Näitä ovat merkkauskieli HTML, ohjelmointikieli JavaScript sekä kuvauskieli CSS. Web-tekniikoissa kuten muissakin ohjelmaratkaisuissa on vahvuutensa ja heikkoutensa. Lisäksi web-tekniikoita käyttävät sovellukset tarvitsevat toimiakseen web-selaimen, joten myös niiden ominaisuudet vaikuttavat siihen, kuinka hyvin nämä sovellukset varsinaisessa käytössä toimivat. Toisaalta sovelluksessa voidaan käyttää myös koodikirjastoja, joiden toimivuus ei ole riippuvainen selaimen ominaisuuksista.

2.1 HTML5

HTML eli HyperText Markup Language on yleinen verkkosivujen tekemiseen tarkoitettu merkkauskieli. HTML5 viittaa sen uusimpaan standardiin, josta tehtiin virallinen W3C-suositus lokakuussa 2014 (W3C 2014). HTML5:n uudet ominaisuudet ovat toimineet jo pidemmän aikaa yhä useammassa webselaimessa, ja moni verkkosivu on jo siirtynyt sen mukaiseen käytäntöön. Esimerkiksi Googlen hakumoottorin pääsivu käyttää jo HTML5:n mukaista, lyhyempää doctypea.

Verkkosivu rakentuu puumaisesti HTML-elementeistä. Tällainen elementti koostuu alkutagista, sisällöstä ja sen sulkevasta lopputagista. Esimerkiksi tekstin kappaletta merkitsevä elementti “p” (paragraph) luotaisiin seuraavasti: Esimerkkikappale . Elementeillä on usein myös attribuutteja eli elementin ominaisuuksia, joilla sille voi antaa jotain olennaista lisätietoa. Yleisimpiä ominaisuuksia ovat elementin yksilöllistävä id-attribuutti tai sen luokan osoittava class-attribuutti. Tarkoitettu attribuutti lisättäisiin aiempaan esimerkkiin näin: Esimerkkikappale .

HTML5 käyttää aiemmasta HTML 4-versiosta poiketen paljon lyhyempää ja yksinkertaisempaa doctypea (kuva 1). Tämä onkin määrityksien oleellisin ero, sillä väärää doctypea käyttämällä sivulla ei voi käyttää uusimmassa määrityksessä olevia elementtejä.

```
HTML 4 Strict Doctype
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

HTML5 Doctype
<!DOCTYPE html>
```

Kuva 1. HTML 4 Strict-tyyppinen doctype on pidempi verrattuna HTML5:n uuteen doctypeen.

Toinen tärkein muutos on se, että HTML5 tuo käyttöön yhä enemmän rakenteellisia elementtejä, ikään kuin palikoita, joista sivut rakennetaan. Uusimpia elementtejä ovat esimerkiksi omat otsakeosan header- ja navigaatioosan navigation-tagit. Aiemmassa versiossa nämä osiot piti luoda erikseen omiin div-tageihinsa, jotka sitten yksilöllistettiin yleensä id-attribuutin kautta.

Lisäksi käyttöön tulee paljon uusia sisältöelementtejä, kuten multimedian elementit video ja audio, ja attribuutteja, esimerkiksi tekstinsyöttökenttään voidaan lisätä email-attribuutti, joka tarkistaa automaattisesti syötteen oikean muodon ilman erityistä lisäohjelmointia. Aiemmissa standardeissa syöte tarkistettiin ohjelmointikielien, kuten JavaScriptin, avulla.

HTML5 sisältää myös monenlaisia sovellusliittymiä, joita voidaan hyödyntää HTML-pohjaisissa sovelluksissa tai verkkosivuilla. Näistä uutuuksista hyötyvät erityisesti web-ohjelmoijat. Yksi tällainen sovellusliittymä on esimerkiksi Geolocation API -ohjelmarajapinta, jota voidaan käyttää osoittamaan käyttäjän senhetkinen olinpaikka.

Vastaavasti W3C:n HTML5:n tämänhetkisessä suositusehdotuksessa puhutaan lukuisista, vanhentuvista elementeistä ja ominaisuuksista, joita ei enää suositella käytettäväksi. Tällaisia määritteitä ovat esimerkiksi lukuisat websivun ulkonäköön vaikuttavat HTML-elementit, kuten font-tagi. Ne on lajiteltu vanhentuviksi, koska ne voidaan helpommin toteuttaa CSS-kuvauskielellä.

2.2 JavaScript ja jQuery

JavaScript on yleinen käyttäjänpuoleinen ohjelmointikieli, jolla verkkosivu saadaan elämään. Sen avulla HTML-sivuun voidaan lisätä erilaista toimintaa kuten interaktiivisuutta ja erilaisia visuaalisia tehosteita. JavaScript on myös se olennainen osa, jolla verkkosivu saadaan antamaan tietynlaista palautetta käyttäjälle sivun toimivuudesta, esimerkiksi sen avulla sivun napin voi laittaa välähtämään hiiren painalluksesta tai syöttämään virheviestin, kun käyttäjä on unohtanut lomakkeesta tarpeellisia tietoja.

JavaScript-koodia voidaan käyttää erittäin dynaamisesti verkkosivulla: Koodin seasta voi suoraan luoda ja muuttaa HTML-elementtejä tai niiden attribuutteja kutsumalla niitä joko elementin tagin tai sen yksilöllisen id-attribuutin kautta. Esimerkiksi, document.getElementById(“output”) etsii HTML-dokumentista sen elementin, jonka id-attribuutin arvo on “output”, jolloin vain tähän elementtiin voidaan tehdä tarvittavia muutoksia JavaScript-koodin avulla, esimerkiksi lisätä CSS-tyylejä.

Jotta verkkosivu näyttää JavaScript-ohjelmakoodilla toimivan sisällön käyttäjän laitteella, käyttäjän verkkoselaimen on ensiksi tuettava sitä ja toiseksi se on asetettava käyttöön verkkoselaimen asetuksista.

JavaScript on hyvin suosittu kieli käyttäjien keskuudessa ja sille on kehitetty monia kirjastoja, joiden avulla sen käyttöä voi tehostaa. JQuery on näistä kirjastoista ehkä suosituin ja sen virallisesta kehittämisestä vastaa jQuery Foundation -yhteisö.

Jotta jQueryn toiminnot saadaan käyttöön omassa web-sovelluksessa, täytyy sen tiedoston löytyä samasta sijainnista HTML-dokumentin kanssa. Lisäksi, tästä HTML-dokumentista pitää löytyä seuraava koodirivi: <script src="jquery.js"> </script> Tämän jälkeen HTML-dokumentissa on mahdollista käyttää kirjaston toimintoja.

Toinen vaihtoehto on hakea jQuery-kirjasto CDN-palvelujen kautta. Content Delivery Network tarkoittaa suurta verkostoa eri paikoissa sijaitsevia palvelinservereitä, joilta voi ladata sisältöä nopeasti ja suorituskykyä parantaen. Tällöin kirjasto ei olekaan suoraan itse sivun resursseissa vaan se ladataan käyttäjälle erikseen tällaisesta servereiden muodostamasta verkostosta (Methvin 2014). Esimerkiksi jQueryn tapauksessa, sivustolla on koodipätkä, joka lataa käyttäjälle jQuery-kirjaston, jota sitten hyödynnetään internetsivun sisällön näyttämisessä. Jos halutaan käyttää sivustolla tätä tapaa, lisätään koodiin seuraava rivi: <script src="https://code.jquery.com/jquery-2.0.0b1.min.js"> </script> Tällöin haetaan jQuerysta minimoitu versio 2.0.0b1 jQuery Foundationin yhteistyökumppanin, MaxCDN-yhtiön, ylläpitämästä verkostosta. Myös Microsoft ja Google ylläpitävät omia palvelinverkostoja koodikirjastoille, joihin kuuluu myös jQuery.

Vaikka jQuery-kirjasto perustuu JavaScriptiin, sen syntaksi on paljolti erilainen puhtaaseen JavaScriptiin verrattuna. Ominaisin jQuerylle on käsite $-selektori, jolla valitaan tietty dokumentissa käytetty elementti, esimerkiksi käyttämällä selektoria $(”p”) valitaan kaikki HTML-dokumentin kappale-elementit (<p>).

JQuery on myös JavaScript-koodia hieman tiiviimpää. Tämä tarkoittaa sitä, että JavaScript-koodia voidaan yleensä kirjoittaa muutamaa riviä lyhyempänä jQuery-kirjastoa hyödyntämällä.

Kirjaston käyttö mahdollistaa myös tietynasteisen selainriippumattomuuden. Kun kirjasto kulkee HTML-dokumentin mukana, käyttäjän selain ei vaikuta juurikaan sovelluksen toimivuuteen.

2.3 CSS

CSS, Cascading Style Sheets, tarkoittaa verkkosivulle suunnattuja erilaisia tyyliohjeita niiden ulkonäköä ja sommittelua varten. “Style” eli tyyli on sääntö, millaisena HTML-elementti sivulla näytetään ja “Style Sheet” viittaa kokoelmaan näitä sääntöjä. CSS ei tuo verkkosivuun mitään sisällöllistä lisää, mutta vaikuttaa siihen, miten tieto näytetään. (McFarland 2009, 1–2.)

CSS tarjoaa lukuisia tyylillisiä sääntöjä verkkosivun esittämiseen luovalla tavalla. Sen avulla voi esimerkiksi määritellä, millaista fonttia sivulla käytetään eri elementeissä tai miten kuvat ja kuviot asettuvat sivulla toisiin elementteihin nähden. CSS:n käyttö on erityisen hyödyllistä tärkeimmän tiedon selvässä esittämisessä.

CSS-tyyliohjeiden käyttö takaa myös verkkosivun tietojen helpon päivittämisen, koska se mahdollistaa ulkonäön ja sisällön erillisen käsittelyn. Jos haluaa muuttaa vain verkkosivun ulkonäköä, tarvitsee käsitellä vain tyyliohjeita eikä HTML-koodia juuri millään tavalla. Vastaavasti, jos sivulle lisätään uutta tietoa, aiemmin määrätyt tyyliohjeet pätevät uuteen sisältöön.

Tyyliohjeiden osoittaminen tietylle elementille vaatii sen HTML-tagiin yleisemmin id- tai luokka-muuttujan. Id-muuttujalla elementtiin voi säätää täysin uniikin tyylin, luokka-muuttujalla taas useisiin saman luokan jakavan elementtien kanssa yhtenevän tyylin. Myös elementin tagien mukaan voi asettaa tyyliohjeita, esimerkiksi asettamalla kaikkien sivun taulukoiden solujen tyylit samanlaisiksi.

CSS:n viimeisin standardi on CSS3, jonka oleellisin uudistus aikaisempiin standardin versioihin on CSS:n hajottaminen itsenäisiin moduuleihin, joista jokainen käsittää kokoelman tietynlaisia CSS-ominaisuuksia ja yksittäinen kokoelma valmistuu yksilöllisellä vauhdillaan. Tämän rakenteen tarkoituksena on, että CSS kehittyy osissa eikä yhtenä suurena kokonaisuutena, jota voi käyttää vasta kun se on kokonaan valmistunut. Sen sijaan, sitä mukaa kun yksittäinen moduuli valmistuu, selainten tarjoajat voivat kehittää sille tuen saman tien. (McFarland 2009, 438.)

Kuten aiemmin tuli sanottua, HTML5:n määrittelystä on julistettu lukuisia ulkonäköön vaikuttavia elementtejä vanhentuviksi sen takia, että ne ovat toteutettavissa helpommin CSS:llä. Tämän lisäksi, CSS on paljon kevyempi käyttää.

2.4 Web-tekniikoiden vahvuudet

Websovellusten tärkein vahvuus on ehdottomasti niiden laaja riippumattomuus käyttöalustasta, sillä niiden toimimiseen tarvitaan vain web-tekniikoita tukeva verkkoselain. Varsinkin mobiilipuolella suurin osa nykyisistä, suosituista verkkoselaimista, kuten Safari, Google Chrome ja Opera, tukevat useimpia HTML5-elementtejä (Mobile HTML5 2014). JavaScript ja CSS ovat yleisesti oleellisia web-tekniikoita, joten niillekin löytyy tuki useimmista selaimista, vain CSS3:n uusimmat ominaisuudet eivät välttämättä toimi pienemmän käyttäjämäärän selaimilla. Täten web-tekniikoin toteutettua yhtä sovellusta voi käyttää yhtä hyvin täysin erilaisten käyttöjärjestelmien laitteilla. Lisäksi on vielä mainittava toistamiseen jQueryn kaltaisten kirjastojen etu. Koska koodikirjasto liitetään joko sovelluksen resursseihin tai haetaan CDN:n kautta, ne eivät ole riippuvaisia ollenkaan selaimen tuesta.

Natiivisovelluksessa tällaista käyttöalustasta riippumattomuutta ei ole. Jos sovellus tehdään ensin Applen laitteille, ja sitten halutaankin tuoda käyttäjäpiiriin myös Android-käyttäjät, täytyy sovellus “kääntää” uudestaan Android Javalle. Jos taas halutaan siirtyä Microsoftin Windows-laitteisiin, vaatii se jälleen yhden sovelluksen ohjelmointikäännöksen lisää.

2.5 Web-tekniikoiden heikkoudet

Web-sovelluksissa suorituskyky voi olla heikompi kuin natiivisovellusten, mikä näkyy etenkin sovellusten latausajoissa. Tämä johtuu siitä, että HTML5-sovellukset tehdään enemmän “yksi malli kaikille”-säännöllä. (Brown 2014.) Tällöin sovelluksen yleiset raamit pyritään tekemään niin, että se toimii mahdollisimman hyvin kaikilla selaimilla, jolloin osa sovelluksen potentiaalista kärsii käyttäjän selainten rajojen takia. Natiivit sovellukset taas tehdään kohdennetusti tietylle laitteelle sen omalla ohjelmointikielellä, esimerkiksi Applen laitteille suunnatut sovellukset tehdään Objective-C:llä. Tällöin sovellus on täysin optimoitu juuri tietynlaiselle laitteelle huomioiden juuri sen kyseisen alustan ominaisuudet.

Lisäksi natiiveilla sovelluksilla pystytään hyödyntämään mobiililaitteiden ominaisuuksia. Sovellukset pystyvät käyttämään laitteen kameraa, ikkunoita voi vaihtaa pyyhkäisyllä ja sovelluksessa voidaan hyödyntää laitteen omia navigaationappeja. Web-tekniikoiden avulla näitä piirteitä ei pystytä hyödyntämään ainakaan vielä yhtä sulavasti, vaikka kehitys kiriikin jo tätä eroa kiinni. (Mahemoff 2011.) Kehitteillä onkin monenlaisia apukoodipaketteja nimenomaan mobiilialustan monipuolisuuden hyödyntämiseen myös web-tekniikoiden puolesta. Tällaisista mainittakoon esimerkiksi jQuery Mobile, joka pohjaa JavaScriptin jQuery-kirjastoon.

Näiden heikkouksien takia kannattaakin pohtia sovelluksen tarkoituksen kannalta, onko järkevämpää käyttää sen toteutuksessa web-tekniikoita vai nimenomaan alustalle kohdennettua ohjelmointikieltä. Jos sovelluksesta tulee suorituskyvyllisesti raskas, on parempi mennä alustan ehdoilla ja ohjelmoida sille ominaisella kielellä. Web-tekniikat taas ovat parhaimmillaan kevyissä sovelluksissa, joissa sisältö on tärkein, ei niinkään rakenne.

2.6 Selainten väliset erot

Koska web-tekniikoilla toteutetut sovellukset toimivat selaimissa, käsitellään lyhyesti suosituimpia selaimia ja erityisesti niiden välisiä eroja lähinnä mobiilitasolla. Suosituimpia selaimia ovat Google Chrome, Firefox, Internet Explorer, Safari sekä Opera.

Googlen Chrome-selain on saatavissa mobiilitasolla Androidin 4.0- ja sitä uudemmille käyttöjärjestelmän versioille sekä iOS-laitteille, tosin erilaisilla teknisillä ominaisuuksilla Android-versioon verrattuna. Chromen mobiiliversio syrjäytti aiemman mobiililaitteissa oletusselaimena olevan Android Browserin. Se pohjautuu Chromen pöytäkoneversioon, vaikka ei olekaan aivan yhtä kehittynyt. Chrome on kuitenkin yksi edistyneimmistä selaimista modernien web-standardien tukemisessa: HTML5-tuki kattaa jo kaikki oleellisimmat uudet elementit ja sovellusliittymät, CSS3-standardin mukaiset tyylit toimivat ja JavaScriptin suorittamiseen käytetään hyviä nopeuksia takovaa V8-moottoria (Mobile HTML5 2014; Developer Chrome 2015). Käyttäjän kannalta selaimen vahvoihin puoliin kuuluu synkronisointi eri laitteiden välillä, mikä toteutuu Chromeen yhdistetyn Google-tilin kautta, mikäli käyttäjä on näin tehnyt. Tällöin on mahdollista esimerkiksi käyttää samoja kirjanmerkkejä ja automaattisia lomaketallennuksia kaikilla Chrome-selaimen omaavilla laitteilla. Lisäksi Chromen mobiiliversioon on mahdollista saada Chrome to Mobile-laajennus, jonka avulla pöytäkoneelta voi lähettää mobiililaitteelle verkkosivun kopion, jota voi selata myös offline-tilassa. Se on kuitenkin ainoa laajennus, joka selaimeen voidaan liittää mobiilialustalla. Toisin kuin selaimen pöytäkoneversiossa, laajennuksia ei ole mahdollista lisätä mobiililaitteiden selainversioon. (Chrome ohjekeskus 2015.)

Mozillan Firefox on selain, joka on tällä hetkellä saatavilla vain Android-käyttöjärjestelmälle. Kuten Chrome, myös Firefoxilla on pitkälle kehittynyt HTML5- ja CSS3-tuki, ja nopea JavaScript-moottori (Mobile HTML5 2014; Mozilla Developer Network 2015). Käyttäjän kannalta muokattavuus on Firefoxin vahvuuksia, sillä selaimen mobiiliversio tukee laajennuksien ja muiden lisäosien lisäämistä, toisin kuin Chrome. Firefox-selaimen tiedot voidaan synkronisoida eri laitteiden välillä Firefox Sync-tilin kautta

Applen selain Safari on vain iOS-laitteisiin. HTML5:tä tuetaan, mutta se ei ole ihan yhtä pitkälle edistynyttä kuin Firefoxissa ja Chromessa. Jotkut ominaisuudet toimivat vain käyttöjärjestelmän uusimmissa versioissa ja joidenkin W3C-sovellusliittymien tuki on puutteellista (Mobile HTML5 2014). Synkronisointi OS-ympäristön laitteiden välillä on mahdollista, mutta se ei ole niinkään Safari-selaimen ominaisuus vaan iCloud-pilvipalvelun kautta tapahtuva toiminto.

Internet Explorer on Microsoftin Windows Phone-käyttöjärjestelmän selain. Selaimen uusin 11-versio tukee HTML5:n oleellisimpia ominaisuuksia, tosin samalla tavalla kuin Safarin suhteen, joidenkin sovellusliittymien kohdalla puutteita on. Esimerkiksi File System -sovellusliittymän tuki puuttuu kokonaan, mutta yleisesti katsottuna, web-tekniikoiden tuki on parantunut huomattavasti selaimen aiempiin versioihin verrattuna (Mobile HTML5

2014). Laitteiden välinen synkronisointi pohjautuu selaimen sijaan laitteiden Microsoft-tilin tai vaihtoehtoisesti OneDrive-pilvipalvelun toimintaan ja selaimen tila synkronisoituu sen kautta.

Microsoft on ilmoittanut Internet Explorerin katoamisesta uuden Windows 10-käyttöjärjestelmän tulon mukana, ja puhunut täysin uudesta, korvaavasta selaimesta, koodinimeltään Project Spartan. Huhtikuun Build 2015- tapahtumassa Microsoft paljasti tämän uuden selaimensa viralliseksi nimeksi ”Edge”. Se on oletusselain kaikkiin Windows 10-käyttöjärjestelmän laitteisiin, niin mobiililaitteisiin kuin pöytäkoneisiinkin. (Foley 2015.) Uusien ominaisuuksien lisäksi sen on tarkoitus olla nopeampi ja kevyempi kuin edeltäjänsä. Lisäksi se tulee tukemaan myös HTML- ja JavaScript-pohjaisia liitännäisiä. (Newman 2015.)

Opera Softwaren Opera-selaimesta on kaksi eri versiota. Opera Mini-version saa kaikille käyttöjärjestelmille ja se toimii lähes kaikissa puhelimissa. Lisäksi on Opera for Android, jonka saa vain Android-käyttöjärjestelmälle.

Opera Mini on selaimen karsittu versio, jonka tärkein ominaisuus on nopeus ja tietojen pakkaus niin, että tiedonsiirto on mahdollisimman vähäistä. (Opera Software 2015). Tämä mahdollistetaan niin, että käyttäjän mennessä jollekin verkkosivulle, itse sivu käsitellään Operan servereillä, minkä jälkeen se lähetetään takaisin renderöitynä versiona käyttäjän puhelimeen. Tämä vaikuttaa etenkin JavaScript-koodin suorittamiseen ja asettaa siihen tietynlaisia rajoituksia. Tavallisessa tapauksessa JavaScript-koodi suoritetaan oitis, mutta mikäli koodin suorittaminen vie liikaa muistia, se lakkautetaan. (Brown 2012.) Muidenkaan web-tekniikoiden kannalta tämä selaimen rajoittuneisuus ei ole ihanteellisin ominaisuus. Opera Mini ei tue uusista HTML5-elementeistä kuin muutamaa ja CSS3-tyyleistä tuetaan vain paria, niitäkin vain osittain (Mobile HTML5 2014).

Opera for Android on sen sijaan kokonaisempi selain, joka perustuu pöytäkoneversioon. Se pohjautuu avoimen lähdekoodin Chromium-projektiin, joka on sama pohja kuin Googlen Chromessa (Opera Software 2015). Sen myötä Opera-selain tarjoaa samankaltaisuuksia Chromeen verrattuna, esimerkiksi saman JavaScript V8-moottorin. Lisäksi HTML5- ja CSS3-tuki on lähes yhtä kehittynyttä (Mobile HTML5 2014). Myös Operan selain tarjoaa mahdollisuuden synkronoida eri laitteiden selaintiedot keskenään Opera Link -palvelun avulla.

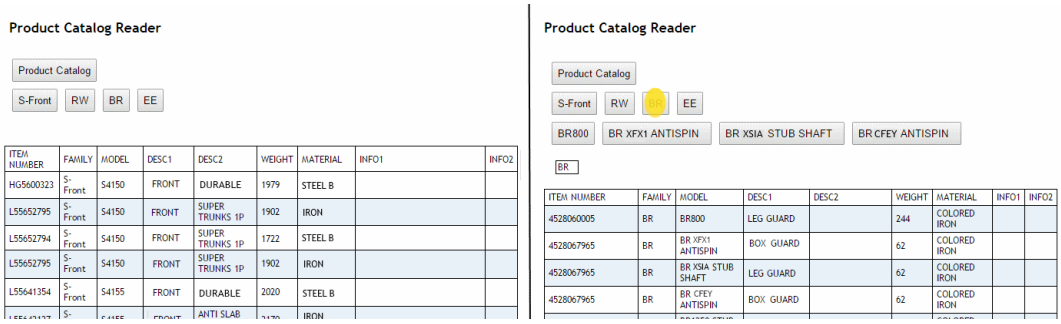
3 ALKUPERÄINEN OHJELMA

Tässä luvussa avaan vähän alkuperäistä ohjelmaa, jonka pohjalta työharjoittelussa tekemäni ohjelmointityö perustuu.

Asiakasyrityksellä oli käytössään CSV-tiedostomuotoisia (Comma Separated Values) tuotekatalogeja, joissa on lueteltu erillisten tuotteiden tietoja, muun muassa näiden tuotteiden tunnistenumerot ja mallistot. Aluksi ohjelma lukee tällaisen käyttäjän valitseman tiedoston ja tulostaa kaikki katalogissa olevat tuotteet ja niiden tiedot taulukkoon. Samalla taulukon yläpuo-

lelle riviin luodaan niin monta nappia kuin tiedostosta löytyy eriäviä tietoalkioita. Näiden nappien sisällöt toimivat eräänlaisina suodattimina, joiden pohjalta taulukon tietoja karsitaan.

Kun käyttäjä painaa tällaista ensimmäisen tason nappia, esimerkiksi tuoteluokkaa, ohjelma karsii taulukon tietoja niin, että näkyville jää vain tämän ehdon täyttävät tuotteet. Jälleen näiden suodatettujen tietoalkioiden pohjalta luodaan uusi rivi nappeja, joilla tuotteita voidaan rajata yhä tarkemmin, esimerkiksi tällaisen toisen tason suodattimet voisivat kuvata valitun tuoteluokan mallistoja. Esimerkkinä toiminnosta on kuva 2.



Kuva 2. Ohjelma karsii taulukon tietoja käyttäjän valintojen mukaan.

Yhä eteenpäin liikuttaessa, taulukon tietoja suodattavista napeista muotoutuu puumainen rakenne, josta käyttäjä voi aina tarkistaa, millaisen tuoteryhmän valikossa liikkuu. Puumainen rakenne mahdollistaa myös sen, että käyttäjä voi halutessaan palata helposti taaksepäin, esimerkiksi hän voi milloin tahansa palata takaisin ja valita uudelleen toisen tason napeista saman tuoteluokan eri malliston. Kun käyttäjä löytää tällaisen taso-tasolta etenemisen kautta etsimänsä tuotteen, hän pystyy helposti lukemaan etsimänsä tuotteen tiedot.

Alkuperäinen ohjelma oli Windows-käyttöjärjestelmälle luotu pöytäkoneversio, joka oli ohjelmoitu Flashilla, jolle taas ei ole tukea mobiilialustalla. Lisäksi mobiiliversion luonnille oli perusteita mobiililaitteiden käytön kasvun takia: Ohjelman käyttäjille olisi paljon mieluisempaa kantaa mukanaan kevyempää laitetta kuin alkuperäisen ohjelman käyttöön tarvittavaa raskaampaa kannettavaa tietokonetta.

4 ALUSTOJEN EROT – DESKTOP JA MOBIILI

Pöytäkoneiden ja mobiilialustan välillä on paljon eroavaisuuksia, jotka on huomioitava kun alkuperäisestä sovelluksesta tehdään mobiiliversiota. Tässä luvussa esittelen tärkeimmät eroavaisuudet tämän työn kannalta näiden kahden eri alustan välillä.

4.1 Flash-tuen puute

Adoben Flashilla on ollut koko ajan yskähtelyä mobiilialustalle siirryttäessä. Kovin isku yhtiölle tuli kun Apple päätti kieltää kaikki Flashia käyttä-

vät sovellukset Apple Store-kaupasta ja samalla Steve Jobs kirjoitti ajatuksistaan muun muassa Flashiin liittyvistä turva-aukoista, HTML5:n tulemisesta ja Flashin soveltumattomuudesta mobiilialustalle (Jobs 2010). Tämän jälkeen Adobe joutui pohtimaan mobiilistrategiaansa uusiksi ja vuoden 2011 syksyllä yhtiö ilmoittikin, ettei se enää kehitä Flash-laajennusta mobiiliselaimille. Yhtiö on myöntänyt, että HTML5:n toimivuus alustasta riippumatta tekee siitä parhaimman tekniikan mobiililaitteiden selaimessa, ja yhtiössä halutaankin panostaa tämän tekniikan jatkokehittämiseen muiden yhteistyökumppanien kanssa (Winokur 2011).

Nimenomaan tämän Flash-tuen puutteellisuus antoi asiakasyritykselle tarpeen tehdä erillinen mobiiliversio alkuperäisestä ohjelmastaan. Pelkkä ominaisuuksien ja ulkonäön karsiminen ei riittänyt kun ohjelmointikielen tuki on niin haparoivaa mobiilialustalla.

4.2 Näyttörajoitukset

Mobiililaitteiden kirjo on valtava. Erilaisten käyttöjärjestelmien lisäksi käyttäjälle on tarjolla erilaisia laitteita tableteista älypuhelimiin ja varsinkin Android-käyttöjärjestelmän laitteita on lukematon määrä eri valmistajien myötä. Mobiililaitteita on erikokoisia, eri tarkkuusresoluutioilla ja erilaisella raudalla. Esimerkiksi Android-laitteita on puhelimia, tabletteja ja jopa televisioita.

Myös niin sanottu ristikäyttäminen on yleistä, ja samoja sovelluksia saa sekä puhelimiin että tablettilaitteisiin. Lisää ulottuvuutta tuo myös se ominaisuus, että laitetta voi tarvittaessa kääntää. Sivuasennossa näyttöön saadaan lisää leveyttä, pystyasennossa pituutta. On kuitenkin täysin käyttäjästä kiinni, kummassa asennossa laitetta on miellyttävämpi käyttää eikä ohjelman tekijän siten kannata ohjeistaa käyttäjäänsä siitä, miten päin laitetta on pidettävä saadakseen sovelluksesta kaiken irti. Havainnollistavana esimerkkinä kaikesta tästä on kuva 3.

Kuva 3. Erikokoisia Android-käyttöjärjestelmän laitteita eri asennossa.

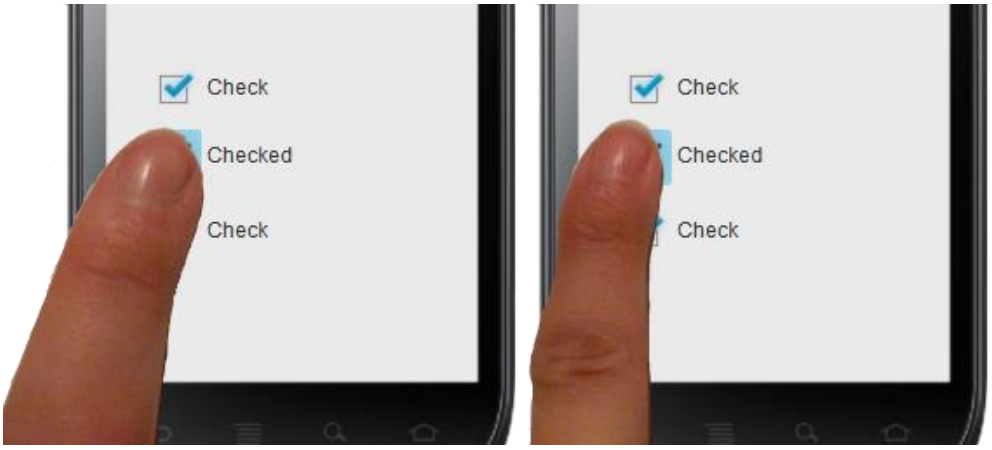
Kuitenkin paljolti laitteesta huolimatta, mobiililaitteen näyttö aiheuttaa merkittävän muutoksen pöytäkoneeseen verrattuna. Vaikkakin uusimmissa laitteissa tarkkuusresoluutio alkaa olla jo melkoisen hyvissä lukemissa, silti mobiilialustalla työskennellään paljon pienemmillä näytöillä. Silti tuohon näyttöön pitää mahtua suhteellisesti sama informaatiomäärä kuin pöytäkoneen näytöllä ilman, että käyttäjän tarvitsee jatkuvasti vuoroin suurentaa ja pienentää näkymää. Haastetta tuo myös se, että mobiililaitteessa vain yksi sovellus voi näkyä näytöllä yhdellä kertaa, kun taas pöytäkoneella voidaan pitää vierekkäin monia eri informaatioikkunoita.

4.3 Sormi osoittimena

Hiiren käyttäminen pöytäkoneen näytöllä on täysin erilainen tapahtuma verrattuna kosketusnäyttöihin, jotka vaativat käyttäjän sormenpään osoittimekseen. Toisin kuin hiiren osoitin, ihmisten sormet ovat erikokoisia ja siten hyvinkin epäeksakteja osoittimia siinä mielessä, että voi olla hankala hahmottaa, mikä kohta isossa peukalossa toimii osoittimen keskustana. Toisaalta taas markkinoilla on erilaisia stylus-kyniä, joilla kosketusnäytöllä työskentelemistä voi helpontaa.

Lisäksi kosketusnäytöllä tämä osoittimen käyttö tulee voimaan vain kun sillä on ensin painettu jotakin näytön elementtiä, toisin kuin hiirellä, jonka liikkeen voi jatkuvasti hahmottaa tietokoneen ruudulla. Esimerkiksi, verkkosivulla hiiren liikkeen voi nähdä käyttäjän viedessä sen sivulla olevan hyperlinkin kohdalle ja kun hiiren osoitin on valitun elementin kohdalla, saa käyttäjä tästä visuaalisen palautteen, useimmiten hiiren normaalin valinnan osoitin muuttuu kämmenen kuvaksi. Kosketusnäytöllä ja sormen käytöllä on erilainen tilanne. Käyttäjä saa palautteen toiminnasta vasta kun hän sormenpäällään painaa näytöllä hyperlinkin kohtaa ja näytöllä oleva sivu muuttuu toiseksi.

Sormien käyttöön liittyy myös elementtien koko. Mobiililaitteille suunnatussa sovelluksessa täytyy tarkasti miettiä, minkä kokoisia elementtejä kannattaa käyttää. Myös interaktiivisten elementtien väli kannattaa huomioida, ettei tapahdu niin, että suurempi sormenpää painaa montaa tällaista elementtiä samalla kertaa. Esimerkiksi Applen (2014) sovellusten kehitysoppaassa ”iOS Human Interface Guidelines” suositellaan käyttämään kosketusnäytöille suunnatussa sovelluksessa vähintään 44 x 44 pisteen kokoisia interaktiivisia nappeja. Esimerkkinä käyttötilanteesta on kuva 4.



Kuva 4. Ihmisillä on erikokoiset sormet, ja elementtien kokoa valittaessa se on syytä huomioida.

Kosketusnäytöille on tosin kehitetty työskentelyä tehostamaan tiettyjä eleitä (gestures), joita pöytäkoneiden hiiret eivät tunne. Osa näistä on mobiilialustalle uniikkeja eleliikkeitä, esimerkiksi nipistäminen (pinch) näkymän lähentämisessä, mutta jotkut eleet tekevät eri asian hiirellä. Oleellisin näistä on kaksoisnapautus. Hiirellä kaksoisnapautus avaa jonkin tiedoston tai verkkosivulla kuvan tai muun elementin erilliseen ikkunaan kun taas mobiililaitteella nopea kaksoisnapautus sormella lähentää näkymää oletusarvon verran. Tämä on ongelma, joka on huomioitava varsinkin kun tehdään sovellusta sekä pöytäkoneelle että mobiilialustalle.

4.4 File Explorer ja tiedostonhallinta

Laitteen tiedostojen selailu ja niiden lataaminen verkkopalveluihin on arkipäivää, varsinkin kuvien ja videoiden kohdalla. Siten kuvan 5 -tyylinen “Valitse tiedostot” -nappi on tuttu näky verkossa kenelle tahansa tietotekniikan käyttäjälle.

Kuva 5. Tyypillinen HTML-tiedostonvalintaan johtava nappi

Pöytäkoneella, varsinkin Microsoftin Windows-käyttöjärjestelmällä, tällaista nappia painaessa avautuu erillinen tiedostonäkymä, jonka kautta voi hakea haluamansa tiedoston kovalevyltä ohjelman luettavaksi.

Tiedostojärjestelmä on kuitenkin hieman erilainen mobiiliympäristössä. Mobiilikäyttöjärjestelmät ovat periaatteessa sovelluskeskeisiä, ja näiden sovelluksien oletetaan toimivan omassa hiekkalaatikossaan. Siksi kuvatiedostot löytyvät niiden selailuun tarkoitetun galleria-sovelluksen kautta, musiikkitiedostot musiikkia toistavan sovelluksen kautta ja niin edespäin. Applen (2014) kehitysoppaassa ”File System Programming Guide” sanotaan suo-

raan, ettei iOS-laitteiden käyttäjillä ole pääsyä suoraan tiedostojärjestelmään, ja sovellusten oikeudet tuohon järjestelmään rajoittuvat vain sovelluksen tiedostoihin. Myös Android käyttöjärjestelmä noudattaa laajalti samoja periaatteita, ja sovellukset toimivat omassa hiekkalaatikossaan. Android pohjaa UNIX-käyttöjärjestelmään, jossa tärkeintä on rajoittaa erillisten käyttäjien pääsyä toistensa tiedostoihin, ja tämä tapa on välittynyt myös mobiililaiteympäristöön. Sovellukset mielletään käyttäjiksi, joilla ei ole lupaa käyttää toisen käyttäjän tiedostoja. (Android Open Source 2014.)

Tiedostojärjestelmän sulkeutuneisuudesta johtuen varsinainen tiedostonhallintakin on hankalaa eri alustoilla. Kuten sanottua, Apple ei päästä käyttäjiään tutkimaan tiedostojärjestelmää vapaasti, ja myös Microsoft oli Windows Phone -mobiilikäyttöjärjestelmänsä kanssa aluksi samoilla linjoilla. Vasta Windows Phone 8.1 -version tultua markkinoille, yhtiö ilmoitti, että käyttöjärjestelmään tulee saataville oma tiedostonhallintasovellus, jolla käyttäjä voi hallita omia tiedostojaan samaan tapaan kuin tavallisella pöytäkoneella (Foley 2014).

Sen sijaan Android-käyttöjärjestelmän tilanne tiedostonhallintasovellusten suhteen riippuu paljolti valmistajista. Joistakin laitteista löytyy niin sanottu tehdassovellus vapaata tiedostojen hallintaa varten, toisista taas ei. Tätä tarkoitusta varten voi kuitenkin käyttää erilaisia kolmannen osapuolen tiedostojen hallintasovelluksia, joita löytyy Google Play-kaupasta.

5 OHJELMAN TOTEUTUS

Kun ohjelmasta on jo käytössä oleva versio, jonka pohjalta tehdään toisella alustalla toimiva versio, sovellus voi tarvita täydellistä uudelleentekemistä. Tässä kappaleessa käsittelen ennen varsinaista koodaamista tehtävää suunnittelua ja räätälöintiä uudelle alustalle, ohjelmaan tehtyjä koodillisia ja ulkonäöllisiä ratkaisuja sekä lopuksi vielä muutama arviollinen sana toteutuksen onnistumisesta.

5.1 Sovelluksen räätälöinti

Sovelluksen siirtäminen pöytäkoneelta mobiililaitteelle vaatii alustan mukaista soveltamista. Mobiililaitteissa on vähemmän tallennustilaa ja RAM-muistia kuin pöytäkoneissa, joten sovellusten koko ja suorittamiseen vaadittava kapasiteetti on pidettävä aisoissa. Sovellus ei saa käyttää liikaa rajallisten akkujen virtaa.

Katalogisovelluksen kehitystyön alkuvaiheessa tärkeintä oli selvittää alkuperäisen ohjelman käyttötarkoitus ja oleellisimmat toiminnot, jotka palvelivat tuota tarkoitusta. Sovellus oli suunniteltava niin, että ne osat, jotka eivät olleet ”yhtä tärkeitä” käyttötarkoituksen osalta, esimerkiksi alkuperäisen sovelluksen kuvat, oli rajattava minimiin näyttörajoitusten, suorituskyvyn ja ohjelman koon takia.

Ensinnäkin, sovellusta käyttäisivät ensi sijassa asiakasyrityksen myyjät. He käyttävät sovellusta pääosin tietyn tuotteen etsimiseen aiemmin esitellyn

puurakenteen avulla. Useimmiten tuotteesta halutaan tietää sen tuotenumero. Mahdollisuus hakea taulukosta teksti- tai numeropätkiä voi auttaa merkittävästi tuotteen löytämisessä, jos käyttäjä esimerkiksi muistaa osan tuotenumerosta tai sanan tuotteen kuvauksesta.

Pääosassa sovelluksessa on kuitenkin suuren tietomäärän sisältävä taulukko, jonka tiedot luetaan CSV-tiedostosta. Taulukon sisällön pitää kuitenkin olla yhtä selkeästi luettavissa, työskenneltiin sitten suuremmalla tai pienemmällä ruudulla. Toinen elementti, jonka työstämisessä on huomioitava näyttörajoitukset, on puurakenne, joka muodostuu tietoja suodattavista napeista. Näitä nappeja voi riviin tulla montakin, riippuen täysin siitä, kuinka monta eriävää tietoalkiota jokaiselta suodatetulta tasolta löytyy. Nappien on oltava niin leveitä ja korkeita, että niitä voi painaa erikokoisilla sormilla. Lisäksi niiden leveyteen vaikuttaa myös suodattavan tietoalkion tekstijonon pituus.

Sovelluksen testaamista varten saatiin käyttöön testitiedoston, joka vastasi varsinaisessa käytössä olevia oikeita tiedostoja. Tässä CSV-tiedostossa oli seitsemän suodatustasoa ja nappien määrä yhdellä rivillä, eli eriävien tietoalkioiden määrä yhdellä tasolla, vaihteli yhdestä seitsemään. Koko taulukon tietorivien määrä oli kuitenkin yli 600 rivin luokkaa.

5.2 Ohjelman uudelleenkirjoitus

Sovellus kirjoitettiin kokonaan uudestaan web-tekniikoita käyttäen. Periaatteessa koko sovelluksen toiminnollisuus pilkottiin pieniksi, itsenäisiksi moduuleiksi: tiedoston valinta, tiedoston luku, taulukon luonti, taulukon täyttö ja niin edespäin.

Kaikki sovelluksen toiminnallisuus perustuu HTML5:n ja JavaScriptin yhteistyöhön, CSS vastaa lähinnä käyttöliittymän ulkonäöstä ja tiedon selkeästä esittämisestä. HTML5 tarjoaa sovellukseen peruselementit, taulukon ja puurakenteen napit, JavaScriptin kautta tehdään CSV-tiedoston luku ja ohjelmakoodi vastaa tietojen suodattamisesta käyttäjän valintojen mukaan. JavaScriptin jQuery-kirjastoa hyödynnettiin muutamissa koodipätkissä, joissa sen käyttö oli puhdasta JavaScriptiä helpompaa tai koodimääräisesti lyhyempää. CSS:n käytöstä tarkemmin seuraavassa osiossa, jossa käsittelen käyttöliittymää.

Kaikki alkaa HTML-elementistä ”input”, napista, jolla valitaan tiedosto luettavaksi. Se luodaan HTML-dokumenttiin kuvan 6 mukaisesti.

```
<input type="file" id="files" name="files[]" multiple />
```

Kuva 6. Tiedostonvalintaan tarkoitetun napin luominen

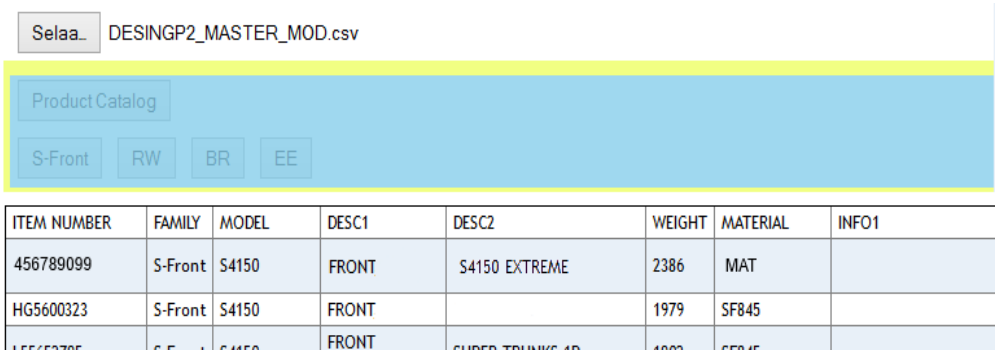
Elementin type=”file”-attribuutin arvo kertoo, että kyseessä on tiedostonvalintaan tarkoitettu, toiminnallinen nappi. Attribuutti ”multiple” on täysin uusi HTML5:n attribuutti, joka mahdollistaa monen tiedoston valinnan yhdellä kertaa. Tämä on hyödyllinen ominaisuus siltä varalta, että käyttäjä voi

avata monta tiedostoa ja käydä ne läpi samalla kertaa. On kuitenkin huomioitava, ettei tätä monen tiedoston valitsemista vielä ole mobiilialustalla, joten tämä ominaisuus toimii toistaiseksi vain pöytäkoneilla. Se ei kuitenkaan tuo mukanaan esimerkiksi suorituskyvyn heikentymistä, joten attribuutin olemassaolosta ei ole mainittavaa haittaa, jonka takia sen poistamista olisi syytä harkita.

Kun käyttäjä on löytänyt, ja valinnut haluamansa tiedoston, JavaScript-koodi lukee tiedoston merkkijonona erillisellä FileReader-API:lla. Tämän jälkeen tiedoston sisältö parseroidaan riveiksi, jotka luetaan taulukkoon, rivin data taas erotellaan erillisiksi rivin soluiksi. Tämä tapahtuu CSV-tiedostossa olevien välimerkkien kautta, ”; ” tarkoittaa solujen erotinta ja ”\n” rivinvaihtoa. Käyttäjälle näytetään vain tämän koko prosessin lopputulos: taulukko täynnä haluttua tietoa, esitettynä eriävillä riveillä ja rivien soluina.

Samalla kun JavaScript-koodissa luetaan tietoa taulukkoon, se poimii suodattamiksi arvoltaan eriävät tietoalkiot per yksi datakolumni. Näiden eriävien alkioiden pohjalta aloitetaan puurakenteen luonti. Tämän puurakenteen napit luodaan JavaScript-koodilla. Ne ovat HTML input-elementtejä, joiden type-attribuutin arvoksi tulee ”submit”. Ohjelma ajaa aluksi eriävien alkioiden nimet array-taulukkoon, ja erillinen funktio luo puurakenteeseen yhtä monta nappia kuin tässä taulukossa on alkioita. Tästä taulukosta otetaan napeille value-attribuutin arvo, joka näkyy käyttäjälle tekstinä napin päällä. Nappeihin on määritetty onclick-tapahtuma, joka suorittaa varsinaisen taulukon suodattamisen painetun napin value-arvon perusteella. Kun taulukon tiedot suodattuvat, luodaan jälleen uusi rivi nappeja samalla periaatteella.

Napit luodaan siis vasta tietojen taulukkoon lukemisen jälkeen. Koska koodi luetaan periaatteessa rivi riviltä funktioiden suorittamisen mukaan, lopuksi käy niin, että nappien pitäisi periaatteessa ilmestyä vasta jättimäisen taulukon alle. Siksi, napit luodaan omaan div-elementtiin (osio), joka on HTML-dokumentissa määritetty ennen varsinaista suodatustaulukkoa. Täten napit ovat aina taulukon yläpuolella ja sieltä helposti käytettävissä. Tätä asettelua esittää kuva 7.

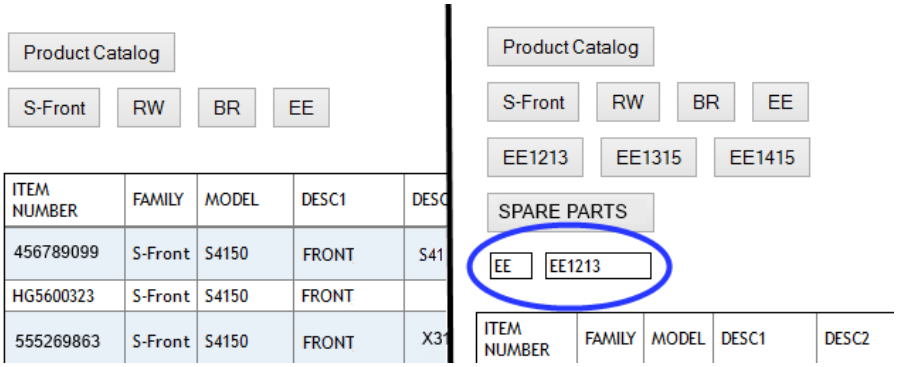


Kuva 7. Oma (tässä kuvassa korostettuna esitetty) div-elementti takaa, että suodatusnapit sijaitsevat aina taulukon yläpuolella.

Samalla kun taulukossa suodatetaan tietoja, suodatettujen alkioiden nimet kerätään toiseen array-taulukkoon, kutsuttakoon tätä ”storage”-taulukoksi.

Tämä on olennainen osa puurakenteen toimivuutta alas- ja ylöspäin luettaessa, sillä ohjelma tarkistaa aina, että taulukossa esitetyt tiedot vastaavat kaikkia ”storage”-taulukossa olevia alkioita ja vain näitä alkioita. Ne rivit, joiden solut eivät vastaa kaikkia tässä taulukossa olevia alkioita, karsitaan pois näkyvistä.

Käyttäjän käyttäessä sovellusta, hän haluaa myös pysyä ns. ajan tasalla, mitä ”polkua” pitkin hän on päässyt tämänhetkiseen taulukon tilaan. Tätä varten kehitettiin heti nappien alapuolelle toinen div-elementti, johon kerätään kaikkien käyttäjän painamien nappien arvot. Nämä esitetään sitten tekstilaatikkoina, jotta käyttäjä tunnistaa eri nappien rajat, mikäli nimissä on vastaavuutta, esimerkiksi saman tuotteen eri ryhmien alkukirjaimet. Tästä on esimerkkinä kuva 8.



Kuva 8. Esimerkki polun näyttävistä tekstilaatikoista, kun käyttäjä on valinnut EE-tuoteryhmän EE1213-sarjan.

Entä sitten kun käyttäjä palaa puurakenteessa takaisin ylemmille tasoille, esimerkiksi jonkin tuotteen viimeiseltä tasolta toiselle tasolle valitakseen jonkin toisen tuoteryhmän? Miten käy napeille ja polun tekstilaatikoille? Ohjelma tunnistaa, että käyttäjä on painanut jotain aiemmin luotua nappia, ja karsii aiemmin mainitusta ”storage”-taulukosta kaikki alkiot, jotka ovat array-taulukossa painettua nappia vastaavan alkion jälkeen. Samalla tiedot päätaulukossa päivittyvät, ja aiemmin pois suodatetut tiedot tulevat jälleen näkyviin. ”Storage”-taulukon pituudesta saadaan tietää myös juuri painetun napin taso, minkä perusteella poistetaan sekä ylimääräiset napit että polun tekstilaatikot. Kun käyttäjä jälleen menee uutta polkua pitkin etsimään jotain tuotetta, napit ja tekstilaatikot päivittyvät uusien tietojen mukaan. Poikkeus tässä osiossa on se, että nappien ja tekstilaatikoiden poistamiseen käytetään jQuery-kirjastoa hyödyntävää koodia. Painetun napin tunnistaminen ja tarpeettomien elementtien poistaminen on jQuerylla helpompaa ja lyhyempää kuin puhtaalla JavaScriptillä.

Aiemmin kuvassa 8 näkynyt ”Product Catalog”-nappi on tavallaan muista napeista ja tietojen suodattamisesta riippumaton, erillinen nappi. Sen ainoa tehtävä on ladata koko tiedosto uudelleen näkyville käyttäjän niin halutessa. Jos nappuloita on painettu kerrankin, koko taulukkoa ei saa enää näkyville ilman erillistä nappia, koska jo ensimmäisen tason napit suodattavat tietoa tuoteperheiden perusteella.

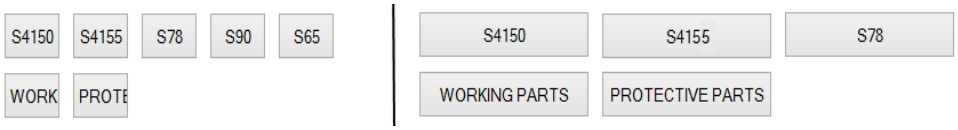
Sovelluksessa on myös hakutoiminto. Se on HTML ”input”-elementin kenttä, joka prosessoi käyttäjän syöttämää tekstiä ja numeroita. Haun toiminnassa oleellisinta on taulukon senhetkinen tila, sillä se hakee etsityn merkkijonon vain näkyvillä olevasta taulukosta eikä mahdollisesti aiemmin suodatettuja tietoja huomioida lainkaan. Haku toimii niin, että heti kun käyttäjä alkaa naputella merkkejä ”input”-tekstikenttään, tämä merkkijono luetaan hausta vastaavan funktion sisään. Funktio käsittelee merkkijonon ja taulukosta etsitään saman tien kaikki alkiot, joissa tämä merkkijono esiintyy. Jos käyttäjä pyyhkii koko tekstin pois, taulukko palautuu automaattisesti siihen tilaan, jossa se oli ennen haun aloittamista. Haun aloittavia tapahtumia, esimerkiksi ylimääräisiä Enter-lyöntejä ei tarvita missään vaiheessa.

5.3 Käyttöliittymän hiominen

Käyttöliittymää säätäessä täytyy muistaa ennen kaikkea se, keitä ovat sen käyttäjät ja miksi he sovellusta käyttävät: Käyttäjiä ovat asiakasyrityksen myyjät, jotka etsivät sovelluksesta tiettyjä tuotteita näiden ominaisuuksien perusteella, useimmiten tärkein asia on löytää tuotteen tuotenumero. Toinen asia, mikä tämän sovelluksen käyttöliittymässä on huomioitava, on sen kohdealusta eli tässä tapauksessa mobiilialusta.

Kaikkien sovelluksessa käytettävien HTML-elementtien sijainnit, leveydet, pituudet ja värit on määritettävissä CSS:n avulla.

Puurakenteen napit ovat hyvin tärkeitä sovelluksen käytössä. Kuten aiemmin kappaleessa 4.3 totesin, mobiililaitteiden näyttöjen osoittimena käytetään sormenpäätä, ja sormenpäiden koot vaihtelevat ihmisestä riippuen. Sen takia nappien korkeutta säädellään oletuskokoa suuremmaksi, jotta sitä pystyy painamaan vähän isommallakin sormenpäällä. Leveyteen puuttumisessa on kuitenkin syytä olla varovainen. Jotkut nappeihin tulevien tuotteiden ominaisuudet ovat tekstijononsa pituudelta pidempiä kuin toiset. Jotkut ovat vain muutaman kirjaimen pituisia ja toiset taas voivat kasvaa yli 30 kirjaimeen. Jos nappi määritetään leveydeltään liian pieneksi, pidempien alkioiden teksti katkeaa kesken. Samankaltaisten tekstien kohdalla tämä voi olla ongelma, jos napin teksti poikkeaa juuri siitä kohtaa, josta eroavaisuus normaalitilanteessa näkyisi. Toisaalta taas, jos nappi määritetään liian leveäksi, muutaman kirjaimen pituisten alkioiden napeissa on liikaa tyhjää tilaa. Tällöin nappi vie aivan liikaa turhaa tilaa ja se tila on pois muilta elementeiltä. Tällainen tilanne taas on mobiililaitteiden näyttöjen rajallisuuden takia täysin katastrofaalinen. Molemmista tilanteista esimerkkiä kuvaa kuva 9.



Kuva 9. Lyhyeksi säädetyt napit katkaisevat pitkän tekstin mutta turhan pitkät napit vievät tilaa.

Toisaalta CSS tarjoaa ominaisuuden ”min-width”. Tämä on elementin minimaalisin sallittu koko leveyssuunnassa. Sen säätäminen lyhyimpien, eli

muutamien kirjainten pituisten tekstijonojen varalta voi olla hyvä ajatus. Ideaalisen pituuden löytäminen kuitenkin vaatii testaamista, sillä nappeja pitää pystyä painamaan isollakin sormenpäällä.

Toinen asia, mikä pitää huomioida nappien koon kannalta, on niiden välillä oleva tyhjä tila. Tämä on tyyliominaisuus ”margin”. Määrittelemällä napeille muutamien pikselien kokoiset marginaalit kaikkiin suuntiin, voidaan välttää se, ettei isommalla sormenpäällä paineta montaa nappia samaan aikaan.

Toinen sovelluksessa pääosassa oleva elementti on täynnä tietoa oleva taulukko, jonka selkeys on yhtä tärkeä kuin puurakenteellakin. Pelkkä HTML-taulukko ilman tyylimuotoilua ei ole pelkästään karun näköinen vaan aika epäselkeäkin. Kuvan 10 esimerkkitaulukkoon ei ole lisätty erillisiä CSS-tyylejä ja tulos on melko epäselkeä: Solun teksti on työntynyt kiinni vasempaan reunaan. Tekstittömät solut on jätetty täysin tyhjiksi eikä niihin tule edes erottelevia reunaviivoja. Toisaalta taas, ne solut, joissa on reunaviivat, näyttävät oudolta ihmissilmälle. Pahin virhe on kuitenkin se, ettei rivien välillä ole minkäänlaista kontrastia, mikä pitkien taulukoiden kohdalla voi hidastaa tarvittavan tiedon etsimistä.

ITEM NUMBER	FAMILY	MODEL	DESC1	DESC2	WEIGHT	MATERIAL	INFO1	INFO2
36769541245	RW	RW800	COVER	METAL S / M / L	2944	SF520		
545645446456	RW	RW800	COVER	METAL S / M / L	2944	SF520		
02546802535	RW	RW800	COVER	METAL S / M / L	2944	SF520		
4000253695	RW	RW800	COVER	METAL S / M / L	2944	SF520		
5656506565	RW	RW800	SEED PLATE		82	SF845		
Please contact product line	RW	RW800	SEED PLATE		82	SF900		
Please contact product line	RW	RW800	SEED PLATE		130	SF845		
35678241232	RW	RW800	SEED PLATE		130	SF900		
56454215455	RW	RW800	SEED PLATE	METAL S / M / L	251	SF845		

Kuva 10. Pelkkä HTML-taulukko

CSS:n tyyleillä voidaan vaikuttaa selkeyteen esimerkiksi valitsemalla selkeämpi fontti ja sen koko, lisäämällä taulukon solujen ”padding”-ominaisuutta muutamilla pikseleillä niin, että tekstin ja reunan väliin tulee hieman tilaa, jolloin solun tekstin voi lukea nopealla silmäyksellä. Solun reunaviivalle asetetaan tyylilliset ominaisuudet niin, että se erottuu selkeästi. Hyvä perusratkaisu on musta, yhtäjaksoinen yhden pikselin paksuinen viiva. Tällöin tyhjiinkin soluihin tulee reunaviivat, ettei rivien välinen raja katoa. Lisäksi riveihin voidaan lisätä kontrastia, jotta varsinkin pitkien taulukoiden lukeminen helpottuu. Kontrastin hakemisessa toimivin klassikkoratkaisu on niin sanotut ”seepraraidat”, jolloin joka toiseen riviin on määritetty valkoisesta poikkeava väri. Värit saatiin alkuperäisestä ohjelmasta, sillä tämä oli asiakasyrityksen edustajan mielestä toimiva ratkaisu. Kuva 11 esittää näillä CSS-ominaisuuksilla koristettua taulukkoa.

ITEM NUMBER	FAMILY	MODEL	DESC1	DESC2	WEIGHT	MATERIAL	INFO1	INFO2
36769541245	RW	RW800	COVER	METAL S / M / L	2944	SF520		
545645446456	RW	RW800	COVER	METAL S / M / L	2944	SF520		
0254692535	RW	RW800	COVER	METAL S / M / L	2944	SF520		
4000253695	RW	RW800	COVER	METAL S / M / L	2944	SF520		
5656506565	RW	RW800	SEED PLATE		82	SF845		
Please contact product line	RW	RW800	SEED PLATE		82	SF900		
Please contact product line	RW	RW800	SEED PLATE		130	SF845		
35678241232	RW	RW800	SEED PLATE		130	SF900		
56454215455	RW	RW800	SEED PLATE	METAL S / M / L	251	SF845		

Kuva 11. CSS-tyyleillä täydennetty HTML-taulukko

HTML-taulukon CSS-ominaisuuksiin voi lisätä myös ominaisuudeksi ”width: 100%”, jolloin taulukko suhteellistuu leveydeltään selaimen kokoon automaattisesti ja taulukko on koko selaimen näkymän kokoinen. Tällaisen ruudun kokoon suhteuttamisen voi viedä askeleen pidemmälle hyödyntämällä CSS3:n ”media query”-ominaisuutta. Tätä ominaisuutta käyttämällä voidaan asettaa eriäviä tyylejä, jotka kohdennetaan tietylle mediatyypille. Kaikki modernit webselaimet tukevat tätä ominaisuutta, myös mobiiliselaimet mukaan lukien. (Van Hove 2015.)

Esimerkiksi, ”media query”-ominaisuudella voidaan havaita, että käyttäjä on navigoinut sivulle mobiililaitteellaan, jolloin tyyleiksi valitaan mobiililaitteen mediatyypille asetetut tyylisäännöt, yleensä pöytäkoneille säädettyjen, oletustyylien sijaan. Näillä ”query”-asetuksilla haetaan tietoa esimerkiksi käytetyn laitteen leveydestä ja käyttöasennosta, eli onko laite pysty- vai vaaka-asennossa käyttäjän navigoidessa sivulle. Näitä asetuksia ei tarvitse itse kuitenkaan selvittää vaan esimerkiksi CSS-Tricks –niminen internetyhteisö tarjoaa tarvittavat CSS-koodipätkät erilaisten laitteiden kohdentamista varten vaikkakin sen kokoelma ei ole täydellinen. Sivulta löytyy eri iPhonet ja iPadit, eri Galaxy-laitteet ja monien muidenkin valmistajien laitteita (CSS-Tricks 2015). Etenkin nyt kun mobiililaitteiden kirjo on valtava ja markkinoille on tullut myös erittäin pieniruutuiset älykellot, voi ”media query”-tyylit olla todella hyvä tapa lisätä oman sovelluksen käyttömukavuutta kaikille eri laitteiden käyttäjille eikä mukautua vain pöytäkonekäyttäjien tarpeisiin.

5.4 Mobiilialustan File Explorer-hallinta

Kun sovellukseen oli saatu pääpiirteiset toiminnot, oli aika testata sitä tulevalla kohdealustalla, eli mobiililaitteella. Käytettävissä oli Samsung Android-käyttöjärjestelmän kosketusnäyttöinen puhelin, ja ensimmäisenä testausselaimena Firefox. Sovellus siirrettiin testauslaitteeseen ja CSV-testitiedosto siirrettiin SD-muistikortille.

Jo testausvaiheen alussa törmättiin alustojen erilaisuudesta johtuvaan ongelmaan. Tiedostonhaku ei ollutkaan mobiililaitteella aivan yhtä yksinkertaista kuin pöytäkoneympäristössä. Tiedostonvalintanappia painettaessa

avautui eräänlainen mediatiedostojen selaukseen tarkoitetun sovelluksen (Media Storage) näkymä, jolla saattoi selata puhelimen muistia erittäin rajatusti. Ainoat näin ladattavissa olevat tiedostot olivat lähinnä kuvatiedostoja, ja suoranaista pääsyä SD-kortille ei ollut.

Oli etsittävä vaihtoehtoisia ratkaisuja. Yksi ensimmäisistä ehdotuksista oli lukea ohjelmaan automaattisesti CSV-tiedosto ilman käyttäjän valintaa. Pöytäkoneympäristössä tällainen tiedoston lukeminen ohjelman koodissa annetusta URI-sijainnista (Uniform Resource Identifier) ei ole lainkaan harvinaista sovelluksien käytössä. Mobiiliympäristössä tämä ei ole kuitenkaan aivan yhtä helppoa.

Otetaan esimerkiksi käytetty testialusta, Android-käyttöjärjestelmä. Kun sovelluksissa käsitellään mobiililaitteiden muistissa olevia tiedostoja, sovelluksien luvat käyttää järjestelmää pitää olla asetettu. Natiivisovelluksissa nämä luvat annetaan Android-manifestissa, joka on XML-tiedosto sovelluksen juurihakemistossa. (Developer Android 2015.) Puhtaasti web-tekniikoihin pohjautuvassa sovelluksessa tällaista tiedostoa ei ole eikä sovellukseen siten voi asettaa lupaa etsiä tiettyä tiedostoa puhelimen muistista, sisäisestä tai ulkoisesta.

Lupien lisäksi tässä ratkaisumallissa oli myös käytännöllinen ongelma. Asiakasyrityksessä ei ollut yhtä yhtenäistä tiedostoa katalogisovellusta varten vaan näitä CSV-tiedostoja saattoi olla useita kappaleita, joissa kaikissa oli eri tuotteiden tiedot. Vaikka tietyn tiedoston automaattinen hakeminen mobiilijärjestelmissä olisikin mahdollista, sovelluksen käyttö rajoittuisi melkoisesti, jos sillä pystyisi lukemaan vain yhden tiedoston, tai vaihtoehtoisesti käyttäjän pitäisi itse hallita tiedostoja saadakseen ohjelmaan eri luettelon luettavaksi. Jo ideatasolla, tämä automaattinen luku ratkaisuna oli siten käyttökelvoton.

Ainoa jäljellä oleva vaihtoehto oli tutkia enemmän keinoja, miten saada käyttäjän tiedoston valinta onnistumaan. Ensimmäinen kysymys oli, onko ”input type=file” –kenttä edes tuettu mobiilialustalla vai pitikö tälle kehittää kokonaan uusi ratkaisu. Ongelma ei onneksi ollut täysin vieras kehittäjien keskuudessa.

Viljami Salminen (2012) selvitti ”File Support on Mobile”-blogimerkinnässään ”input type=file” –kentän toimivuutta lukuisissa erilaisissa mobiililaitteissa. Testeissä käytettiin laitteiden oletusselaimia, ja tulokset on luokiteltu käytettyjen käyttöjärjestelmien mukaan. Tästä päätellen tukeen voi vaikuttaa myös laitteen käyttöjärjestelmän versio, ei pelkästään laitteen internetselain, sillä uudemmat selaimet eivät välttämättä toimi vanhemmissa käyttöjärjestelmissä. Taulukossa 1 on koottuna blogimerkinnästä saadut tiedot tämän työn kannalta tutkituissa tärkeimmissä käyttöjärjestelmissä.

Taulukko 1. ”Input type=file” –kentän tuki eri käyttöjärjestelmissä

Testattu käyttöjärjestelmä	Onko ”input type=file” tuettu?
Android 1.0 – 2.1	Ei
Android 2.2 +	Kyllä
iOS 1.0 - 5.11	Ei
iOS 6.0 +	Kyllä
Windows Phone 7.0 – 8.0	Ei
Windows Phone 8.1	Kyllä
Windows RT	Kyllä

Näiden tietojen perusteella voi sanoa, että melko uusien laitteiden ja selainten tuki tälle kentälle on aika hyvä. Käytössä oleva Android-laite kuului tuen piiriin, joten tästä päätellen tiedostonvalinnan pitäisi onnistua jollain tavalla. Salmisen selvityksessä oli kuitenkin vielä yksi ongelma katalogisovelluksen koodin kannalta. Blogissa esitetyssä koodista käy ilmi, että testissä selvitettiin vain ”input type=file” –kentän toimivuutta kun ladataan kuvia. Kuva- ja video-tiedostojen lataaminen on hyvin yleinen toiminnallisuus mobiililaitteissa mutta ei kuitenkaan vastaa sovelluksen tarvetta. Vaikka tieto tiedostonvalintakentän tukemisesta helpottikin tilannetta, täytyi asiaa tutkia edelleen.

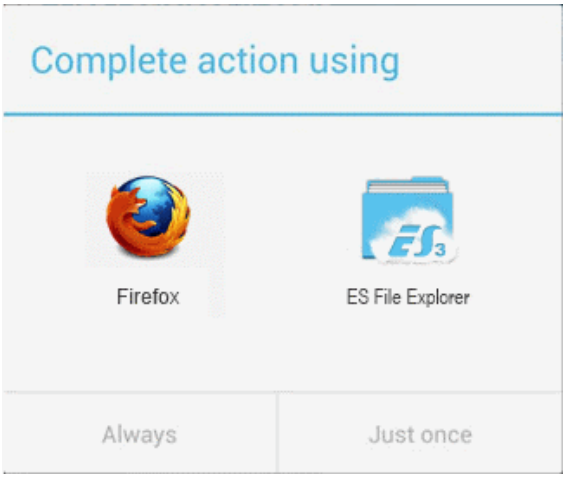
Asiaa lähestyttiin uudelleen käyttäjän kannalta ja ongelmaan etsittiin ratkaisuja hakemalla tietoa internetin keskustelupalstoilta. Monet muutkin kehittäjät ovat joutuneet kamppailemaan erilaisten tiedostotyyppien lataamisten kanssa, mutta onneksi tähän ongelmaan tarjottiin myös ratkaisuja.

Ensimmäinen ehdotus oli selaimen vaihto Opera for Android-selaimeen, ja ainoa tapa arvioida ehdotuksen toimivuus oli testata itse. Kyseinen selain asennettiin Googlen Play-kaupan kautta ja sovellusta testattiin uudestaan. Tiedostonvalintanappia painettaessa päästiinkin yllättäen tutkimaan sen kansion tiedostoja, mistä sovellus oli avattu. Tämä ei täysin vastannut alkutilannetta, jossa CSV-tiedosto oli SD-kortilla. Pilvipalveluun perustuvan varastosovelluksen avulla pystyttiin kuitenkin lataamaan testitiedosto ja sovellus samaan kansioon. Uudelleen testaamalla tiedostonvalintaa, saatiin haettua haluttu CSV-tiedosto samasta kansiosta. Sovellus myös toimi moitteettomasti. Sovellus luki CSV-tiedoston suhteellisen nopeasti ja taulukon tiedot suodattuivat painettujen nappien mukaan.

Vaikka selaimen vaihto vaikuttikin ratkaisevan ongelman, oli silti tarpeellista kokeilla toista löydettyä ratkaisuehdotusta. Koska Chrome ja Firefox ovat Android-alustalla keskimääräisesti suositumpia selaimia kuin Opera, olisi harmi, jos sovellus ei toimisi myös niillä.

Toinen ratkaisuehdotus oli tiedostonhallintasovellusten käyttö. Nämä ovat sovelluksia, joiden avulla käyttäjä pystyy hallitsemaan mobiililaitteen muistissa olevia tiedostoja, kuten poistamaan, uudelleennimeämään ja siirtämään niitä aivan kuten pöytäkoneella. Lisäksi, näiden sovellusten avulla käyttäjä pääsee näkemään erilaisten tiedostojen sijainnin laitteellaan. Tällaisia hallintasovelluksia on saatavissa melkoinen määrä Google Play-sovelluskaupassa Android-käyttöjärjestelmälle.

Tämän tekniikan testaamiseen käytettiin ilmaista ES File Explorer File Manager -nimistä tiedostonhallintasovelluspakettia lähinnä siksi, että sen käyttöä suositeltiin moneen otteeseen ongelman ratkaisuksi. Sovellus asennettiin, ja katalogisovelluksen tiedostonvalintanappia testattiin uudelleen Firefox-selaimella. Tällä kertaa näytölle saatiin mahdollisuus valita, millä ohjelmalla toiminto suoritetaan, ja vaihtoehtona kuvan 12 mukaisesti oli juuri asennetun paketin ES File Explorer -sovellus. Valitsemalla tämän sovelluksen, päästiin etsimään oikeaa tiedostoa laitteen muistista melko helposti. Varsinaisen tiedoston katalogisovellus luki jälleen melko nopeasti, ja toimi myös halutulla tavalla suodattamisen kanssa.



Kuva 12. Näkymä, josta oli mahdollista valita tiedostonvalintaa varten hallintasovellus

Tietenkin oli harmillista, ettei sovellusta ilmeisesti saanut toimimaan kuin tietyllä selaimella tai vaihtoehtoisesti käyttämällä kolmannen osapuolen tiedostonhallintasovellusta mutta mobiilialustalle ominaiset rajallisuudet on vain huomioitava tässä suhteessa.

Loppujen lopuksi katalogisovellus kuitenkin toimi melko sujuvasti Android-alustalla. Mutta entä iOS- tai Windows Phone-käyttöjärjestelmä? Kuten aiemmin tuli mainittua, Apple on ollut haluamaton antamaan käyttäjille vapaata pääsyä mobiililaitteen tiedostoihin turvallisuussyistä ja tämä on suurin ongelma iOS-laitteiden suhteen tämän sovelluksen kohdalla. Sitten iOS6-käyttöjärjestelmän version Safari-selain tukee kyllä ”input type=file”-kenttää mutta sallii vain kuvien ja videoiden latauksen. Muita tiedostotyyppejä ei tueta. (Firtman 2012.) Applen iTunes-sovelluskaupasta löytyy kyllä muutamia File Explorer-sovelluksia mutta nämä eivät ole samanlaisia kuin Android-alustalle tarkoitetut tiedostonhallintasovellukset, joilla näkee tiedostojen sijainnin muistissa.

Windows Phone-alustalla vaikuttaisi olevan sama rajoitus. Vaikka uudemman käyttöjärjestelmän 8.1-version Internet Explorer 11 tiedostonvalintakenttää tukeekin, silläkään ei voi ladata muita tiedostoja kuin kuvia (Mohta 2014). Windows Phone 7.0 - 8.0 -järjestelmissä koko tukea ei edes ole. Lisäksi Microsoftin julkaisema tiedostonhallintasovellus ”Files” ei myöskään tarjoa samanlaista ratkaisua kuin samanlaiset sovellukset Android-alustalla.

Sillä voi vain hallita tiedostoja tietyissä sijainneissa mutta se ei tarjoa apua tiedostojen lukemiseen web-sovelluksen kautta.

5.5 Sovelluksen arviointi

Tehdyn työn perusteella voi huomata, että web-tekniikoiden käyttö on varsin toimiva ratkaisu tämän kokoisen sovelluksen tekoon. Valmis versio katalogisovelluksesta toimi todella hyvin, vaikka varsinainen tiedoston luku vaatikin lopulta ulkopuolisia ratkaisuja toimiakseen. Tästä huolimatta, sovellus luki pitkänkin luettelotiedoston melko nopeasti ja tuotti suodattavat napit aivan kuten haluttiin poimien aina CSV-tiedoston kolumneista eriävät alkiot napeiksi. Suodattimet toimivat myös halutusti ja nopeasti. Taulukosta poistettiin vain ne tiedot, jotka eivät kuuluneet käyttäjän valitseman tuotetyypin ominaisuuksiin. Vastaavasti sovellus myös toi tiedot napakasti takaisin näkyville, kun käyttäjä valitsi aiemmista napeista jonkin uuden luokan, ja poisti tarpeettomat napit ja aiemmin tuotetun kulkupolun tekstilaatikot.

Myös käyttöliittymä palveli sovelluksen käyttötarkoitusta. Ensin löytyvät napit, ja vasta sitten suodattamisen myötä muuttuva taulukko. Taulukon ulkonäkö suunniteltiin niin, että se olisi mahdollisimman selkeä ja miellyttävän näköinen selata. Joitakin tietosarakkeita tosin piilotettiin näkyvistä mobiililaitteiden näytön rajallisuuden vuoksi, esimerkiksi tuotekuvat. Tämä siksi, että vain oleellisimmat tiedot korostuisivat.

Sovelluksen haku oli yksinkertainen mutta palveli tarkoitustaan. Juuri yksinkertaisuutensa takia se etsi syötetyn tekstijonon erittäin nopeasti ja piilotti taulukosta heti ne alkiot, jotka eivät vastanneet syötettyä merkkijonoa. Myös se, ettei käyttäjän tarvitse painaa erikseen enter-näppäintä haun aloittamiseksi ja odottaa tuloksien latautumista, on pieni mutta käyttöä nopeuttava yksityiskohta.

Sovelluksessa oli kuitenkin muutamia puutteita. Esimerkiksi käyttäjän valinnoista muodostuvan polun esittäminen olisi voitu toteuttaa toisin. Nappien alle luotaviin tekstilaatikoihin päädyttiin lähinnä aikaresurssien rajallisuuden takia. Periaatteessa toimivin ratkaisu olisi ollut värjätä valitut napit poikkeavasti muista napeista, jotta käyttäjä olisi näin pystynyt näkemään saman, mikä nyt toteutettiin tekstilaatikoiden avulla.

Toinen piirre, joka jouduttiin jättämään pois sovelluksen lopullisesta versiosta, oli mahdollisuus piilottaa tiettyjä taulukon sarakkeita. Ajatuksena oli, että jos käyttäjä ei ole käyttöhetkellä kiinnostunut niinkään tuotteiden painosta tai materiaalista, hän voisi joko napauttaa sormellaan tätä saraketta, jolloin se poistuisi näkyvistä, tai vaihtoehtoisesti valita valintaruutujen kautta, mitkä sarakkeet näkyvät ja mitkä eivät. Tämän ominaisuuden ajateltiin tuovan nopeutta ja selkeyttävän taulukkoa edelleen. Ongelmana tämän ominaisuuden toteutuksessa oli kuitenkin se, että taulukon tiedot muuttuivat jatkuvasti käyttäjän valintojen myötä, mikä toi omat ongelmansa molempien ratkaisuvaihtoehtojen kohdalla. Sarakkeita napsauttaessa kohteena olevat tiedot kyllä katosivat halutusti mutta jos taulukon suodattamista jatkettiin tämän jälkeen, piilotetut tiedot ilmestyivät takaisin taulukon päivittyessä ja hajottivat sen rakenteen kokonaan. Valintaruuduilla taas ominaisuus

saatiin toimimaan halutulla tavalla, tosin suorituskyvyn kustannuksella. Kun tätä ominaisuutta testattiin pöytäkoneella, käytetty testiselain hyytyi täysin ja vasta pitkän tovin kuluttua näytti päivitetyn taulukon. Mobiililaitteilla tilanne olisi luultavasti vielä pahempi. Siten, vaikka ominaisuus toimikin, se ei olisi palvellut käyttötarkoitusta vaan suorastaan huonontanut sitä. Paremman suorituskyvyn takia, tämä ominaisuus jätettiin pois.

Kolmas asia, mikä valmiin sovelluksen kohdalla oli harmillista, oli käyttöjärjestelmien tuki. Web-tekniikoiden ehdottomasti vahvin puoli, alustariippumattomuus, jäi hyödyntämättä, koska sovelluksen tiedostonlukua ei yksinkertaisesti pysty toteuttamaan iOS- ja Windows Phone-laitteilla, sillä järjestelmien rajat tulevat vastaan. Ennen kuin Apple ja Microsoft muuttavat alustojensa tukea niin, että myös mobiililaitteilla pystyttäisiin lataamaan muitakin tiedostotyyppejä kuin kuvia tai videoita, sovellus toimii vain Android-järjestelmän laitteilla. Pelkän sovelluksen koodin kautta tähän ei pystytä vaikuttamaan.

Siten, vaikka sovellus toimikin hyvin, on siinä vielä jonkin verran jatkokehityksen varaa ja joitakin ominaisuuksia voisi parantaa nykyisistä ratkaisuista. Erityisen kiinnostavaa olisi saada sovellus toimimaan Applen ja Microsoftin laitteilla, mikäli näihin käyttöjärjestelmiin joskus mahdollisuus tämän sovelluksen tarvitsemaan tiedostonvalintaan tulisi.

6 VAIHTOEHTOISET RATKAISUT

Katalogisovellus toteutettiin web-tekniikoita käyttäen, koska asiakasyrityksen edustaja näin ehdotti. Sen tekoon olisi ollut kuitenkin myös vaihtoehtoisia ratkaisuja, joita voisi pohtia tarkemmin. Tällaisia ratkaisuja ovat esimerkiksi natiivisovelluksen teko, WebView-näkymän käyttö tai PhoneGap ja Xamarin -ohjelmien kaltaiset mobiilisovellusten tekoon tarkoitetut koodiympäristöt, joita on näiden kahden lisäksi lukematon määrä.

6.1 Natiivisovellus

Natiivisovelluksella tarkoitetaan sovellusta, joka tehdään mobiililaitteen käyttöjärjestelmälle tarkoitetulla ohjelmointikielellä. Android-käyttöjärjestelmän laitteilla se on Android Java, iOS-laitteilla Objective-C ja niin edespäin. Sovellukset koodataan käyttämällä alustalle tehtyjä sovellustyökaluja yhdistettynä johonkin koodin kehitysympäristöön, esimerkiksi Android-sovelluksien tekoon Android Developers-sivustolta löytyy Android SDK (Software Development Kit), jonka voi yhdistää muun muassa Eclipse-kehitysympäristöön.

Käyttöjärjestelmälle ominaisen ohjelmointikielen käyttö tuo sovellukseen paljon etuuksia, joita muilla käyttötavoilla ei saavuteta. Suorituskyky ja nopeus ovat näistä kaikkein ilmeisimmät vahvuudet. Tämä siksi, koska mobiilialustojen omat ohjelmointikielet perustuvat vahvempiin ohjelmistokieliin, jotka alun perin suunniteltiin monipuolisia ohjelmia varten, kuten Android Java.

Natiivit sovellukset pystyvät myös hyödyntämään mobiililaitteen laitteellisia ominaisuuksia, kuten kameraa tai GPS-laitetta. Lisäksi natiivin sovelluksen käytössä voidaan hyödyntää alustakohtaisia eleitä, joita mobiililaitteiden käyttäjät ovat tottuneet luonnostaan käyttämään. Natiiveja sovelluksia voidaan myös käyttää offline-tilassa kun verkkoa ei ole saatavilla. (Budiu 2013.)

Kun mietitään työstettyä katalogisovellusta, suorituskyky ja nopeus ovat avainsanoja, jotka ratkaisevat. Silti on pohdittava, ovatko edut niin ratkaisevia, että mobiilialustakohtaista kieltä olisi kannattanut hyödyntää tämänkin sovelluksen kohdalla ehdotettujen web-tekniikoiden sijaan? Tuskin. Nykyisissä selaimissa on todella nopeat JavaScriptiä suorittavat moottorit ja niiden kehitys on jatkuvaa. Lisäksi, sovellus ei ole niin raskas ja moninainen eivätkä luettavat tiedostot niin jättimäisiä datamääriltään, että sen suorituskyvystä saisi mainittavia etuja optimoiduilla ohjelmistokielillä.

Sovelluksessa ei myöskään ole tarvetta päästä käsiksi mobiililaitteiden laitteellisiin ominaisuuksiin, kuten kameraan. Ainoa tarpeellinen osa on mobiililaitteen muistissa olevat tiedostot, joiden löytämiseen ja lukemiseen tietenkin jouduttiin pohtimaan erillisiä ratkaisutapoja. On kuitenkin oma lukunsa, miten sama ongelma olisi ratkaistu natiivisovelluksen rakentamisessa. Ehkä ongelma olisi ratkennut helpommin, koska periaatteessa sovelluksen Android-manifestissa voidaan antaa sovellukselle lupa käyttää esimerkiksi laitteen SD-muistikorttia.

Eleiden käyttö ei ollut myöskään katalogisovelluksessa oleellisimpia pohdinnan aiheita, mutta mikäli natiivisovellusta olisi lähdetty tekemään lähtökohtaisesti, olisi niitä voinut hyödyntää esimerkiksi pyyhkäisy-elettä erillisen haun toteutuksessa.

Entä sitten offline-tilassa toimivuus? Kieltämättä se ominaisuus on edelleen natiivisovelluksen hyödyllisimpiä piirteitä mutta HTML5:n myötä myös web-sovelluksiin on tullut mahdollisuus toimia offline-tilanteessa. HTML5:n kautta tämä tapahtuu sovelluksen välimuistin tai uusien offline-varaston API-liittymien kautta, esimerkiksi näitä ovat Web Storage ja File Storage. On kuitenkin huomioitava, etteivät kaikki selaimet, varsinkin mobiiliversiot näistä selaimista, välttämättä vielä tue kaikkia ominaisuuksia, joita näiden hyödyntäminen vaatii. (Mahemoff 2010.)

Ominaisen ohjelmointikielen käyttö poistaa samanaikaisen ristiinkehitysmahdollisuuden. Koodi joudutaan kirjoittamaan alustansiirrossa aina uudestaan toisella kielellä, esimerkiksi Android-käyttöjärjestelmästä iOS-laitteille siirryttäessä joudutaan vaihtamaan Android Java Applen laitteille ominaiseen Objective-C-kieleen. Usein pelkkä suora ”kääntäminen” ei riitä vaan joissakin ongelmatilanteissa joudutaan pohtimaan erilaisia ratkaisuja uuden kohdealustan ominaisuudet huomioiden.

Natiivisovellus on siis hyvä ja vahva vaihtoehtoinen ratkaisu sovelluksen kehitykselle ja yleensä se ensisijainen ratkaisumalli. Kuitenkin HTML5:n myötä, web-tekniikat kehittyvät koko ajan ja tarjoavat jo paljolti vastaavanlaisia ominaisuuksia kuin natiivit ratkaisut.

6.2 WebView-näkymä

WebView-näkymä on uniikki luokka, jolla mobiilialustan natiivisovellukseen voidaan upottaa web-sisältöä. Tällainen ominaisuus löytyy Android-, Windows Phone- ja iOS-alustasta, jossa sitä kutsutaan UIWebView:ksi vaikka toimintaperiaate on kuitenkin sama.

Kaikilla kolmella alustalla näkymän käyttö on melkein samanlaista. Esimerkiksi Android-alustalla, WebView-näkymä upotetaan osaksi natiivia sovellusta ja tässä näkymässä näytettävä web-sisältö määritetään koodissa komentamalla sovellus lataamaan sisältö tietystä internet-osoitteesta. Android-manifestissa on myös asennettava sovellukselle lupa käyttää internet-yhteyttä sisällön hakemista varten. (Android Developers 2015.) Vaihtoehtoisesti sovellukseen voidaan sisällyttää koodausvaiheessa erillinen HTML-tiedosto sovelluksen resurssikansioihin, joka sitten ladataan tähän näkymään. WebView-näkymä pystyy myös suorittamaan JavaScript-koodia, tosin se täytyy asettaa sallituksi sovelluksen asetuksissa.

Ajatus WebView-näkymän käytöstä on kuitenkin ihanteellisempi ajatustasolla kuin varsinaisessa toteutuksessa. Se on hyvä sovellusta täydentävässä tarkoituksessa, esimerkiksi jos verkkosivulla on sovelluksen käyttöön liittyviä, usein päivitettäviä tietoja, joista käyttäjän olisi hyvä olla tietoinen, voi WebView-näkymästä olla paljonkin hyötyä. Kokonaan web-sisältöön pohjautuva sovellus taas ei välttämättä ole parhaimmillaan tällaisessa näkymässä. Näkymä upotettuna natiivisovellukseen ei esimerkiksi takaa yhtä hyvää suorituskykyä ja nopeutta kuin puhtaalla kohdealustan omalla ohjelmakoodilla saataisiin, sillä web-tekniikoin toteutettu sovellus on edelleen riippuvainen käytetyn selaimen ominaisuuksista koodin suorittamisessa.

WebView-näkymän kanssa on myös huomioitava, että vaikka sillä pystyykin esittämään web-sisältöä, se ei kuitenkaan ole suoranainen internetselain eikä siten aivan yhtä tehokas. Voi olla myös, että jotkin ominaisuudet, jotka toimivat kunnollisella selaimella, eivät välttämättä toimi WebView-näkymässä. Sellaisia sivuja, joiden tarjoama web-sisältö on riippuvainen käyttäjän aktiivisesta interaktiosta, ei kannata rakentaa WebView-näkymän varaan. (Android Developer Reference 2015.)

Varsinkin Android-alustalla, WebView-näkymä on yksi niistä ominaisuuksista, joiden kohdalla kannattaa aina huomioida kohdealustojen versiot ja sovelluksen todelliset käyttäjät. WebView-näkymä on jo pitkän aikaa ollut turvallisuuden kannalta altis verkkohyökkäyksille johtuen pohjana olevan WebKit-moottorin haavoittuvuuksista. Moottorista johtuvista turvallisuusongelmista johtuen Google on päättänyt lopettaa Android 4.3 ja sitä aikaisempien versioiden turvallisuuspäivitykset WebView-näkymään liittyen. (Ludwig 2015.) Android 4.4-versiosta lähtien on mahdollisuus käyttää täysin uudenlaista WebView-näkymää, joka perustuu Chromium-lähdekoodiin ja tuo siten enemmän turvallisuutta ja myös enemmän oikean selaimen kaltaista tukea HTML5-, CSS3- ja JavaScript-tekniikoiden käyttöä varten (Android Developers 2015). Silti, ottaen huomioon, että Android 4.3-versio julkaistiin vuonna 2012, tätä käyttöjärjestelmää tai sitä aikaisempia versi-

oita käyttäviä laitteita on vielä paljon, ja siten ne eivät välttämättä saa käyttöönsä uudempaa käyttöjärjestelmän versiota ja WebView-näkymän päivitykset jäävät niiden ulottumattomiin.

WebView-näkymän käyttö tämän katalogisovelluksen suhteen oli kyllä yksi vaihtoehdoista, joita harkittiin toteutusvaiheen alussa. Se vaihtoehto suljettiin kuitenkin myöhemmin pois, koska WebKit-moottorin tarjoama selain ei vaikuttanut tarpeeksi kehittyneeltä sovelluksen käyttöön. Lähinnä tiedostonvalinta olisi koitunut suureksi ongelmaksi, koska se oli hankalaa jo kehittyneemmillä, oikeilla web-selaimilla kuten Chromella ja Firefoxilla. Lisäksi ongelmana oli suositeltu käyttö. Android Developers-oppaassakin kehotetaan, että WebView-näkymää käytettäisiin vain osana sovellusta ja käyttäjän interaktio sivun sisältöön minimoitaisiin. Tämä oli kuitenkin täysin päinvastainen tilanne katalogisovelluksessa, koska koko sovelluksen toiminta perustui web-tekniikoihin ja käyttäjän oli tarkoitus koko ajan muokata, mitä sovelluksen sisältöä näytetään. WebView-näkymän käyttö ei siten ollut tämän sovelluksen kannalta paras vaihtoehto.

WebView-näkymän käyttöönotossa pitää siis huomioida melkoinen määrä asioita ennen kuin lähtee sen varassa tekemään sovellusta. Aina on huolehdittava siitä, että valittu toteutusratkaisu palvelee sovelluksen käyttötarkoitusta. WebView-näkymän kehittyminen uusien päivitysten johdosta tuo jo melkoisen kehitysharppauksen sovellusten kehittäjille vaikkakin vain Android 4.4-versioiden käyttäjille. Toisaalta taas WebView-näkymän käyttöönotossa pitää edelleen pohtia, miten huomioida ne käyttäjät, joilla on sitä vanhempi käyttöjärjestelmän versio.

6.3 PhoneGap

Nykyään Adoben omistama PhoneGap on avoimen lähdekoodin sovelluskehitysympäristö, joka perustuu Apache Cordovaan. Sillä voi tehdä sovelluksia mobiililaitteille käyttämällä HTML-, CSS- ja JavaScript-tekniikoita. Se on web-tekniikoita käyttävien kehittäjien keskuudessa yksi suosituimmista kehitysympäristöistä (Developer Economics 2013).

PhoneGapista löytyy kehitystyökalut monelle eri mobiilialustalle. Yleisimpien käyttöjärjestelmien, iOS-, Android- sekä Windows Phone-käyttöjärjestelmien lisäksi tukipiiriin kuuluu myös vähemmän yleisempiä mobiilialustoja kuten BlackBerry ja Tizen.

Kehitysympäristön toimintaperiaate on seuraavanlainen: PhoneGap esittää web-tekniikoin toteutetun webdokumentin natiivisovelluksessa hyödyntäen WebView-näkymää. PhoneGap-kehitysympäristön käyttö kuitenkin mahdollistaa laiteominaisuuksien hyödyntämisen sovelluksessa natiivisovelluksen tapaan, kuten laitteen kameran ja kiihdytysanturin. PhoneGapin kattavuutta laiteominaisuuksiin voidaan myös jatkaa plugin-paketeilla. (Traeg 2014.) Koska PhoneGap perustuu avoimeen lähdekoodiin, voi sovelluskehittäjä kehittää myös omat plugin-pakettinsa, jos ei löydä omaan tarpeeseen sopivaa pakettia jo valmiiden pakettien joukosta.

PhoneGapissa voi työskennellä kahdella eri kehityspolulla. Ensimmäinen polku on alustakeskeinen tapa, jossa sovellus tehdään tietylle alustalle. Tällöin kehittäjän on valittava kohdealusta ja asennettava työpisteelleen PhoneGapin lisäksi halutun alustan sovelluksentekotyökalut, esimerkiksi Android-kehitystä varten tarvitaan Android SDK -paketti. Koodin kirjoituksessa hyödynnetään toista kehitysympäristöä, jonka päällä PhoneGap toimii. (PhoneGap 2015.)

Toinen polku on ristiinkehitystapa, jolla sama sovellus kehitetään monelle eri alustalle käyttämällä PhoneGapin Cordova Command Line Interface-työkalua. Ristiinkehitys mahdollistetaan asentamalla työpisteelle myös kaikkien kohdealustojen SDK-paketit. Komennot syötetään suurimmaksi osaksi komentoriviltä. (PhoneGap 2015.)

PhoneGapin käytössä on omat hyvät puolensa. Joillakin kehittäjillä voi olla vankka tuntemus web-tekniikoista mutta mobiilialustalle ohjelmointi voi olla aivan vieras käsitys. Sellaisille kehittäjille PhoneGap-kehitysympäristö voi olla natiivisovelluksia helpompi aloitusympäristö, jos halutaan luoda esimerkiksi sovellus, joka vaatii kameran käyttöä. PhoneGapin avulla tällainen onnistuu melko helpostikin, mutta puhtaasti web-tekniikoihin perustuvassa sovelluksessa ei ainakaan toistaiseksi.

PhoneGapin suurin heikkous on suorituskyky. Vaikka PhoneGapilla on mahdollista käyttää laiteominaisuuksia niin kuin natiivisovelluksissa, se on periaatteessa kuitenkin WebView-näkymä käärittynä natiivisovellukseen.

Lisäksi PhoneGapin käyttöönotto on hankalaa. Lähinnä isoin kysymysmerkki nousee tarvittavasta kehitysympäristöstä PhoneGapin käyttöön. Toisin kuin Android-sovellukset, joita voi kehittää sekä Mac- että Windows-ympäristössä, Applen iOS-laitteet tarvitsevat Mac-käyttöjärjestelmän kehitystä varten ja tämä koskee myös PhoneGapin käyttöä. Windows Phone 8 tarvitsee taas Microsoftin Windows 8-käyttöjärjestelmän. Myös Windows Phone 8.1 lisää tarvittavia ominaisuuksia käytettävästä ohjelmistosta. (PhoneGap 2015.) Tämä ei ole niinkään ongelma, jos kehitetään sovellusta vain yhdelle alustalle mutta jos käytetään ristiinkehitystapaa, voi siitä tulla ongelma. PhoneGapin dokumentit ovat myös hieman sekalaisia, erityisesti näissä kehitysympäristöjä koskevissa selityksissä. Luettavaa materiaalia on paljon, ehkä jopa liikaakin. Asennettavaa tavaraa riittää, mikä ei ole kehitysympäristössä kovinkaan puoleensavetävä piirre.

Eräs huomioitava, hyvä puoli PhoneGapin käytössä on PhoneGap Build, pilvipalveluversio kehitysympäristöstä. Sen toiminta on hieman erilainen. Kehittäjä lataa tekemänsä HTML-, CSS- ja JavaScript-tiedostot pilvipalveluun ja tämä tekee niistä automaattisesti paketin halutuille alustoille. Koska palvelu toimii pilvessä, ei kehittäjän tarvitse itse hallita SDK-koodauspaketteja tai kehitysympäristöjä, koska palvelu kokoaa sovelluksen automaattisesti ja tarjoaa siitä valmiin version. Palvelusta on ilmainen ja kuukausimaksullinen versio, ja ilmaisessa versiossa odotetusti on rajoituksensa. Esimerkiksi maksimisovelluskoko on 50 megatavua kun maksullisessa versiossa se on 100 megatavua. (Adobe PhoneGap Build 2015.)

Sovelluskatalogin kohdalla PhoneGapin käyttö ei tuntunut tarpeelliselta. WebView-näkymän käyttö oli jo aiemmin suljettu pois vaihtoehdoista, joten PhoneGap ei vaikuttanut loogiselta käyttötavalta. Sovelluksella ei ollut tarvetta käyttää laiteominaisuuksista kuin tiedostojärjestelmää, mutta ei vaikuttanut siltä, että PhoneGap tekisi sen käsittelystä yhtään mutkattomampaa. Lisäksi, koska PhoneGapin käytöstä tuskin olisi seurannut suorituskyvyllisiä parannuksia, tuntui käytetty, puhtaasti web-tekniikoihin pohjautuva ratkaisu paremmalta vaihtoehdolta.

PhoneGap on siten hyvä vaihtoehto sellaiselle kehittäjälle, joka on tottunut käyttämään web-tekniikoita. Se kärsii silti heikommasta suorituskyvystä natiivisovelluksiin verrattuna. Lisäksi kehittäjä voi joutua pohtimaan, millaisen kehitysympäristön hän tarvitsee riippuen siitä, kuinka monelle eri mobiilialustalle hän haluaa sovelluksensa tuoda. Toisaalta pilvipalvelu PhoneGap Build osoittaa jo, että PhoneGapin jatkokehitystä voi olla mielenkiintoista seurata myös tulevaisuudessa.

6.4 Xamarin

Xamarin on samannimisen yhtiön sovelluskehitysympäristö. Kehitys tapahtuu Microsoftin kehittämällä C#-kielellä, joka kuuluu .Net-ohjelmointikieliin. Xamarinilla on mahdollista kehittää sovelluksia joko yksittäiselle alustalle tai ristiinkehitystavalla kattaen iOS-, Android- ja Windows Phone-käyttöjärjestelmät.

Xamarin tarjoaa oman kehitysympäristönsä, Xamarin Studion, mutta Windows-alustalla kehittäjä voi käyttää myös Microsoftin Visual Studio-kehitysympäristöä, johon on liitetty Xamarin plug-in. Xamarin kuitenkin mahdollistaa kaikkien ohjelmointiin tarvittavien pakettien asentamisen laitteelle omalla asennusohjelmallaan vaikka manuaalinen asennuskin on mahdollista. Tosin, vähän samaan tapaan kuin PhoneGap, koko kehitysympäristöllä on jonkin verran vaikutusta siihen, miten Xamarinia käytetään.

Jos kehitysympäristönä on Applen Mac, voidaan tehdä sovelluksia sekä iOS- että Android-laitteille. Tällöin käytettynä ohjelmistoympäristönä on Xamarin Studio, lisäksi tarvitaan molempien kohdealustojen SDK-paketit. Windows-ympäristössä taas voidaan käyttää sekä Visual Studiota tai Xamarin-studiota. Android-sovelluksissa riittää vain SDK-pakettien asennus käytetylle Windows-koneelle, mutta iOS-sovellusten tapauksessa Windows-koneen täytyy olla samassa verkossa Mac-koneen kanssa, ja tässä Mac-koneessa on oltava asennettuna iOS-alustan SDK-paketit. Sovelluksen kääntäminen ja testaus tapahtuu Mac-koneessa, sillä iOS-laitteille voidaan tehdä sovelluksia vain Applen omassa ympäristössä. Xamarinilla ei voi kiertää tätä ominaisuutta. (Xamarin 2015.)

Xamarinissa ei suoranaisesti ole tukea Windows Phone-alustalle, mutta C# on kuitenkin yksi alustan ominaisista ohjelmistokielistä ja sille ohjelmointi onnistuu Visual Studiolla ja Windows Phone SDK-paketilla. Xamarinin plug-in-paketin avulla kuitenkin Windows Phone-sovellukset voivat käyt-

tää samaa koodia kuin Android- ja iOS-laitteet ristiinkehityksessä. Siten saman sovelluksen voi kehittää kaikille kolmelle alustalle käyttämällä samaa koodia.

Ristiinkehitystapoja on kaksi: Jaettu projekti (Shared Projects) ja luokkakirjastot (Portable Class Libraries). Lyhyesti, jaettu projekti tarkoittaa sitä, että Windows Phone-, Android- ja iOS-versiot sovelluksesta käyttävät samoja C#-kooditiedostoja mutta niille voidaan tehdä erilaiset, oman alustansa mukaiset käyttöliittymät. Luokkakirjastoilla taas voidaan luoda kirjasto, joka kohdistetaan niille alustoille, joille sovellus halutaan. Eri sovellusversioissa tätä kirjastoa käytetään sitten erilaisten liittymien kautta, jotka tarjoavat alustakohtaisia toimintoja. Esimerkki tällaisesta liittymästä on SQLite. (Xamarin 2015.)

Toisin kuin web-tekniikoihin pohjautuvat ratkaisut, Xamarin-sovellusten suorituskyky on todella hyvä, koska C# on vahva, oliopohjainen ohjelmointikieli. Lisäksi Xamarin tekee sovelluksista natiivisovelluksia. Kehitysympäristön uusimmalla versiolla pystytään rakentamaan sovelluksia, jotka näyttävät ja tuntuvat alustakohtaisilta sovelluksilta. Esimerkiksi, Android-sovellus näyttää siten hyvin samalta kuin muutkin Android-alustan sovellukset. Lisäksi, Xamarin mahdollistaa myös natiivien sovellusliittymien ja laiteominaisuuksien käytön. (Xamarin 2015.)

Xamarin on kuitenkin maksullinen ratkaisu, aika kalliskin. Kuukausimaksu Indie-versiosta (alle viisi konetta / kehittäjää) on 25 dollaria tai vaihtoehtoisesti vuosimaksu 299 dollaria, jos kehittäjä haluaa mieluummin maksaa vuosittain. Jos ohjelmistoa tarvitaan useammalla kehittäjälle, hinta kiipeää reippaasti. (Xamarin 2015.) Vaikkakin se on maksullinen, se tarjoaa kuitenkin melkoisen määrän ominaisuuksia, joten hinta voi hyvinkin vastata laatua, jos omilla sovelluksilla on tarkoitus tahkoa rahaa.

Juuri tämä hintavuus oli syy, minkä takia Xamarin oli oitis poissuljettu vaihtoehto sovelluskatalogin parissa, koska projektin resurssit eivät olisi antaneet myöten. On kuitenkin epäselvää, olisiko tälläkään ohjelmalla ollut mahdollista ratkaista tiedostonlukua koskevaa ongelmaa etenkin iOS- ja Windows Phone-laitteissa, koska näissä järjestelmissä ei tiedostojärjestelmän käsittelyä käyttäjälle sallita. Silti, yleisellä tasolla Xamarin kyllä vaikuttaa monipuoliselta ja mielenkiintoiselta vaihtoehdolta, varsinkin jos on kokenut C#-kielen käyttäjä.

Xamarin tarjoaa myös Xamarin Test Cloud -pilvipalvelun, joka on tarkoitettu mobiilisovelluksien testauksen helpottamiseen. Sovellusta voi testata tuhansilla erilaisilla mobiililaitteilla pilvessä ilman erilaisten ohjelmien ja emulaattoreiden asentamista koneelle, ja lisäksi testistä voi koota testausraportin, joka sisältää hyödyllistä dataa sovelluksen käytöstä, kuten suorituskyvystä ja muistinkäytöstä sekä ruutukaappauksia testauksen eri vaiheista. Hyödyllisyydestä huolimatta, Xamarin Test Cloud on kallis. Tarjottu perustarjous ”Basic”-maksuohjelma maksaa jopa tuhat dollaria kuukaudessa. (Xamarin 2015.)

Xamarinin käyttö voi olla hyvä vaihtoehto kehittäjälle, joka haluaa hyödyntää C#-kieltä mobiililaitesovelluksessa. Se on kuitenkin melko hintava ratkaisu, mikä ei välttämättä tee siitä hyvää vaihtoehtoa pienemmissä projekteissa.

7 YHTEENVETO

Opinnäytetyössäni käsittelin työharjoittelussa työstämääni projektia. Projektin tarkoituksena oli tehdä mobiiliversio asiakasyrityksessä jo käytössä olevasta pöytäkoneella toimivasta tuotekatalogista. Työn kohdealusta oli Android-käyttöjärjestelmä mutta käsittelin ohessa myös sivuavasti muita suurimpia alustoja, iOS- ja Windows Phone-alustaa.

Sovellus oli tarkoitus tehdä web-tekniikoita käyttäen. Näihin tekniikoihin lukeutuvat merkkauskieli HTML5, JavaScript ja kuvauskieli CSS. Lisäksi apuna käytettiin JavaScriptiä täydentävää jQuery-kirjastoa. Työssä jouduttiin pohtimaan myös erilaisten web-selainten ominaisuuksia, sillä ne ovat niitä ympäristöjä, joissa katalogisovellus tulisi toimimaan.

Asiakasyrityksen alkuperäinen ohjelma toimi pöytäkoneympäristössä. Tässä ympäristössä oli mobiilialustaan verrattuna muutamia oleellisia eroavaisuuksia, jotka piti huomioida työn suunnittelussa. Tärkeimmiksi eroavaisuuksiksi nousi Flash-tuen puute, mobiililaitteiden näyttörajoitukset verrattuna pöytäkoneisiin, sormen käyttö osoittimena ja erilainen tiedostonhallinta.

Lisäksi työn suunnittelussa piti räätälöidä joitakin alkuperäisen ohjelman ominaisuuksia niin, että ne sopivat paremmin mobiilialustan ympäristöön. Suunnittelun jälkeen sovellus voitiin toteuttaa. Suurimmaksi ongelmaksi alustan siirrossa osoittautui mobiililaitteen tiedostojärjestelmän erilaisuus, minkä kohdalla jouduttiin pohtimaan erilaisia ratkaisuja, joihin ei pelkän ohjelmakoodin avulla voitu vaikuttaa. Pääpiirteissään ohjelma toimi melko hyvin. Tärkeimmät toiminnot saatiin toteutettua ja käyttöliittymä palveli tarkoitustaan. Silti ohjelmaan jäi jonkin verran jatkokehityksen varaa.

Web-tekniikoiden käyttö oli sinällään toimiva ratkaisu, mutta toisaalta työn olisi voinut tehdä myös eri tavoin. Sovellus olisi voitu toteuttaa natiivisovelluksena, jolloin ohjelmakoodi olisi kirjoitettu alustalle ominaisella ohjelmointikielellä. Toisena vaihtoehtona olisi voitu käyttää WebView-näkymää, joka on verkon tarjoamaa sisältöä näytettynä natiivisovelluksessa. Kolmas vaihtoehto olisi ollut käyttää PhoneGap-nimistä koodiympäristöä, jolla voidaan toteuttaa sovellus web-tekniikoin mutta sovellus pyörii natiivisovelluksen kuoressa. Vielä yhden vaihtoehdon olisi tarjonnut maksullinen Xamarin, jolla sovellus voidaan toteuttaa C#-kielellä iOS-, Android- ja Windows Phone-käyttöjärjestelmille. Näissä kaikissa oli omat vahvuutensa ja heikkoutensa web-tekniikoihin verrattuna.

LÄHTEET

Adobe PhoneGap Build. 2015. Home. Adobe PhoneGap Build. Viitattu 29.4.2015. https://build.phonegap.com/

Android Developers. 2015. App Manifest. Android Developers. Viitattu 20.3.2015. http://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developers. 2015. Building Web Apps in WebView. Android Developers. Viitattu 23.4.2015. http://developer.android.com/guide/webapps/webview.html

Android Developers. 2015. Migrating to WebView in Android 4.4. Android Developers. Viitattu 25.4.2015. http://developer.android.com/guide/webapps/migrating.html

Android Developers Reference. 2015. WebView Basic Usage. Android Developers. Android Developers. Viitattu 24.4.2015. http://developer.android.com/reference/android/webkit/WebView.html

Android Open Source. 2014. System and Kernel Security. Android Source. Viitattu 1.12.2014. https://source.android.com/devices/tech/security/overview/kernel-security.html

Apple. 2014. iOS File System Programming Guide. Apple. Viitattu 1.12.2014. https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html

Apple. 2014. iOS Human Interface Guidelines. Apple. Viitattu 13.11.2014. https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/

Brown, M. 2014. Why native app performance still beats HTML5. Developer Tech. Viitattu 30.10.2014. http://www.developertech.com/news/2014/feb/11/why-native-app-performance-still-beats-html5/

Brown, T. 2012. Opera Mini and JavaScript. Dev. Opera. Viitattu 27.2.2015. https://dev.opera.com/articles/opera-mini-and-javascript/

Budiu, R. 2013. Mobile: Native Apps, Web Apps, and Hybrid Apps. NN/g Nielsen Norman Group. Viitattu 16.4.2015. http://www.nngroup.com/articles/mobile-native-apps/

Chrome ohjekeskus. 2015. Chrome mobiililaitteille Android. Viitattu 16.2.2015. Chrome ohjekeskus. https://support.google.com/chrome/topic/3422804?hl=fi&ref_topic=3422738

CSS-Tricks. 2015. Media Queries for Standard Devices. CSS-Tricks. Viitattu 8.5.2015. https://css-tricks.com/snippets/css/media-queries-for-standard-devices/

Developer Chrome. 2015. Chrome for Android. Developer Chrome. Viitattu 20.2.2015. https://developer.chrome.com/multidevice/android/overview

Developer Economics. 2013. The Tools Report: Cross-platform Tools. Vision Mobile. Viitattu 27.4.2015. http://www.visionmobile.com/product/developer-economics-2013-the-tools-report/#

Firtman, M. 2012. iPhone 5 and iOS 6 for HTML5 developers, a big step forward: web inspector, new APIs and more. Mobilexweb. Viitattu 3.4.2015. http://www.mobilexweb.com/blog/iphone-5-ios-6-html5-developers

Foley, M.J. 2014. Microsoft makes Windows Phone 8.1 file manager available. ZDNet. Viitattu 1.12.2014. http://www.zdnet.com/article/microsoft-makes-windows-phone-8-1-file-manager-available/

Foley, M.J. 2015. Microsoft's new browser: It's 'Edge'. ZDNet. Viitattu 6.5.2015. http://www.zdnet.com/article/microsofts-new-browser-its-edge/

HTML5 compatibility on mobile and tablet browsers with testing on real devices. 2014. Mobile HTML5. Viitattu 30.10.2014. http://mobilehtml5.org/

Jobs, S. 2010. Thoughts on Flash. Apple. Viitattu 10.11.2014. https://www.apple.com/hotnews/thoughts-on-flash/

Ludwig, A. 2015. Blogikirjoitus. Google Plus. Viitattu 25.4.2015. https://plus.google.com/+AdrianLudwig/posts/1md7ruEwBLF

Mahemoff, M. 2010. ”Offline”: What does it mean and why should I care? HTML5 Rocks. Viitattu 18.4.2015. http://www.html5rocks.com/en/tutorials/offline/whats-offline/

Mahemoff, M. 2011. HTML5 vs Native: The Mobile App Debate. HTML5 Rocks. Viitattu 2.11.2014. http://www.html5rocks.com/en/mobile/native-debate/

McFarland, D.S. 2009. CSS: The Missing Manual. Kalifornia. O’Reilly.

Methvin, D. 2014. jQuery’s Content Delivery Network: You Got Served! jQuery Blog. Viitattu 3.4.2015. http://blog.jquery.com/2014/01/14/jquerys-content-delivery-network-you-got-served

Mohta, A. 2014. 10 Things to Know about IE on Windows Phone 8.1. WPXBOX. Viitattu 3.4.2015. http://www.wpxbox.com/ie-features-wp-8-1/

Mozilla Developer Network. 2015. Mozilla. Viitattu 20.2.2015. https://developer.mozilla.org/en-US/

Newman, J. 2015. 'Project Spartan' no more: Microsoft's new browser is called Edge. PCWorld. Viitattu 6.5.2015. http://www.pcworld.com/article/2916480/project-spartan-no-more-microsofts-new-browser-is-called-edge.html

Opera Software. 2015. Opera Mini. Opera. Viitattu 20.2.2015. http://www.opera.com/fi/mobile/mini/android

Opera Software. 2015. Opera Version History. Opera. Viitattu 27.2.2015. http://www.opera.com/docs/history/

PhoneGap. 2015. PhoneGap Docs: Overview. PhoneGap. Viitattu 27.4.2015. http://docs.phonegap.com/en/4.0.0/guide_overview_index.md.html#Overview

PhoneGap. 2015. PhoneGap Docs: Windows Platform Guide. PhoneGap. Viitattu 27.4.2015. http://docs.phonegap.com/en/4.0.0/guide_platforms_win8_index.md.html#Windows%20Platform%20Guide

Salminen, V. 2012. File Support on Mobile. VS. Viitattu 23.3.2015. http://viljamis.com/blog/2012/file-upload-support-on-mobile/

Traeg, P. 2014. Four Ways to Build A Mobile Application, Part 3: PhoneGap. Smashing Magazine. Viitattu 27.4.2015. http://www.smashingmagazine.com/2014/02/11/four-ways-to-build-a-mobile-app-part3-phonegap/

Van Hoven, N. 2015. What are CSS Media Queries and how to implement them. CSS Media Queries. Viitattu 8.5.2015. http://cssmediaqueries.com/what-are-css-media-queries.html

W3C. 2014. HTML5 is a W3C recommendation. W3C. Viitattu 30.10.2014. http://www.w3.org/blog/news/archives/4167

Winokur, D. 2011. Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5. Adobe Blogs. Viitattu 10.11.2014. http://blogs.adobe.com/conversations/2011/11/flash-focus.html

Xamarin. 2015. Guides: Getting Started. Xamarin. Viitattu 3.5.2015. http://developer.xamarin.com/guides/cross-platform/getting_started/

Xamarin. 2015. Guides: Sharing Code Options. Xamarin. Viitattu 4.5.2015. http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/sharing_code_options/#Shared_Projects

Xamarin. 2015. Meet Xamarin Test Cloud. Xamarin. Viitattu 6.5.2015. http://xamarin.com/test-cloud

Xamarin. 2015. Platform. Xamarin. Viitattu 4.5.2015. http://xamarin.com/platform

Xamarin. 2015. Simple and transparent pricing. Xamarin. Viitattu 4.5.2015. https://store.xamarin.com/